Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

FDM: Effective and efficient incident detection on sparse trajectory data

Xiaolin Han [a], Tobias Grubenmann [b], Chenhao Ma [c,*], Xiaodong Li [d], Wenya Sun [d], Sze Chun Wong [d], Xuequn Shang [a], Reynold Cheng [d,e,f]

[a] The Northwestern Polytechnical University, Xi'an, China
[b] Edinburgh Napier University, Edinburgh, United Kingdom
[c] The Chinese University of Hong Kong, Shenzhen, China
[d] The University of Hong Kong, Hong Kong Special Administrative Region of China
[e] HKU Musketeers Foundation Institute of Data Science, Hong Kong
[f] Guangdong–Hong Kong-Macau Joint Laboratory Program 2020, Hong Kong

ARTICLE INFO

Keywords:
Trajectory data mining
Traffic incident detection
Sparsity

ABSTRACT

Incident detection (ID), or the automatic discovery of anomalies from road traffic data (e.g., road sensor and GPS data), enables emergency actions (e.g., rescuing injured people) to be carried out in a timely fashion. Existing ID solutions based on data mining or machine learning often rely on *dense* traffic data; for instance, sensors installed in highways provide frequent updates of road information. In this paper, we ask the question: can ID be performed on *sparse* traffic data (e.g., location data obtained from GPS devices equipped on vehicles)? As these data may not be enough to describe the state of the roads involved, they can undermine the effectiveness of existing ID solutions. To tackle this challenge, we borrow an important insight from the transportation area, which uses trajectories (i.e., moving histories of vehicles) to derive *incident patterns*. We study how to obtain incident patterns from trajectories and devise a new solution (called Filter-Discovery-Match (**FDM**)) to detect anomalies in sparse traffic data. We have also developed a fast algorithm to support FDM. Experiments on a taxi dataset in Hong Kong and a simulated dataset show that FDM is more effective than state-of-the-art ID solutions on sparse traffic data, and is also efficient.

1. Introduction

Advances in traffic data acquisition technologies (e.g., loop sensors, road detectors, Global Positioning System (GPS), and CCTV cameras) provide gigantic amounts of Big Transportation Data (BTD) in real-time [1–4]. These data (e.g., road traffic conditions, vehicle locations, speeds, pedestrian information) enable urban computing applications (e.g., autonomous vehicles, intelligent navigation, smart traffic light control, and air pollution reduction), with the goal of improving transportation conditions and living quality of citizens [5–8]. A fundamental problem in BTD is the automatic discovery of incidents (e.g., roadblock and traffic incidents). Particularly, a number of *incident detection* (ID) algorithms have been proposed (e.g., [9–12]), which perform mining and machine learning on BTD, identify abnormal traffic states, so that appropriate actions could be taken (e.g., dispatch medical and police resources to prevent life loss, or detour drivers to incident-free routes to avoid congestion).

For existing ID algorithms, the underlying traffic data is often assumed to be *dense* – i.e., each road involved is covered by huge volumes of traffic data (e.g., vehicle locations). This is a reasonable assumption for freeways, where fixed equipment such as road detectors and CCTV cameras are installed to acquire traffic information regularly. In fact, most experiments on existing ID solutions [9–12] are conducted on freeways. However, it is doubtful whether these solutions are effective on *urban roads* (i.e., roads in cities with many neighboring junctions and traffic lights), where road detectors may be rare. For example, in Hong Kong, 1210 road detectors have been installed until the end of 2020, covering only 6.2% of all the roads.[1] A road detector, which can be more than US$ 1K, may not be deployed to cover the whole metropolitan city with dense road networks.[2]

An alternative BTD source whose data provide a higher coverage of roads is GPS data. Due to the low costs of GPS devices, they are commonly found in many smart phones and vehicles. They are much less costly than road detectors, and the positions of vehicles can be

* Corresponding author.
E-mail addresses: xiaolinh@nwpu.edu.cn (X. Han), t.grubenmann@napier.ac.uk (T. Grubenmann), machenhao@cuhk.edu.cn (C. Ma), xdli@cs.hku.hk (X. Li), wysun@cs.hku.hk (W. Sun), hhecwsc@hku.hk (S.C. Wong), shang@nwpu.edu.cn (X. Shang), ckcheng@cs.hku.hk (R. Cheng).

[1] https://www.td.gov.hk/en/transport_in_hong_kong/its/intelligent_transport_systems_strategy_review_and_/traffic_detectors/index.html.
[2] https://www.itscosts.its.dot.gov/ITS/benecost.nsf/0/F416B93F3196D5C185257B1E0054A440?OpenDocument.

https://doi.org/10.1016/j.is.2024.102418
Received 23 May 2023; Received in revised form 12 May 2024; Accepted 31 May 2024
Available online 1 June 2024

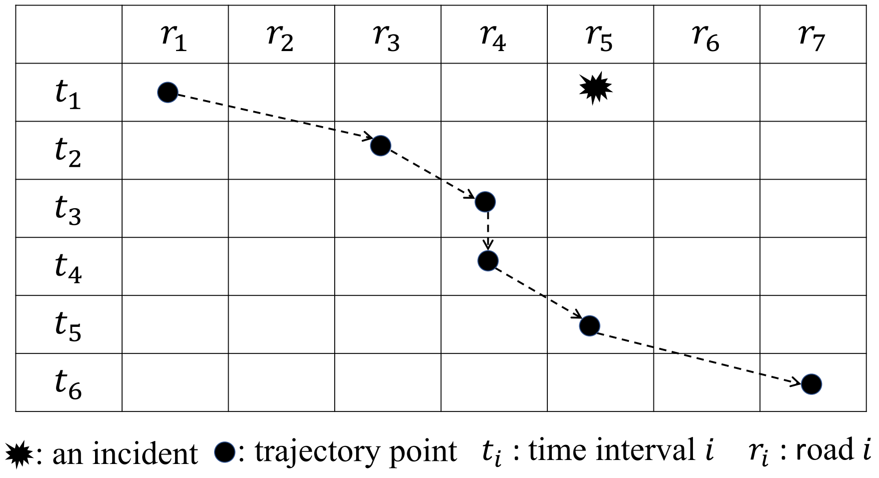
r_1 r_2 r_3 r_4 r_5 r_6 r_7
t_1 t_2 t_3 t_4 t_5 t_6
✹: an incident ●: trajectory point t_i : time interval i r_i : road i

Fig. 1. Illustrating a trajectory that drives around an incident spot.

acquired by GPS devices in real-time. Hence, by using GPS data, the traffic state of urban roads can be obtained. However, GPS data may not be readily available, which can be due to privacy reasons (e.g., a mobile phone service provider is unlikely to share their customer information), or commercial reasons (e.g., a company like Google that owns GPS data may see it as a valuable asset). Oftentimes, companies that provide GPS equipment to their vehicles have GPS data. In Didi, one of the largest share-riding companies in China, data is acquired from the GPS equipment installed in their own vehicles. We have also collected GPS data from the vehicles owned by a taxi company in Hong Kong. A problem common to these data sources is that they may not cover all the roads with the same amount of data. For example, roads located in commercial districts are traveled more frequently than other roads. For roads in residential or suburb areas, traffic data can be *sparse*.

In this paper, we investigate the question: can ID be performed on sparse traffic (e.g., GPS data)? Experiments on GPS data provided by a taxi company show that sparse traffic renders existing ID solutions less effective. The main reason is that these data mining or machine learning solutions often require a huge amount of traffic data over both spatial and temporal dimensions. They utilize the difference between spatial and temporal features to detect incidents, since these features deviate in a large degree once an incident happens. However, not all spatial and temporal features' differences may be available when only a few trajectories are contained in the dataset. For example, in Fig. 1, only one trajectory is passing through the incident location at road r_5 within six time steps $t_1, \dots, t_6$. Since the spatial features f_{t_2,r_4} of r_4 and f_{t_2,r_6} of r_6 at the time interval t_2 are missing, we cannot obtain the spatial feature difference $f_{t_2,r_4} - f_{t_2,r_6}$. Our research question is: *can we still detect incidents when trajectories are sparse in the spatial and temporal dimensions?* This paper is an extension of our previous work [13].

To tackle this challenge, we borrow an insight from the transportation community, which makes use of the movement history (or *trajectory*) of a vehicle to derive a *speed pattern* (i.e., an observation of the speed of a vehicle over a certain period of time) [14]. As pointed out by [14], when a vehicle passes through a location where an incident occurs, these speed patterns can be used to detect incidents effectively. Based on this intuition, we develop a solution called Filter-Discovery-Match (**FDM**). The main idea of FDM is to use speed patterns to identify anomalies. Specifically, we use the divergence between the speed pattern of the current vehicle and those patterns that are not affected by incidents to find out whether a vehicle is passing the scene of an incident. In sparse settings where roads do not receive a lot of traffic data, FDM yields a lower mean time-to-detect (MTTD) than existing solutions.

The MTTD measures the time between the occurrence of an incident and the detection of this incident by the ID algorithm. This includes the time until all necessary data is available and the time to actually run the ID algorithm. If we focus on a single incident, we often have to wait for several minutes before the observed vehicles have produced enough data for a successful detection of the incident. In contrast, running the ID algorithm often requires only a few seconds. In light of this observation, one might argue that the runtime of the ID algorithm is of lower priority, as the MTTD is dominated by the waiting time until all necessary data is available. However, this is only true as long as we focus on detecting a single incident. Indeed, as soon as we wish to continuously observe a large road network, the runtime of the ID algorithm becomes an important factor. If the runtime of our algorithm is too slow, we may not be able to process all the available data in real-time. As FDM requires extensive analysis over trajectory data, the computational cost of our solution can be quite high. To enable fast detection of incidents, we develop a fast detection algorithm. It compresses the patterns behind incidents by a hierarchical pattern tree. When checking whether an upcoming trajectory is associated with an incident, we can stop the computation early based on a data structure we developed, called the hierarchical pattern tree. Our experiments on a large taxi dataset in Hong Kong and a simulated dataset show that (1) FDM is more effective than existing ID algorithms, and (2) FDM is computationally efficient, which is on average 38 times faster than the original detection algorithm without speeding up.

The rest of this paper is organized as follows. Section 2 discusses related work. In Section 3, we introduce the definition of incident detection and present our solution, named FDM. In Section 4, we propose an efficient algorithm to facilitate the incident detection. Section 5 describes the experimental results. Finally, Section 7 concludes our paper.

2. Related work

Previous work on traffic incident detection can be divided into three categories: pattern recognition, deviation detection, and machine learning. Most of these categories provide methods for incident detection on freeways. As the data is collected from static detectors installed along freeway roads, these algorithms generally assume that the data is updated at very short time intervals for each individual road segment. This assumption limits their applicability for incident detection on urban roads using GPS-equipped vehicles, as the time between two successive vehicles traveling through some road segments can be very large. We call this the *data sparsity problem in urban roads*.

Pattern Recognition: California [15] and DELOS [16] make use of the occupancy difference in the spatial and temporal dimensions to detect incidents. In their experiments, the data is updated in one-minute and thirty-seconds intervals, respectively. Such short intervals are unrealistic for urban roads, however, since many urban road segments are not covered by even one vehicle during some time intervals. To adapt their solution to GPS signals collected by vehicles on urban roads, aggregation over longer time intervals is necessary. This can lead to the problem of a high mean time-to-detect (MTTD).

Deviation Detection: Time series [17] and nonparametric regression [18] are used to predict the road occupancy or traffic volume at a specific time interval t. A traffic situation is classified as an incident when the deviation between the predicted value and the observed value is higher than a user-defined threshold. However, there are many missing values in urban roads due to data sparsity. The forecasted values based on many missing values can be inaccurate and can lead to low detection rates. Particle filters [19] are utilized to estimate the traffic state, which can be used to detect incidents. However, real-time estimation is not feasible when the number of sensors is limited. Traffic state estimation [20] uses the spatial–temporal feature deviation to estimate the traffic state and to detect incidents using these states. However, the estimated traffic state can be biased when the GPS data is limited. Probabilistic topic modeling [21] uses the Latent Dirichlet Allocation (LDA) equivalent model [22] to calculate the divergence of the current traffic state from the normal traffic state. A traffic state is classified as an incident when the divergence exceeds a user-defined threshold. However, it requires mass data to estimate the real distribution of data. As all of these methods are designed for scenarios with dense datasets, their performance can be negatively affected by the sparsity of GPS data on urban roads.

Machine Learning: Neural network (NN) based methods [9,11] use the upstream and downstream of traffic volume, road occupancy, and traffic speed at different time intervals as features to train NN based models. Support vector machines (SVM) [10,12] use the same features to detect incidents. Wong and Wong [12] show that SVM models can outperform NN-based models. Convolutional Neural Networks (CNN) [23] are utilized to detect incidents on traffic flow data aggregated over 5 min intervals. However, all these methods are designed for freeways and may perform poorly when applied to urban roads, which generate much sparser vehicular data. One could address this problem by aggregating data over a longer time period, but that would significantly increase the MTTD and may render these methods impractical. In [24], a time series method is used to forecast normal traffic and SVM is used to learn the difference between current traffic and normal traffic. However, the time series method does not perform well under data sparsity, as the regular traffic behavior has to be estimated due to the lack of available data. In [25], a heat diffusion model is used to model the propagation of traffic anomalies along road networks. However, the diffusion model cannot be fitted well based on sparse data. Multivariate time series classification [26] utilizes incident impact intervals to detect incidents on freeways. However, these impact intervals cannot be extracted due to many missing values in sparse trajectories on urban roads.

We summarize in Table 1 the state-of-the-art techniques which can be applied to the urban incident detection problem.

Handling Sparsity: In [27–30], decomposition techniques are utilized to decompose a matrix or tensor into a product of low-rank matrices and use these decomposed matrices to estimate missing values. If the matrices and tensors are very sparse, estimations based on these few data are unsatisfactory. To overcome this problem, they introduce a context to borrow more information from other data sources when decomposing. Unfortunately, this context is often very domain-specific and does not generalize well since different data is used in different scenarios.

3. Filter-Discovery-Match (FDM)

In this section, we first give definitions of the basic concepts and the problem (Section 3.1). Then, we introduce the incident patterns, which are the basic building blocks for incident detection (Section 3.2). After that, our FDM method is further explained in Section 3.3. We summarize the notations of our method in Table 2.

Table 1
Comparison of other traffic incident detection methods.

Category	Methods	Handle sparsity	Low MTTD
Pattern recognition	California [15] DELOS [16]	No	No
Deviation detection	Time series [17,20] Regression [18] Topic model [21]	No	Yes
Machine learning	SVM [10,12] Hybrid method [24] Heat diffusion [25] MTS [26] NN [9,11,23]	No	No

Table 2
Table of notations.

Notation	Description
T	A trajectory
P_{t_i}	A trajectory point at time t_i
r	A road
r_i	The i-th road segment
$\overrightarrow{S}$	A speed vector
m	The size of the speed vector
$\overrightarrow{S_a}$	An incident speed vector
τ	The threshold for defining an incident speed vector
$\overrightarrow{S_n}$	A normal speed vector
$R(t_j)$	A set of reference time points for time t_j
$\overrightarrow{S_T}$	The speed vector for trajectory T
$\overrightarrow{S_n^{nor}}$	The normalized normal speed vector
$\overrightarrow{S_a^{nor}}$	The normalized incident speed vector
$\overrightarrow{P}$	A pattern
$\overrightarrow{P_{a_i}}$	An incident pattern, where $i \in \{1, \dots, k\}$
$\overrightarrow{P_T}$	The pattern for trajectory T

3.1. Problem definition

As discussed previously, the core purpose of this paper is to use GPS trajectories for incident detection. A GPS device is continuously tracking its location. Due to technical limitations, the GPS locations are only tracked at discrete timestamps, which results in an approximation of a GPS trajectory using a finite number of GPS points.

Definition 1 (*Trajectory*)**.** A trajectory is an ordered sequence of points $\langle p_{t_1}, p_{t_2}, p_{t_3}, \dots, p_{t_n} \rangle$ where $t_1 < t_2 < \cdots < t_n$. A point p_{t_n} is a three-tuple (t_n, x, y) which contains a timestamp t_n, the latitude x and longitude y of its position at t_n. $\square$

To facilitate the analysis of a road network, roads are usually subdivided into smaller partitions, which we call *road segments*.

Definition 2 (*Road Segment*)**.** Given a road r, we subdivide the road into disjoint *road segments* $r_1, \dots, r_i, \dots, r_m$ such that $r = \bigcup_{i=1}^{m} r_i$. $\square$

The motivation of using road segments is to partition a road network into smaller spatial units of (approximately) equal length, as long road edges cannot be directly compared to short road edges due to the high variability fo road length.

Definition 3 (*Speed Vector*)**.** Let $r_1, \dots, r_m$ be a sequence of road segments and $t_1, \dots, t_m$ with $t_1 < t_2 < \cdots < t_m$ a sequence of time steps. We define the *speed vector* $\overrightarrow{S} = (S_{r_1,t_1}, \dots, S_{r_m,t_m})$ as the vector of speed values S_{r_i,t_i} of the speed of a vehicle which travels through road segment r_i at time t_i. $\square$

The speed for a given point p_{t_i} can be directly obtained from the GPS device. The speed information is an additional feature which is

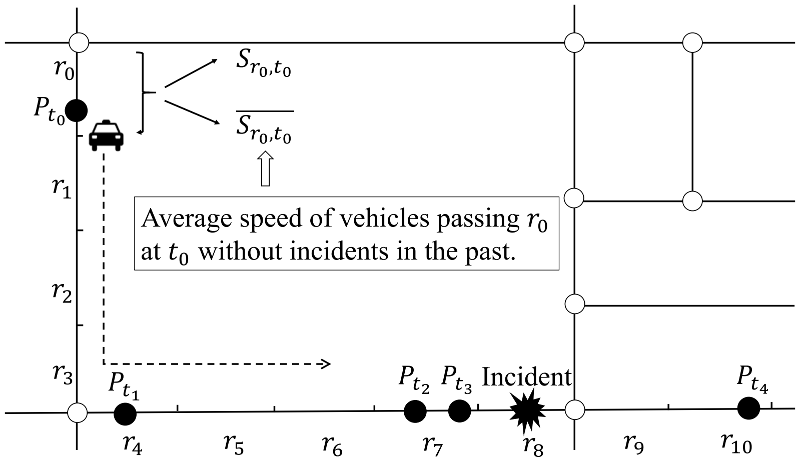

Fig. 2. From trajectory to speed vector.

provided along the location. In our method, we make use of both, location of a vehicle as well as the speed. By using the speed as an additional feature, we are able to perform better incident detection, as we show in Section 5. Based on the definition of the speed vector, we can define the *incident speed vector*. An incident speed vector is a speed vector of a vehicle which passes through an incident location after an incident.

Definition 4 (*Incident Speed Vector*). Let $\overrightarrow{S_a} = (S_{r_i,t_i}, \dots, S_{r_{i+m-1},t_{i+m-1}})$ be a speed vector. The speed vector $\overrightarrow{S_a}$ is an *incident speed vector* if there exists an incident in a road segment r_{inc} at time t_{inc} and a speed value $S_{r_a,t_a} \in \overrightarrow{S_a}$ s.t. $r_a = r_{\text{inc}}$ and $t_a \in [t_{\text{inc}}, t_{\text{inc}} + \tau]$ for a given threshold $\tau \in \mathbb{R}_+$. □

In Fig. 1, for example, the speed vector corresponding to the trajectory is an incident speed vector if we set $\tau \geq t_5 - t_1$.

To classify this incident speed vector, we need a reference for comparison. For this, we use the *normal speed vector*, which is defined as follows:

Definition 5 (*Normal Speed Vector*). Given sequence of road segments $r_i, r_{i+1}, \dots, r_{i+m-1}$ and the time sequence $t_i < t_{i+1} < \cdots < t_{i+m-1}$. Let $R(t_j)$ be a set of reference time points for $j \in [i, i+m-1]$. The normal average speed $\overline{S_{r_i,t_i}}$ of r_i at t_i is the average speed of all trajectories which passed through r_i at some time point $t \in R(t_j)$ and are not classified as incident speed vectors given a threshold $\tau \in \mathbb{R}_+$. The normal speed vector $\overrightarrow{S_n}$ is defined as $\overrightarrow{S_n} = (\overline{S_{r_i,t_i}}, \overline{S_{r_{i+1},t_{i+1}}}, \dots, \overline{S_{r_{i+m-1},t_{i+m-1}}})$. □

The reference set $R(t_j)$ defines how trajectories are aggregated for the normal speed vectors. A typical choice is to define $R(t_j)$ as all the time points which have the same time of the day, or the same weekday and time of the day, as t_j.

With these definitions, we can now formulate the problem definition of our paper:

Incident Detection: The aim of real-time traffic incident detection with sparse trajectories is to raise an alarm when an incident a happens on a road segment r_i at the current time t by comparing the speed vector $\overrightarrow{S_a}$ of by-passing vehicle with the corresponding normal speed vector $\overrightarrow{S_n}$. If there is a heavy deviation between $\overrightarrow{S_a}$ and $\overrightarrow{S_n}$, and the pattern of the deviation is similar as the one caused by incidents, then incidents are detected. The incident detection should be performed in real-time, i.e., the detection algorithm should scan the road network continuously and dynamically adapt to changing traffic conditions in order to detect an incident as soon as possible.

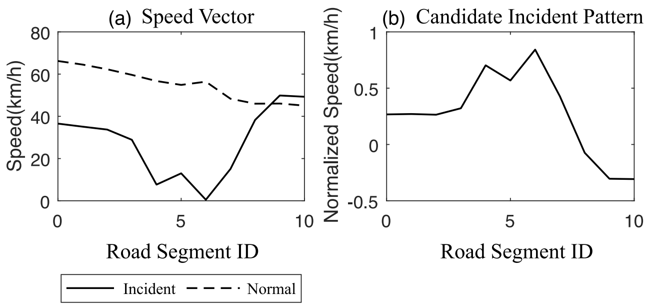

Fig. 3. From speed vector to candidate incident pattern.

3.2. Incident patterns

Experts from the transportation field [14] evaluated the impact of traffic incidents using trajectories, finding that vehicles first reduce their speed, maintain that speed for a certain time-interval, and then finally increase in speed, having passed the incident locations. However, in [14], they did not generalize these incident patterns which would allow one to detect incidents. We discovered that a lot of incidents do not strictly follow the pattern found in [14]. Hence, we developed a more data-driven approach to extract these patterns. In Fig. 2, we illustrate the trace of one vehicle v passing through the incident location from road segment r_0 until r_{10} in the road network, where the sequence of black dots $\langle p_{t_0}, p_{t_1}, p_{t_2}, p_{t_3}, p_{t_4} \rangle$ represent one trajectory and the black explosion mark stands for an incident. For each road segment r_i, we have a time t_i when the vehicle passes through r_i and a corresponding speed S_{r_i,t_i}. To get the normal speed $\overline{S_{r_i,t_i}}$ on r_i at t_i, we aggregate the trajectories which passed through r_i at some time $t \in R(t_i)$ without being involved in any incident (according to our threshold τ). To limit our search, we only track m road segments in our sliding window. For example, in Fig. 2, $m = 11$. Our idea is to discover incident patterns by comparing the incident speed vector and the normal speed vector as the incident impacts the speed of vehicle v which passes through the incident location.

Once we obtain the incident speed vector and the normal speed vector, we can use the difference between them as the *candidate incident pattern*. We will give a formal definition of the candidate incident pattern later. In Fig. 3(a), the dashed line represents the normal speed vector while the solid line represents the incident speed vector. The Y-axis represents the speed, and the X-axis represents the road segments that a vehicle passes through. In Fig. 3(b), the Y-axis represents the normalized speed. The normalization procedure will be introduced in

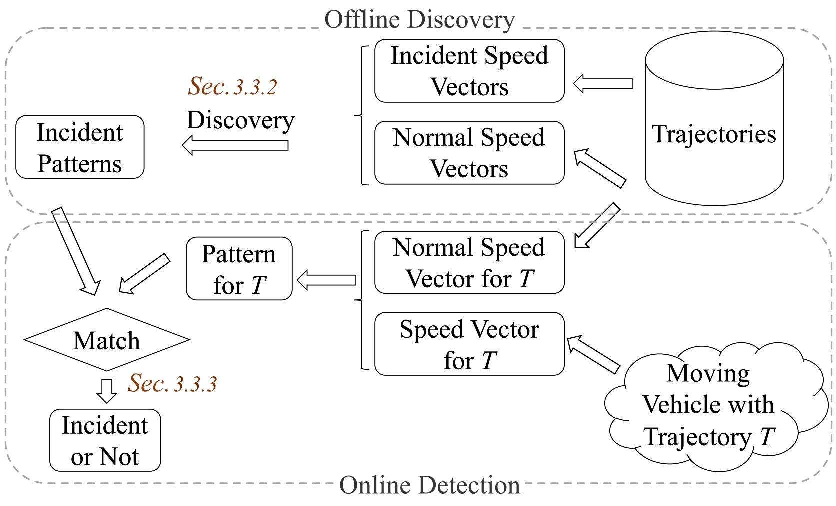

Fig. 4. Overview of **FDM**.

Section 3.3.1. The line represents the candidate incident pattern, which is obtained from the difference between the normalized normal speed vector and the normalized incident speed vector. From Fig. 3(b), we can see that the candidate incident pattern first increases in trend, then fluctuates a little, and finally decreases.

Based on this observation, we propose a model to group candidate incident patterns by clustering them and choose the centers of clusters as the incident patterns. Then, these incident patterns can be treated as templates for real-time detection in the online phase. If the pattern for a speed vector of a real-time trajectory is close to one of our discovered incident patterns, an alarm is triggered. If no alarm has been triggered, we conclude that no incident happened at this specific time and location. Our method not only detects incidents in real-time, but also returns the matched incident patterns as further evidence to the user.

3.3. Method

Challenges. A trajectory may pass through many road segments, e.g., 100 road segments, but only a few segments are affected by an incident, e.g., 4 road segments. If we need to detect the incident, we should only consider the segments affected by the incident. If we simply detect the patterns on the whole trajectory, it does not make sense, because most road segments cannot observe any incident-related patterns. It is quite challenging to extract the pattern behind these road segments affected by the incidents. We use a sliding window with a fixed size m to slide the coming trajectories. So even though there is a trajectory passing through many road segments, what we are doing is using the sliding window with the size m to calculate the distance between it and the incident patterns. When the sliding window covers these road segments that are affected by an incident, our method can capture the small distance and then detect the incident. The sliding window leads to an online process, in which we can keep tracking trajectories generated in a streaming mode.

Therefore, we propose a method to detect incidents with a sliding window. The method FDM is named after three main steps: noise **f**iltering, incident pattern **d**iscovery, and incident pattern **m**atching. The first two steps are utilized to extract incident patterns during the offline phase and the last step is applied for the online detection phase. We depict the overview of FDM in Fig. 4. We will explain the individual parts of Fig. 4 throughout this section.

3.3.1. *Noise filtering*

First, we extract incident speed vectors from trajectories that pass through incident locations when an incident occurred, according to Definition 4. The corresponding normal speed vectors are obtained by taking the average speed of those vehicles which passed through the same road segments at the corresponding reference time points, according to Definition 5. This step is depicted in the upper right part of Fig. 4.

When comparing a speed vector with the normal speed vector, we are not interested in the absolute difference between the two vectors but in how their speed vectors differ. Thus, in order to compare the speed trend, we first have to normalize both speed vectors. For normalization, we scale both speed vectors such that their maximum value equals to 1. Eq. (1) shows the normalization formula.

$$\overrightarrow{S_{nor}} = \frac{\vec{S}}{\max(\vec{S})} \tag{1}$$

Using such a normalization strategy allows us to mitigate the general speed fluctuations and to concentrate on the trend of the vectors in question. In addition, the normalization also allows us to generalize the discovered patterns to other districts, which might have different speed limits and average speeds. Fig. 5 shows the normal speed vector (dashed line) and the incident speed vector (solid line) before and after the normalization. The normal speed vector $\overrightarrow{S_n}$ and the incident speed vector $\overrightarrow{S_a}$ have a similar trend but their distance is larger before normalization (left-hand-side of Fig. 5) than after normalization (righthand-side of Fig. 5). As we can see, after normalization, these two speed vectors are quite similar in their trend.

After the normalization step, we filter out incident vectors which are too similar to their corresponding normal speed vector. This step is necessary as not all vehicles passing through an incident location are actually negatively affected by the incident (e.g., the incident might already be resolved by the time the vehicle passes by, or the incident was not severe enough to affect the traffic). Thus, we use a filter method to ignore incident speed vectors which do not exhibit a strong enough deviation from the normal speed vector. For filtering, we compare the L_1 similarity against a given distance threshold δ. Incident speed vectors with a L_1 smaller than δ are not included into the generation of the incident patterns.

3.3.2. *Incident pattern discovery*

In the following, we first give the definitions of patterns and candidate incident patterns, and then we show the process of pattern discovery.

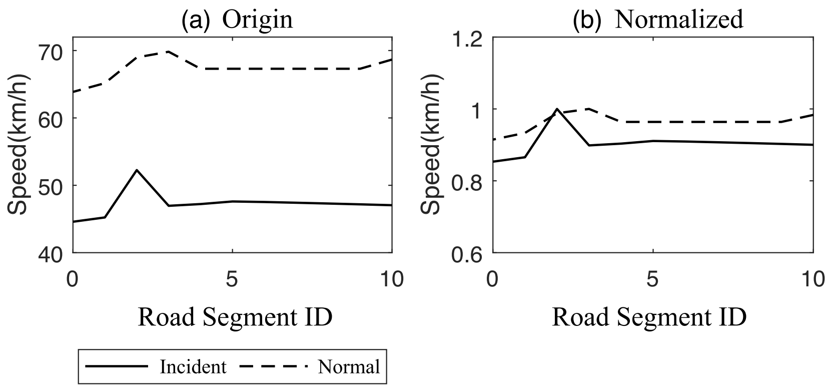

Fig. 5. Illustration of similar trends.

Definition 6 (*Pattern*)**.** Given a normalized incident speed vector $\overrightarrow{S_a^{\text{nor}}}$ and a normalized normal speed vector $\overrightarrow{S_n^{\text{nor}}}$, we define a pattern as the vector $\vec{P} = \overrightarrow{S_n^{\text{nor}}} - \overrightarrow{S_a^{\text{nor}}}$. □

Definition 7 (*Candidate Incident Pattern*)**.** Given a filtering threshold $\delta \in \mathbb{R}_+$, a candidate pattern $\overrightarrow{P_{cd}}$ is a pattern $\vec{P}$ s.t. $L_1(\overrightarrow{S_n^{\text{nor}}}, \overrightarrow{S_a^{\text{nor}}}) \geq \delta$. □

To obtain the incident patterns $\overrightarrow{P_{a_1}}, \ldots, \overrightarrow{P_{a_k}}$ from the set of all candidate patterns $\overrightarrow{P_{cd}}$, we group them into $k \in \mathbb{N}$ clusters w.r.t L_1 similarity and choose the center of these k clusters as the incident patterns. For clustering, we use K-means. The clustering process which results in the incident patterns is depicted in the upper left part of Fig. 4.

3.3.3. Pattern matching

After we have obtained the incident patterns $\overrightarrow{P_{a_1}}, \ldots, \overrightarrow{P_{a_k}}$ from the offline discovery phase, we can use these incident patterns to classify new patterns observed in the online phase (lower part of Fig. 4). We will now formally define the Speed Vector for Trajectory T and the Pattern for Trajectory T, which are used for the online pattern matching.

Definition 8 (*Speed Vector for Trajectory T*)**.** Let $T = \langle p_{t_1}, p_{t_2}, p_{t_3}, \ldots, p_{t_n} \rangle$ be a trajectory passing through road segments $r_i, r_{i+1}, \ldots, r_{i+m-1}$ during the time sequence $t_i, t_{i+1}, \ldots, t_{i+m-1}$. The speed for road segment r_i at time t_i is defined in the following way: for each segment r_i, we define the speed S_{r_i,t_i} as the interpolated speed at the center of this road segment r_i using the speed of the trajectory $\langle p_{t_1}, p_{t_2}, p_{t_3}, \ldots, p_{t_n} \rangle$. The *Speed Vector for Trajectory* T is defined as $\overrightarrow{S_T} = (S_{r_i,t_i}, S_{r_{i+1},t_{i+1}}, \ldots, S_{r_{i+m-1},t_{i+m-1}})$. □

In the online phase, after we obtain the speed vector $\overrightarrow{S_T}$ for trajectory T and its corresponding normal speed vector $\overrightarrow{S_n}$, we apply the normalization from Eq. (1) to obtain $\overrightarrow{S_T^{\text{nor}}}$ and $\overrightarrow{S_n^{\text{nor}}}$.

Definition 9 (*Pattern for Trajectory T*)**.** Given a normalized speed vector $\overrightarrow{S_T^{\text{nor}}}$ for T and the corresponding normalized normal speed vector $\overrightarrow{S_n^{\text{nor}}}$, we define the pattern for T as the vector $\overrightarrow{P_T} = \overrightarrow{S_n^{\text{nor}}} - \overrightarrow{S_T^{\text{nor}}}$. □

Given one pattern $\overrightarrow{P_T}$ for trajectory T, we measure the distance between it and the k incident patterns $\overrightarrow{P_{a_1}}, \ldots, \overrightarrow{P_{a_k}}$ using L_1 distance. If there exists one incident pattern $\overrightarrow{P_{a_i}}$ such that its distance to pattern $\overrightarrow{P_T}$ is smaller than the predefined threshold γ (condition in Eq. (2)), then we detect an incident. As long as we find one pattern $\overrightarrow{P_{a_i}}$ which satisfies the condition, we can stop the computation. This condition is different from kNN based algorithms, which must find the nearest neighbor first and then check the distance between the pattern $\overrightarrow{P_T}$ and the nearest neighbor.

$$\exists\ i\ :\ d(\overrightarrow{P_T}, \overrightarrow{P_{a_i}}) < \gamma\ , i \in \{1, 2, \ldots, k\}\ , \tag{2}$$

where $d(\overrightarrow{P_T}, \overrightarrow{P_{a_i}}) = \sum_{j=1}^{m} |\overrightarrow{P_{T_j}} - \overrightarrow{P_{a_{ij}}}|$.

Suppose that there are n patterns $\overrightarrow{P_T} \in \mathbb{R}^m$ generated by vehicles. Each $\overrightarrow{P_T}$ needs to be compared with k incident patterns $\overrightarrow{P_a} \in \mathbb{R}^m$. Hence, the resulting time complexity for the online detection is $O(nkm)$. This can be very expensive during online detection if many vehicles have to be tracked at the same time.

To speed up the process of the extraction of the normal speed vector for both offline discovery and online detection, we apply an index structure named *aRB-tree* [31]. In an *aRB-tree*, an R tree is used to index the spatial dimension, while a B tree is used to index the temporal information. Give a road segment r and time t, we traverse the R tree using r, then traverse the leaf node found in the R tree using t to obtain the averaged normal speed of r at t. Moreover, to improve the efficiency of our solutions, we propose exact and approximate algorithms to further speed up the matching process of patterns $\overrightarrow{P_T}$ in Section 4.

Note that our method is user-friendly, as incident patterns can be further visualized, judged, analyzed, and developed by experts from different backgrounds. This can be very beneficial, as a domain expert can double check the output of our method and intervene if it is considered to be necessary. This sets our method apart from other machine learning methods where the detection mechanism operates as a black box which gives its user only limited insight into why a certain situation was classified as an incident.

4. Efficient online detection of FDM

Given n vehicles which are tracked during the online phase and k incident patterns discovered during the offline phase, an incident detection algorithm has to perform $n \cdot k$ checks to match all n trajectories — one check for each trajectory for each of the k patterns. However, n and k can be large and an inefficient matching algorithm can slow down the online phase significantly. For example, in transportation data for Hong Kong,[3] the value of n can be 762,000 and k can be several hundred (see Fig. 9(a and c)) due to complex traffic conditions. It takes several minutes to match n trajectories with k incident patterns in our powerful server. Since the trajectories collected from vehicles are analyzed in real-time [20], the speed of the matching algorithm should be at most in the order of seconds to detect incidents as fast as possible. Therefore, we propose an exact and fast pattern matching algorithm to speed up online pattern matching.

4.1. Hierarchical pattern tree

We observe that there are some common prefixes between incident patterns we discovered, therefore, we develop a novel *hierarchical*

[3] https://www.hyd.gov.hk/en/road_and_railway/existing/road_network/road.html.

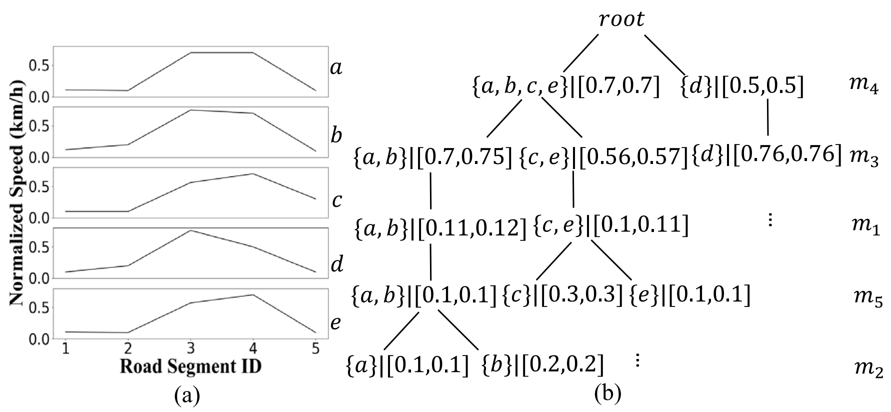

Fig. 6. Hierarchical pattern tree.

pattern tree to speed up the matching further. The hierarchical pattern tree organizes the incident patterns based on their similarities over dimensions, as illustrated in Fig. 6. On the left of the figure are incident patterns from a to e, and on the right part is the corresponding hierarchical pattern tree with the bounding rectangle R, which is denoted as $[R_L, R_U]$. Each node contains a set of incident patterns and the minimum value R_L and maximum value R_U of the bounding rectangle R at the ith dimension m_i. For instance, m_2 indicates that the nodes are partitioned by values at the second dimension. As the values of one dimension among incident patterns are not exactly the same, we introduce a relaxation factor ρ here. As long as the values' differences in one dimension among some incident patterns are within ρ, we group them within one bounding rectangle R. For instance, the values of the third dimension of pattern a, b are 0.7, 0.75, respectively. If ρ is 0.06, then a, b fall into the same branch since their values' difference is 0.05 which is smaller than ρ. Then the minimum value R_L equals 0.7, and maximum R_U equals 0.75.

Rescheduling the order of dimensions: There are some dimensions that have heavier fluctuations in the incident patterns than others. If we can first compare the pattern $\overrightarrow{P_T}$ for trajectory T with incident patterns over these highly fluctuating dimensions, we have higher chances to stop the computation earlier. Therefore, we further propose to reschedule the dimensions of incident patterns when constructing the hierarchical pattern tree, instead of treating dimensions in their original order.

The idea is to give higher priority to the dimensions with heavy fluctuation when conducting pattern matching. Another intuition is that the number of splits over each dimension should be minimal in order to take less computation over each dimension. The main ideas are as follows:

1. Dimensions with high fluctuations should be chosen before dimensions with lower fluctuation, when constructing the pattern tree.
2. Multiple splits over one dimension should be penalized.

To achieve a good ordering of dimensions, we assign a score s_i to each dimension m_i according to the following equation:

$$s_i = \frac{1}{b_i} \cdot \sum_{j=1}^{b} |m_{i,j} \cdot c_{i,j}| \ , \tag{3}$$

where b_i is the number of splits in dimension m_i, j is the jth split in dimension m_i, $m_{i,j}$ is the average value in the jth split of m_i, $c_{i,j}$ is the size of patterns falling into the jth split of m_i, $\frac{1}{b_i}$ is the penalty factor for a large number of splits. $\sum_{j=1}^{b} |m_{i,j} \cdot c_{i,j}|$ calculates the fluctuation of patterns in m_i. Since negative values exist in the value of patterns, we use the absolute value to evaluate the fluctuation. The larger the value of $|c_{i,j}|$ is, the greater the fluctuation of the speed pattern, because if the speed vector remains close to the normal speed vector $|c_{i,j}|$ should be close to zero. We use the order of dimensions in the ranking of s_i to construct the hierarchical pattern tree, and represent the ith dimension after rescheduling as m_{o_i}.

Algorithm 1: Hierarchical Pattern Tree

```
function constructTree ($p, m, i, S_{\text{index}}, P_{\text{a}}, \rho$);
Input : tree node $p$, total number of dimensions $m$, count $i$, index set
        $S_{\text{index}}$, matrix of incident patterns $P_{\text{a}}$, factor $\rho$
if $i = m$ then return;
$curDim \leftarrow$ nextDimByRescheduling();
$S_{\text{sub}} \leftarrow$ split($\rho$, $P_{\text{a}}$, $curDim$, $S_{\text{index}}$);
for every $s_j$ in $S_{\text{sub}}$ do
    $c \leftarrow p$.createNode();
    $c.idList$.extend($s_j$);
    $c.m_{o_i} \leftarrow curDim$;
    $c.R_L, c.R_U \leftarrow$ getNode_Boundary($c$);
    constructTree($c, m, i+1, s_j, P_{\text{a}}, \rho$);
    $p.children$.append($c$);
    $c.p = p$;
```

We show the detailed process of constructing the hierarchical pattern tree in Algorithm 1, which is a recursive algorithm. If i equals to the total number of dimensions m, then the current recursion returns (Line 2). Otherwise, we find the next best dimension $curDim$ by Eq. (3). Then we apply the split method to split over the $curDim$-th dimension with factor ρ. For the $curDim$-th dimension, if the range in the $curDim$-th dimension of adding one pattern p_k to one set of patterns P is within ρ, then we add p_k to P. Then, we set S_{sub} to be the list of the partitioned set which contains the IDs of incident patterns in the same partition (Line 4). For each item e_j in S_{sub}, we create a child node and recursively build the tree on the child node (Lines 5–11). Specifically, for each item s_j, we calculate the minimum value R_L and maximum value R_U in the $curDim$-th dimension of the bounding rectangle R containing patterns with IDs in s_j (Line 9). Finally, we start with the next recursion based on the tree node c (Line 10), and set the child node of the tree node in the current iteration as the c (Lines 11).

Note that the hierarchical pattern tree can be constructed offline since all the incident patterns are available from historical traffic data. It will not lead to any cost in the online matching process as the hierarchical pattern tree can be obtained before the online match.

4.2. Fast match

Here, we introduce how to efficiently check whether the pattern $\overrightarrow{P_T}$ for trajectory T can match any incident pattern maintained in the hierarchical pattern tree.

As a first step, since incidents rarely occur in the real world, we use the threshold δ to prune trajectories which are close to the trajectories under normal traffic conditions based on the distances between their speed vectors, $\overrightarrow{S_T^{\text{nor}}}$ and $\overrightarrow{S_n^{\text{nor}}}$. Only trajectories with a distance larger or equal than δ are considered for further investigation. After pruning, only n' patterns $\overrightarrow{P_T} \in \mathbb{R}^m$ needs to be checked, where $n' \ll n$.

For each pattern that passes the first step, we need to check whether it can match any incident pattern in the hierarchical pattern tree. To avoid traversing the whole tree, we maintain the lower and upper bounds of the distances between the pattern $\overrightarrow{P_T}$ and the patterns stored in the current sub-tree during the traversal process. Before discussing how the bounds can be used for an early stop, we first present the computation and maintenance of the lower and upper bounds.

Lower bound. According to Eq. (2), different dimensions are independent to each other when computing the pattern distance. Meanwhile, only one dimension is considered in each layer of our hierarchical pattern tree. Hence, the lower bound of the distance between $\overrightarrow{P_T}$ and the patterns in the subtree rooted at current node c can be defined recursively as:

$$L(c) = \begin{cases} L(c.p) + c.R_L - \overrightarrow{P_T}[c.m_{o_i}], & \text{if } c.R_L > \overrightarrow{P_T}[c.m_{o_i}] \\ L(c.p) + \overrightarrow{P_T}[c.m_{o_i}] - c.R_U, & \text{if } c.R_U < \overrightarrow{P_T}[c.m_{o_i}] \\ L(c.p), & \text{otherwise} \end{cases} \tag{4}$$

where $L(c)$ (resp. $L(c.p)$) is the lower bound of the distances between $\overrightarrow{P_T}$ and the patterns in the subtree rooted at c (resp. $c.p$). Note $c.p$ denotes the parent node of the current node c. Because the root node does not have a parent node, we set $L(root.p) = 0$.

When matching $\overrightarrow{P_T}$ via traversing the tree, we use a min-heap to maintain the nodes to be explored using their lower bounds as the key. If the minimum value maintained in the min-heap already exceeds or equals to γ, no incident patterns will be matched. Under such situation, we can safely early stop the matching process.

Upper bound. Similarly, we can also calculate the upper bound of the distance between $\overrightarrow{P_T}$ and the patterns in the subtree rooted at node c with respect to the first l dimensions (assuming c locates in the lth layer in the tree). Hence, the upper bounds can be recursively defined as follows:

$$U(c) = U(c.p) + \max(|c.R_L - \overrightarrow{P_T}[c.m_{o_i}]|, |c.R_U - \overrightarrow{P_T}[c.m_{o_i}]|), \tag{5}$$

where $U(c)$ (resp. $U(c.p)$) is the upper bound of the distances between $\overrightarrow{P_T}$ and the patterns in the subtree rooted at c (resp. $c.p$) with respect to the first l dimensions. Analogically, $U(root.p)$ is set to 0.

We can derive that the distance between the pattern P_T for testing trajectory and each incident pattern P_{a_i} in the subtree rooted at node c, i.e., $d(P_T, P_{a_i})$, satisfies that:

$$L(c) \le d(P_T, P_{a_i}) \le U(c)\ . \tag{6}$$

Note the upper bound will only be used when we traverse into a leaf node, because until then the upper bound has taken all dimensions into calculation. If the upper bound of a leaf node is less than γ, we can safely stop the matching process, because the distances between $\overrightarrow{P_T}$ and all patterns in the leaf node are less than γ.

Based on the above lower-bound- and upper-bound-based pruning techniques, we propose an efficient matching algorithm, detailed in Algorithm 2. We first initialize the min-heap Q (Line 1), compute the lower and upper bounds of the root node (Lines 2–3), and push the root node and its lower bound into the heap (Line 4). Next, we keep processing the nodes in the heap until Q is empty (Lines 5–22). In each iteration, we pop out the node p with minimum lower bound from Q (Line 6), and use the lower bound to check whether we can terminate the match process with `false` returned (Line 7). If p is a non-leaf node, we compute the lower and upper bounds for each child of p and push the child into the heap Q if the lower bound is less than γ (Lines 9–12). If p is a leaf node, we first use the upper bound whether we can stop the process with `true` returned (Line 14). If not, we need to calculate the distance between $\overrightarrow{P_T}$ and each incident pattern in p (Lines 16–19) and check whether the distance is less than γ (Line 20). If all nodes in Q are processed and the process is not terminated, we return `false`, as no pattern can match $\overrightarrow{P_T}$.

Algorithm 2: Fast Match

```
Input : dimension $m$, the pattern for testing trajectory $\overrightarrow{P_T}$, pattern
        tree $tree$, threshold $\gamma$
1  $Q \leftarrow$ min_heap();
2  computes $L(root)$ via Equation (4);
3  computes $U(root)$ via Equation (5);
4  push pair $\langle root, L(root)\rangle$ into $Q$;
5  while $Q$ is not empty do
6      $\langle p, L(p)\rangle \leftarrow Q$.pop();
7      if $L(p) \ge \gamma$ then return false;
8      if $p$ is a non-leaf node then
9          for $c \in p.children$ do
10             computes $L(c)$ via Equation (4);
11             computes $U(c)$ via Equation (5);
12             if $L(c) < \gamma$ then push pair $\langle c, L(c)\rangle$ to $Q$;
13     if $p$ is a leaf node then
14         if $U(p) < \gamma$ then return true;
15         for each incident pattern $P_{a_i} \in p$ do
16             $f_{cur} \leftarrow 0$;
17             for $m_{o_i}$ in $m$ do
18                 $f_{cur} \leftarrow f_{cur} + |\overrightarrow{P_T}[m_{o_i}]$-$P_{a_i}[m_{o_i}]|$ ;
19                 if $f_{cur} \ge \gamma$ then break;
20             if $f_{cur} < \gamma$ then return true;
21 return false;
```

The hierarchical pattern tree based algorithm is an exact solution to speed up the matching process. The lower bound (Eq. (4)) and upper bound (Eq. (5)) are used to reduce the computation cost safely, which do not involve any approximation. Note that the relaxation factor ρ is only used to decide the range of bounding rectangle R, which does not introduce any error.

Comparison with kNN. The kNN based algorithms always find the nearest neighbor first and then check the distance between the pattern of the testing trajectory and the nearest neighbor. However, incidents rarely occur in the real world. Testing trajectories in most cases are normal ones, whose distances to the nearest neighbor in incident patterns are quite large, therefore, they could be pruned earlier to save computation cost. Hence, our algorithm is different from kNN since we can prune earlier and do not require to find the nearest neighbor as in Algorithm 2.

Complexity. The complexity of the fast match is $O(n'k'm')$. In the worst case, it is $O(nkm)$. However, $n' \ll n$, $k' \le k$, and $m' \le m$, in practice. It is efficient as shown in Section 5.

Hyperparameter setting. Since hyperparameters in our method may vary regarding the nature of different trajectory datasets, we use Bayesian optimization to find the optimal hyperparameters in the validation data for each dataset in practice, which can be found in Section 5.4 in detail.

5. Experiments

5.1. Dataset

We conduct experiments on two GPS datasets. The first data set (**HK**) consists of 35.1 GB of trajectory data collected from 440 taxis in Hong Kong in the year of 2010. The generation rate of the GPS positions is around one position every 40 s. The city map of Hong

Kong is collected from OpenStreetMap.[4] We also obtain incident data from the Transportation Department of Hong Kong for the year 2010. From this data, we use 4386 incidents which occurred in road segments which are visited by at least one taxi during the specified time window.

The second dataset (**BJ**) simulates GPS data and incident data using the well-known simulation software Simulation of Urban MObility (SUMO) [32]. SUMO is used in various work on traffic analysis [20, 33,34]. We follow the same setting as in *Traffic State Estimation* [20] to simulate GPS data and incident data in the city map containing all roads within the second ring road in Beijing. We simulated 5000 incident data and 4.41 GB of GPS data.

In addition to the incident instances, we choose randomly non-incident instances — these are randomly chosen locations in the road network where no incident happened at the given time — such that both datasets are balanced with a 1:2.3 split [12,24] between incident and non-incident instances.

Note that we also evaluate on the dataset with dense GPS trajectories to compare the performance with existing methods that target on dense datasets. In Fig. 10, we evaluate all methods on relatively dense datasets (with sparsity = 50%). The reason that we do not use the dense datasets from existing methods is that the dense data they used are summarized data, e.g., average speed over 5 min of all vehicles passing through each segment, and we cannot obtain the trajectory for each individual vehicle. So we extract dense trajectories from GPS trajectory datasets for our trajectory-based incident detection problem, and evaluate the performance in Fig. 10.

We preprocess our datasets in four steps. We first partition the whole road network into road segments not longer than 100 m as [35]. Then, incidents are matched to road segments. We filter out incidents whose distances to their closest road segments exceed 100 m – this only applies to the **HK** dataset, as SUMO ensures that incidents always coincide with road segments for the **BJ** dataset. Then, we conduct map matching by applying the algorithm in [36]. Third, as the time interval between two GPS points is very short (around 40 s), we assume that the acceleration between two GPS points stays constant. We use linear interpolation to estimate the speed of a vehicle for segments in which we do not have a GPS signal — either due to a loss of signal or simply because the vehicles drive too fast or the segment is too short. Fourth, we randomly split the data into 64% for the discovery phase (training), 16% for the hyperparameter tuning (validation), and 20% for the online detection phase (testing).

5.2. Competitors

We compare our method with five advanced competitors that are designed for trajectory data or sensor data.

• SVMN [10]. This method uses an SVM with aggregated spatio-temporal sensor data as features to detect incidents.

• NNA [11]. This method uses a NN with aggregated spatio-temporal sensor data as features to detect incidents.

• Topic Model (TM) [21]. This method applies an *LDA equivalent model* to estimate the state over the speed of trajectories. Then, they measure the deviation between the current state and normal state to detect incidents.

• CNNU [23]. This method uses CNN as model with aggregated spatio-temporal sensor data as features to detect incidents. Convolution, ReLU, Max-Pool and Fully-Connected Layers are applied in their CNN network architecture.

• Traffic State Estimation (TSE) [20]. This method conducts spatial–temporal traffic state analysis from GPS data and uses estimated states to detect incidents. They use SUMO [32] to simulate 24 incidents and the required GPS data.

[4] https://www.openstreetmap.org.

5.3. Performance metrics

Following [10,11,20,23], we evaluate our model based on detection rate (DR), false alarm rate (FAR), mean time-to-detect (MTTD), and F1 score:

$$\text{DR} = \frac{\text{number of detected incidents}}{\text{total number of tested incidents}}, \tag{7}$$

$$\text{FAR} = \frac{\text{number of false alarms}}{\text{total number of tested non-incidents}}, \tag{8}$$

$$\text{MTTD} = \frac{1}{n}\sum_{i=1}^{n}(t_i^{\text{detected}} - t_i^{\text{occurred}}), \tag{9}$$

where t_i^{detected} is the time when the incident is detected, t_i^{occurred} is the time when the incident occurs, and n is the number of correctly detected incidents.

5.4. Hyperparameter tuning

We conduct hyper-parameter tuning by Bayesian optimizer on the validation dataset. The scopes of hyper-parameters are k {10, 50, 100, 150, 200}, m {5, 10, 15, 20}, δ {1.6, 2, 2.4, 2.8, 3.2}, γ {1.5, 2.25, 3, 3.75, 4.5}, τ {2, 4, 6, 8, 10, 12, 14} for effectiveness, and ρ {0, 0.4, 0.8, 1.2, 1.6} for efficiency.

5.5. Evaluation results

For evaluation on efficiency and effectiveness, we choose a time window of 5 min, the size of speed vector $m = 15$, and $k = 100$ incident patterns as default settings. We will show the evaluation over different parameter settings. The evaluation is conducted on a machine with a 2.2 GHz Intel Core i7 CPU and 16 GB 2400 MHz DDR4 Memory.

5.5.1. Incident patterns visualization

We show four different incident patterns in Fig. 7 which are identified during the offline discovery phase in the **HK** dataset. Note that the y-axis represents the difference between the average historical non-incident speed vectors and the incident speed vector. A positive value in the y-axis means that a vehicle affected by an incident is slower than average. We use r_a to indicate the incident location. For example, in the pattern (d) we can see that the vehicle influenced by the incident reduces the speed until reaching the incident location. After passing the incident, the vehicle speeds up and finally the speed is faster than average (negative value on y-axis).

5.5.2. Evaluation on efficiency

Fig. 8(b) and (d) evaluate the efficiency of *Annoy* (AN) [37], our proposed *fast match* (FM in Section 4) and *match* (M) – *match* is the base method as described in Section 3.3.3 without any efficiency improvements. *Annoy* (AN) has been recognized as one of the best Nearest Neighbor (NN) libraries [37]. The ratio of the number of incident speed patterns and normal speed patterns is 1:100 000 as estimated from the real data set **HK**. The number of incident patterns k is 200, and the size of the sliding window m is 20. Fig. 8 shows the average running time of one speed pattern in the unit of milliseconds. The execution time of FM is much lower than AN. It verifies that FM is more efficient than the kNN based algorithm AN. Moreover, FM is 48.1 times faster than M in the **BJ** dataset, and 28.5 times faster than M in the **HK** dataset. Note that FM is an exact algorithm, which means that the effectiveness is not affected when changing the relaxation factor ρ.

We illustrate in Fig. 8(a) and (c) how the fast matching algorithm behaves with changing the relaxation factor ρ. As shown in Fig. 8, the execution time decreases when increasing ρ. As a result, the execution time can be reduced. For example, the execution time drops 44.11% when increasing ρ from 0 to 1.2 in the **HK** dataset.

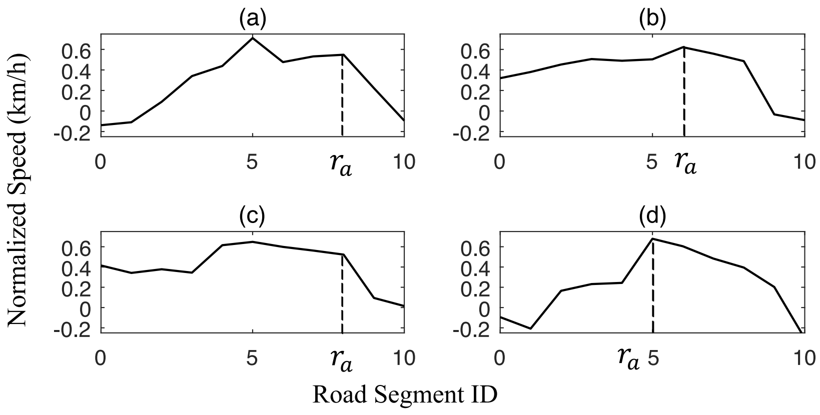

Fig. 7. Incident patterns discovered in HK.

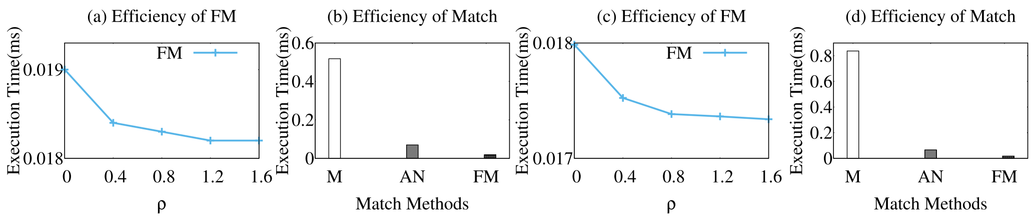

Fig. 8. Efficiency evaluation over ρ in (a–b) HK and (c–d) BJ. Note that ρ equals 1.2 in (b) and (d) for FM.

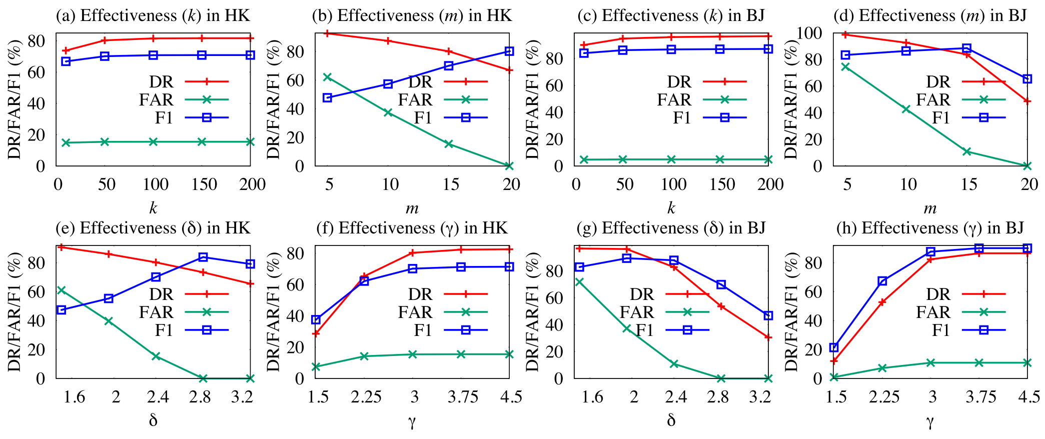

Fig. 9. Effectiveness evaluation over the number of incident patterns k, the size of sliding window m, threshold δ, threshold γ.

Table 3
Comparison with competitors in HK.

Methods	F1	DR	FAR	MTTD (min)
TM [21]	72.9	71.1	23.9	2.5
TSE [20]	62.0	51.7	15.0	2.6
CNNU [23]	70.8	79.7	42.7	5.0
SVMN [10]	70.9	**85.2**	55.2	5.0
NNA [11]	70.6	83.2	52.3	5.0
FDM	**79.4**	75.8	**14.9**	**2.27**

Table 4
Comparison with competitors in BJ.

Methods	F1	DR	FAR	MTTD (min)
TM [21]	74.4	71.7	20.9	2.4
TSE [20]	85.6	83.1	10.9	2.5
CNNU [23]	73.4	73.4	26.6	5.0
SVMN [10]	67.5	65.9	29.2	5.0
NNA [11]	63.0	60.7	31.8	5.0
FDM	**89.9**	**86.4**	**8.69**	**1.1**

5.5.3. *Evaluations on effectiveness*

First, we list the evaluation results in Tables 3 and 4 by comparing our method (FDM) with the competitors in both the **HK** and **BJ** datasets. As one can see, our method has the highest F1 score. In addition, our method has the lowest FAR and MTTD. Finally, our method has the highest DR for the **BJ** dataset and is still on a competitive level on the **HK** dataset. We want to point out that a low FAR is especially important as, in real-world applications, most investigated trajectories are expected to be non-incidents, rather than incidents.

Second, we test how the performance of our method changes when changing the number of incident patterns k and the size of the speed vector m. In Fig. 9(a and c), we can see that the F1 score initially increases when using more incident patterns. This confirms our hypothesis, that there is no single incident pattern which is valid for all

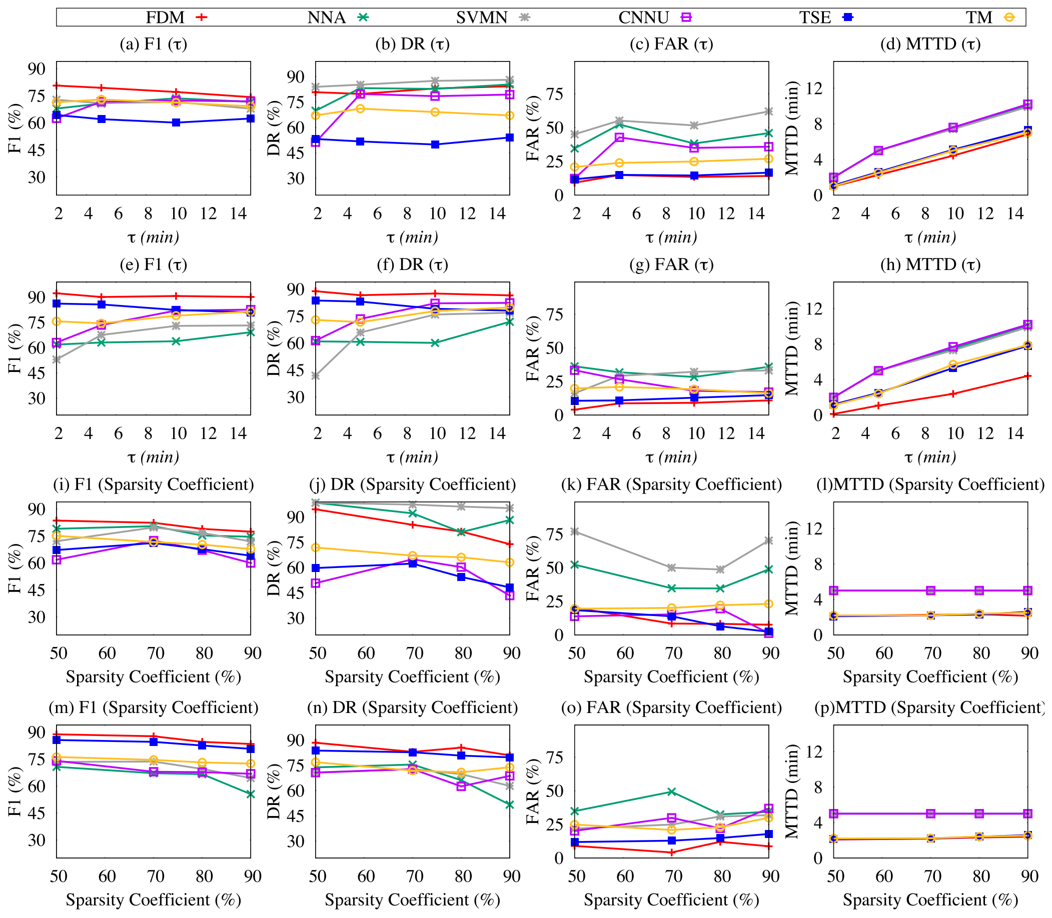

Fig. 10. Evaluation as τ increases in (a–d) HK and (e–h) BJ and as sparsity coefficient increases in (i–l) HK and (m–p) BJ.

incidents. When increasing the number of patterns further, however, the F1 score remains stable. As one can see from Fig. 9(a and c), the F1 score is maximized for $k = 100$, which is the reason why we chose this as the default parameter in our experiments.

In Fig. 9, we compare DR, FAR, and F1 scores for varying sizes of the speed vector m (b and d). The F1 scores are maximized when $m = 20$ and $m = 15$ for **HK** and **BJ** datasets respectively. We also evaluate δ in Fig. 9(e and g) and γ in Fig. 9(f and h). The F1 scores are maximized when $\delta = 2.8$ and $\delta = 2.4$ for **HK** and **BJ** accordingly. $\gamma = 3.75$ gives the best F1 scores for both **HK** and **BJ**.

Third, we evaluate how the different methods perform when varying the time window τ (cf. Definition 4) used in these methods. Fig. 10(a–h) shows the F1 score, MTTD, DR, and FAR for the following methods: NNA, SVMN, CNNU, TSE, TM and our own method: FDM.

As one can see, CNNU benefits most from a bigger time window with respect to the F1 score. In contrast, TSE first drops as the time window increases and finally increases slightly. The other methods are quite stable beyond a time window of 5 min, even showing some slight decrease in performance when the windows increase in both datasets, **HK** and **BJ**.

5.5.4. *Evaluation over different sparsity levels*

To evaluate the effectiveness over sparsity levels, we sample different subsets from both datasets to obtain datasets with different sparsity coefficient. Then, we evaluate the performance of our method and competitors over datasets having different sparsity coefficients. We give a formal definition as follows:

The *sparsity coefficient* of a road network dataset is the percentage of cases where no vehicle passed through a road segment s_i during a time window $[t_{j-1}, t_j)$.

$$\text{Sparsity Coefficient} = \frac{\sum_{i,j} \mathbb{1}_{\{V_{s_i,[t_{j-1},t_j)}=\texttt{false}\}}}{S \cdot T} \cdot 100\% \,, \tag{10}$$

where s_i with $i \in [0, S]$ represents the ith road segment and $[t_{j-1}, t_j)$ with $j \in [0, T]$ stands for the jth time window between t_{j-1} and t_j. $V_{s_i,[t_{j-1},t_j)}$ is a Boolean variable that indicates whether at least one vehicle passes through road segment s_i during time window $[t_{j-1}, t_j)$. In addition, $\mathbb{1}_{\{V_{s_i,t_j}=\texttt{false}\}}$ is an indicator function that is 1 if V_{s_i,t_j} = `false` and 0 otherwise, S is the total number of road segments s_i and T is the total number of time windows $[t_{j-1}, t_j)$. The impact of the Sparsity Coefficient is evaluated in Fig. 10(i–p). Although SVMN and NNA achieve better DR scores, their FAR scores are much higher than our FDM, and their F1 scores are quite low in the **HK** dataset. As we can see, when the sparsity of the dataset increases (sparsity coefficient from 50% to 80%), our FDM method still outperforms the competitors with respect to F1 score and achieves the best MTTD, whereas the DR and FAR are still within a competitive range in both **HK** and **BJ** datasets. It means that the trajectory-based method is tolerant to data sparsity in spatial and temporal dimensions.

6. Limitations

We conducted a manual examination of the weaknesses in our methodology. We discovered that False Positives, although not true incidents, exhibit a deceleration followed by an acceleration pattern, leading to their erroneous identification. Despite not being true incidents, False Positives display a pattern of behavior similar to real incidents. This pattern mimics the typical behavior seen in actual incidents, such as a vehicle slowing down due to an obstacle or hazard on the road and then accelerating once the obstacle is passed. However, in the case of False Positives, this pattern occurs without any actual incident present, leading to their mistaken identification as true incidents. This discovery highlights a crucial challenge in incident detection systems, where the presence of certain behavioral patterns can lead to false alarms or erroneous identifications.

Unlike False Positives, where there is a pattern resembling true incidents, False Negatives involve actual incidents that the system misses. False Negatives involve minor incidents that quickly recover, resulting in a short duration of queuing or slowly moving. In this case, the short duration of the queuing makes it difficult for the system to detect the incident, especially since the typical pattern of deceleration followed by acceleration, which is often associated with incidents, may not be clearly observed. As a result, these incidents are incorrectly classified as non-incidents or missed entirely. This highlights another challenge in incident detection systems, where certain types of incidents may not exhibit the expected patterns or may occur in a manner that makes them challenging to detect, leading to False Negatives.

7. Conclusions

In this paper, we study the problem of incident detection on urban roads on a city-wide scale. We propose a new model (FDM) that is inspired by an insight from the transportation field. Further, we propose a hierarchical pattern tree based algorithm to speed up the detection process. We evaluate our method according to different parameter settings by extensive experiments. Our results show that our method is more effective while having the lowest MTTD among all competitors. Thus, our method is a feasible solution to incident detection with sparse trajectories on a city-wide scale. And it is efficient on large trajectory data, which is important to give immediate response in the real time.

CRediT authorship contribution statement

Xiaolin Han: Funding acquisition, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Tobias Grubenmann:** Writing – review & editing. **Chenhao Ma:** Writing – original draft, Writing – review & editing. **Xiaodong Li:** Investigation, Validation. **Wenya Sun:** Formal analysis. **Sze Chun Wong:** Data curation. **Xuequn Shang:** Writing – review & editing. **Reynold Cheng:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

Xiaolin han was supported by NSFC under Grant 62302397, Advanced Computing and Intelligent Engineering (National) Laboratory Fund Project (No. 2023-LYJJ-01-021) and Fundamental Research Funds for the Central Universities, China under Grant No. D5000230191. Chenhao Ma was supported by NSFC under Grant 62302421, Basic and Applied Basic Research Fund in Guangdong Province under Grant 2023A1515011280. Tobias Grubenmann was supported by the Federal Ministry of Education and Research (BMBF), Germany, under Simple-ML (01IS1 8054), the European Commission under PLATOON (872592) and Cleopatra (812997), and UK Research and Innovation, United Kingdom, under the STFC follow-on funding ST/V002945/1. Reynold Cheng, Xiaodong Li and Wenya Sun were supported by the University of Hong Kong (Projects 109000579), the HKU Outstanding Research Student Supervisor Award (2022-23). Sze Chun Wong is supported by the Guangdong–Hong Kong-Macau Joint Laboratory Program 2020 (Project No.: 2020B1212030009).

References

[1] R.P. Magalhaes, F. Lettich, J.A. Macedo, F.M. Nardini, R. Perego, C. Renso, R. Trani, Speed prediction in large and dynamic traffic sensor networks, Inf. Syst. 98 (2021) 101444.
[2] S. Taghizadeh, A. Elekes, M. Schäler, K. Böhm, How meaningful are similarities in deep trajectory representations? Inf. Syst. 98 (2021) 101452.
[3] M.T. Cazzolato, A.J. Traina, K. Böhm, Establishing trajectories of moving objects without identities: The intricacies of cell tracking and a solution, Inf. Syst. 105 (2022) 101955.
[4] A. Efentakis, N. Grivas, D. Pfoser, Y. Vassiliou, Crowdsourcing turning-restrictions from map-matched trajectories, Inf. Syst. 64 (2017) 221–236.
[5] T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, Dynamic route planning with real-time traffic predictions, Inf. Syst. 64 (2017) 258–265.
[6] A. Gal, A. Mandelbaum, F. Schnitzler, A. Senderovich, M. Weidlich, Traveling time prediction in scheduled transportation with journey segments, Inf. Syst. 64 (2017) 266–280.
[7] Z. Jiang, M. Evans, D. Oliver, S. Shekhar, Identifying K Primary Corridors from urban bicycle GPS trajectories on a road network, Inf. Syst. 57 (2016) 142–159.
[8] N. Andrienko, G. Andrienko, S. Rinzivillo, Leveraging spatial abstraction in traffic analysis and forecasting with visual analytics, Inf. Syst. 57 (2016) 172–194.
[9] S. Ghosh-Dastidar, H. Adeli, Wavelet-clustering-neural network model for freeway incident detection, Comput.-Aided Civ. Infrastruct. Eng. 18 (2003) 325–338.
[10] T. Šingliar, et al., Learning to detect incidents from noisily labeled data, Mach. Learn. 79 (3) (2010) 335–354.
[11] D. Srinivasan, X. Jin, R.L. Cheu, Adaptive neural network models for automatic incident detection on freeways, Neurocomputing 64 (2005) 473–496.
[12] F. Yuan, R.L. Cheu, Incident detection using support vector machines, Transp. Res. C 11 (3–4) (2003) 309–328.
[13] X. Han, T. Grubenmann, R. Cheng, S.C. Wong, X. Li, W. Sun, Traffic incident detection: a trajectory-based approach, in: 2020 IEEE 36th international conference on data engineering (ICDE), IEEE, 2020, pp. 1866–1869.
[14] W. Wong, S.C. Wong, Evaluation of the impact of traffic incidents using GPS data, Proc. Inst. Civ. Eng.-Transp. 169 (2016) 148–162.
[15] H.J. Payne, S.C. Tignor, Freeway incident-detection algorithms based on decision trees with states, Transp. Res. Rec. (682) (1978).
[16] Y.J. Stephanedes, J. Hourdakis, Transferability of freeway incident detection algorithms, Transp. Res. Rec. 1554 (1) (1996) 184–195.
[17] S.A. Ahmed, A.R. Cook, Discrete dynamic models for freeway incident detection systems, Transp. Plan. Technol. 7 (4) (1982) 231–242.
[18] S. Tang, H. Gao, Traffic-incident detection-algorithm based on nonparametric regression, IEEE Trans. Intell. Transp. Syst. 6 (1) (2005) 38–42.
[19] R. Wang, D.B. Work, R. Sowers, Multiple model particle filter for traffic estimation and incident detection, IEEE Trans. Intell. Transp. Syst. 17 (12) (2016) 3461–3470.
[20] E. D'Andrea, F. Marcelloni, Detection of traffic congestion and incidents from GPS trace analysis, Expert Syst. Appl. 73 (2017) 43–56.
[21] A. Kinoshita, A. Takasu, J. Adachi, Real-time traffic incident detection using a probabilistic topic model, Inf. Syst. 54 (1) (2015) 169–188.
[22] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent dirichlet allocation, J. Mach. Learn. Res. 3 (Jan) (2003) 993–1022.
[23] L. Zhu, F. Guo, R. Krishnan, J.W. Polak, A deep learning approach for traffic incident detection in urban networks, in: IEEE Intelligent Transportation Systems Conference, 2018.
[24] J. Wang, X. Li, et al., A hybrid approach for automatic incident detection, IEEE Trans. Intell. Transp. Syst. 14 (3) (2013) 1176–1185.
[25] J. Lan, C. Long, R.C.-W. Wong, Y. Chen, Y. Fu, D. Guo, S. Liu, Y. Ge, Y. Zhou, J. Li, A new framework for traffic anomaly detection, in: SDM, SIAM, 2014, pp. 875–883.
[26] M. Yue, L. Fan, C. Shahabi, Traffic accident detection with spatiotemporal impact measurement, in: PAKDD, 2018.
[27] V.W. Zheng, B. Cao, Y. Zheng, X. Xie, Q. Yang, Collaborative filtering meets mobile recommendation: A user-centered approach, in: AAAI, 2010.
[28] V.W. Zheng, Y. Zheng, X. Xie, Q. Yang, Collaborative location and activity recommendations with GPS history data, in: WWW, 2010.
[29] V.W. Zheng, Y. Zheng, X. Xie, Q. Yang, Towards mobile intelligence: Learning from GPS history data for collaborative recommendation, Artificial Intelligence 184 (2012).
[30] Y. Zheng, T. Liu, Y. Wang, et al., Diagnosing New York city's noises with ubiquitous data, in: UbiComp, 2014.
[31] Y. Tao, D. Papadias, Historical spatio-temporal aggregation, ACM Trans. Inf. Syst. 23 (1) (2005) 61–102.

[32] P.A. Lopez, M. Behrisch, B.-W. et al., Microscopic traffic simulation using sumo, in: ITSC, IEEE, 2018, pp. 2575–2582.
[33] X.-F. Xie, S.F. Smith, L. Lu, G.J. Barlow, Schedule-driven intersection control, Transp. Res. C 24 (2012) 168–189.
[34] C. Sommer, Z. Yao, R. German, F. Dressler, Simulating the influence of IVC on road traffic using bidirectionally coupled simulators, in: IEEE INFOCOM Workshops 2008, 2008, pp. 1–6.
[35] D. Dai, D. Jaworski, Influence of built environment on pedestrian crashes: A network-based GIS analysis, Appl. Geogr. 73 (2016) 53–61.
[36] P. Newson, J. Krumm, Hidden Markov map matching through noise and sparseness, in: GIS, 2009, pp. 336–343.
[37] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, X. Lin, Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement, IEEE Trans. Knowl. Data Eng. (2019).