Embedding-Based Similarity Computation for Massive Vehicle Trajectory Data

Yuanyi Chen, Peng Yu, Wenwang Chen, Zengwei Zheng, and Minyi Guo, *Fellow, IEEE*

***Abstract*—Trajectory similarity computation is a fundamental problem for many intelligent applications such as trip trajectory mining to find the most popular routes and road anomaly detection. Existing similarity computation methods for vehicle trajectory, such as dynamic time warping (DTW) and Fréchet distance, are dynamic programming problems with a quadratic time complexity in all cases and need to handle local time shifts when computing the distance between trajectories, thus limiting the scalability of these methods. In addition, GPS-based vehicle trajectories usually contain errors, such as noise and outliers, due to poor satellite visibility in urban regions and nonuniform sampling rates. To this end, we propose an embedding-based method for trajectory similarity computation with linear time complexity, which encodes a trajectory as a vector via deep representation learning and learns the similarity between trajectories with an attention-based learning to rank model. We use an interpolation-based approach to reduce noise and outliers by considering vehicle trajectory is physically constrained to the road network. Experiments on a massive vehicle trajectory data set show that the proposed approach outperforms state-of-the-art baselines consistently and significantly.**

***Index Terms*—Deep neural network, learning to rank (L2R), list-wise ranking loss, trajectory encoding, trajectory similarity computation.**

I. Introduction

THE PROLIFERATION and ubiquity of GPS-enabled devices across many transport has generated substantial interest in the analysis and mining of vehicle trajectory data. Vehicle trajectory data offer us crucial information for applications in different research fields, such as location selection (e.g., selecting new public toilet location [1]). Among all trajectory analysis techniques, quantifying the similarity between a pair of trajectories is a fundamental research problem due to its importance for many important mining operations, such as anomalous trajectory detection [2] and city-wide package delivery path selection [3], which aims to extract qualified trajectories that have passed a user-specified path (i.e., a sequence of discrete points or locations). Fig. 1 gives an example, where a user retrieves the trajectories that have passed points ($p1 \longmapsto p2 \longmapsto p3$) and the qualified trajectories are returned as the colorful lines. Many intelligent transportation applications rely heavily on trajectory similarity computation.

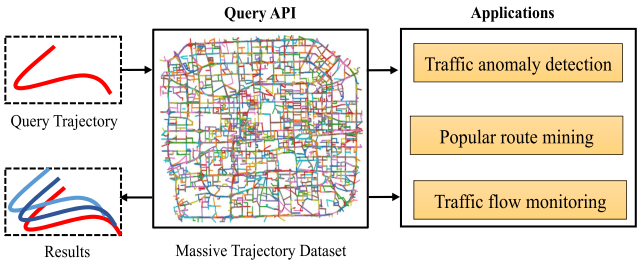


Fig. 1. Motivating examples.

1) *Prediction the Flight Time of Unmanned Aerial Vehicles (UAVs):* Nowadays, UAVs are particularly useful for collecting information in areas that are too dangerous or difficult for humans to reach, and it is important to know beforehand the expected flight time of a UAV in order to ensure a successful flight. One can identify trajectories similar to the current trajectory from historical data sets to predict the flight time when a UAV is traveling on a known route. As the employed similarity measure directly impacts the effectiveness and efficiency of these kinds of tasks, one needs to choose the most suitable similarity measure carefully.
2) *Detection of Abnormal Animal Migration:* Animal movement is studied in fields as diverse as population ecology, foraging theory, and resource exploitation and conservation biology. The migration trajectory of animals plays an important role in detecting new/rare animal behaviors and environmental changes. Researchers can judge whether the animal migration route is abnormal by trajectory similarity computation. Because of the limited capacity of network facilities in wildlife living environments, trajectory data cannot be uploaded to the powerful data centers, thus we still need efficient approaches to tackle the similarity-based search of big trajectory data.

Researchers have proposed various similarity methods to capture the intrinsic structural similarities between vehicle trajectories, including dynamic time warping (DTW) [4], Hausdorff distance [5], Fréchet distance [6], and edit distance with real penalty (ERP) [7]. For handling the variety of

Manuscript received February 22, 2021; revised June 26, 2021; accepted August 19, 2021. Date of publication August 24, 2021; date of current version March 7, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 61802343 and Grant 62072402; in part by the Zhejiang Provincial Natural Science Foundation of China under Grant LGN20F020003; and in part by the Hangzhou Science and Technology Bureau under Grant 20191203B37 and the Intelligent Plant Factory of Zhejiang Province Engineering Lab. *(Corresponding author: Zengwei Zheng.)*

Yuanyi Chen, Peng Yu, Wenwang Chen, and Zengwei Zheng are with the Department of Computer Science and Computing, Zhejiang University City College, Hangzhou 310015, China (e-mail: zhengzw@zucc.edu.cn).

Minyi Guo is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China.

Digital Object Identifier 10.1109/JIOT.2021.3107327

distortions when comparing two trajectory sequences, these existing methods usually try to form a pairwise matching for the sample points of two trajectories and identify the best alignment using dynamic programming. More specifically, these existing methods rely on local point matching by finding an optimal alignment using dynamic programming, which leads to quadratic computational complexity $O(n^2)$, where n is the mean length of the trajectories. Therefore, existing methods are computation costly and cannot be scalable to large data sets. For example, it takes us more than 10 h to compute the pairwise Fréchet distance for merely 10 000 GPS-based vehicle trajectories on a high-end server. To cope with the aforementioned problem, many research efforts [8]–[13] have been proposed on accelerating trajectory similarity computation. Some efforts [8]–[10] aim to reduce the involved number of computations at a global level. Instead of reducing the computation complexity for an *ad hoc* pair of trajectories, they focus on designing indexing and pruning strategies for a given trajectory database. Another efforts [11]–[13] are to reduce the involved number of computations at a global level, they focus on designing indexing and pruning strategies for a given trajectory database instead of reducing the computation complexity for an *ad hoc* pair of trajectories. Unfortunately, these techniques are designed for one specific similarity method and not applicable to any other methods.

The rapid development of Internet of Things (IoT) and 5G communications has given rise to an extremely large volume of vehicle trajectory data, thus most IoT devices especially those with limited resources are not capable of handling such large amounts of data in real time. The massive data transferred from the vehicle data source to the remote cloud centers will bring many challenges, such as increasing the capacity pressure of the backhaul link and augmenting the computing load of the cloud center [14], [15]. These challenges motivate a move toward an efficient trajectory similarity computing approach that facilitates the processing of data closer to IoT devices (e.g., driver-less vehicles and UAV), which increase the processing efficiency and reduce cloud server storage costs by transferring only relevant information [16], [17].

To cope with the aforementioned challenges, we propose an embedding-based vehicle trajectory similarity computation method with deep representation learning, which can drastically accelerate trajectory similarity computation with linear-time complexity. The proposed method first transforms each trajectory data into a low-dimensional vector with representation learning, then samples a set of training trajectories as seeds and learns their pairwise similarities through list-wise ranking learning. Specifically, we utilize memory-augmented recurrent unit [18] to capture the correlations between training trajectories to produce better trajectory embeddings, then learn the similarity from the embedding space with a list-wise ranking learning module. The novelty of the proposed method lies in it only takes a linear time $O(n)$ to compute the similarity between a pair of trajectories, while existing methods take quadratic computational complexity. The main contributions of this article are threefold.

1) We propose a deep representation learning to encode vehicle trajectories as a low-dimensional vector by utilizing memory-augmented recurrent unit to capture the correlations between trajectories.
2) We propose the list-wise ranking learning with weighted sampling, which can effectively learn the similarity between trajectories and sample the discriminative training pairs to accelerate the process of learning.
3) We conduct extensive experiments on a massive data set with more than one million vehicle trajectories. The results demonstrate that the proposed model consistently outperforms state-of-the-art baselines in both accuracy and efficiency.

The remainder of this article is organized as follows. In Section II, we show related work about similarity computation of vehicle trajectory. In Sections III and IV, we present the problem definition and detail the proposed embedding-based similarity computation method. In Section V, we introduce the data set and discuss the experimental results comparing to state-of-the-art baselines. Finally, we present our conclusion and future work in Section VI.

II. Related Work

Quantifying the similarity between a pair of vehicle trajectories is a fundamental concern in the smart mobility solutions. Previous efforts to tackle vehicle trajectory similarity can be mainly divided into two groups: 1) similarity measure-based methods and 2) representation learning-based methods. This section reviews these mainstream methods.

Similarity Measure-Based Methods: DTW [4] is the first attempt to cope with the local time shift issue for computing vehicle trajectory similarity. ERP [7], edit distance on real sequence (EDR) [19], dissimilarity metric (DISSIM) [20], and the model-driven assignment (MA) [21] are developed to further improve the ability of capturing the spatial semantics in vehicle trajectories. Wang *et al.* [22] studied the effectiveness of these similarity methods according to their robustness to noise, varying sampling rates, and shifting. These methods mainly focus on identifying the optimal alignment based on sample point matching, which is inherently sensitive to variation in the sampling rates. To solve this issue, Su *et al.* [9] proposed an anchor-based calibration system (ACS) that learns transition patterns of anchor points from historical trajectories, while Edit Distance with Projections (EDwPs) [23] computes the cheapest set of replacement and insertion operations using linear interpolation to make them identical. To speed up the calculation of trajectory similarity, Hausdorff [5] and Fréchet distances [6] are widely used in computational geometry to measure distances between geometric objects, especially for curves and trajectories. Toohey [24] performed extensive experiments on four similarity measures, namely, longest common subsequence (LCSS) [25], Fréchet distance, DTW, and EDR, which concluded there is a strong correlation between the values of the Fréchet distance and DTW. The work [22] presented an experimental study to compare popular trajectory similarity measures (e.g., DTW, ERP, EDR, and DISSIM), which aimed to evaluate the effectiveness of the

TABLE I
SIMILARITY MEASURE-BASED METHODS FOR VEHICLE TRAJECTORIES

Method	DTW	ERP	EDR	DISSIM	EDwP	LCSS	ACS	Hausdorff	Fréchet distance
Time complexity	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O((m+n)^2))$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2 logn^2)$
Parameter-free	√	√		√	√		√	√	√
Robustness			√		√	√	√		
Different lengths	√	√	√	√	√	√	√	√	√
Local time shifting	√	√	√	√	√	√	√		

TABLE II
FREQUENTLY USED NOTATION

Symbol	Description
T	A trajectory in the dataset consist of a series of (longitude, latitude) tuples
X_t^c	The original coordinate tuple input of our deep representation learning model at $t-$step
X_t^g	The grid coordinate tuple input of our deep representation learning model at $t-$step
$f(\cdot,\cdot)$	The pre-defined similarity function needed to be learned, or say the ground-truth of two trajectories
$g(\cdot,\cdot)$	The learned similarity function through our model
$\mathbf{h}_t, \mathbf{h}_{t-1}$	The hidden states of GRU unit at time step t and $t-1$
$\tilde{\mathbf{h}}_t$,$\mathbf{c}_t$	The medium states of GRU unit at time step t
S	An $N \times N$ symmetric similarity matrix by computing the similarity of N trajectories over similarity function $f(\cdot,\cdot)$
M	The normalized similarity matrix from S
$\mathbf{E}_i$	Embedding vector learned by deep representation learning
Γ_a^s, Γ_a^d	The two weighted sampled lists of similar trajectories and dissimilar trajectories of anchor trajectory T_a
L_a^s, L_a^d	List-wise loss of similarity list Γ_a^s and dissimilarity list Γ_a^d
L	The total loss calculated based on L_a^s and L_a^d

measures and demonstrate their advantages and drawbacks in different circumstances.

We summarize the properties of these similarity methods for vehicle trajectories in Table I from the following five aspects.

1) Efficiency, which is determined by the computational complexity to calculate the similarity between trajectories.
2) Local time shifting, which reflects whether these methods can handle local time shifts when computing the distance between trajectories.
3) Different lengths, which reflects whether these methods are able to handle trajectories that do not have the same length of samples.
4) Robustness, which means whether these methods can handle noise and outliers caused by low satellite visibility and anomalies in the sensor collecting phase.
5) Parameter-free, which means whether these methods need to introduce parameters to compare trajectories. Unfortunately, most of the aforementioned methods are based on the dynamic programming technique to identify the optimal alignment, which leads to $O(n^2)$ computation complexity.

Given the complexity, it will be computationally expensive if we want to apply these similarity measures to a large trajectory database.

Representation Learning-Based Methods: To facilitate advanced mining tasks such as similarity computation and classification, several studies [26]–[30] utilized representation learning technology to transform raw GPS-based vehicle trajectories as a vector by preserving most information from the original data. Zheng *et al.* [26] first studied the hidden routes from partial observations, and Banerjee *et al.* [28] further explored it using Bayesian posterior inference to estimate the top-k most likely routes. Chen *et al.* [27] proposed an efficient trajectory compressing approach by spatial-directional matching and heading change compression, which innovatively utilizes a new dimension of information (i.e., heading direction and heading change) collected by vehicle-mounted GPS devices. T2vec [29] proposed a kind of deep representation learning to perform the trajectory encoding. Trajectory representation learning via road networks [30] is another deep-learning-based approach for trajectory representation by incorporating the underlying road network. Our work is related to the recent development of deep representation learning, which aims at learning a representation vector for an individual trajectory based on neural networks. To capture the sequential order information emerging in the sequence trajectory data, we utilize the spatial attention mechanism to capture the correlations among all the trajectories instead of modeling one sequence independently, then adopt a weighted-sampling strategy to focus on discriminative trajectory pairs instead of randomly sampling which leads to low efficiency. In addition, an interpolation-based approach is utilized to reduce noise and outliers by considering vehicle trajectory is physically constrained to the road network.

III. PROBLEM STATEMENT

For ease of the following presentation, we first define the key data structures and notations used in the problem description and proposed solution approach. Table II lists the relevant notations used in this article.

A. Definitions

Definition 1 (Trajectory): A trajectory is a sequence of time-stamped point records describing the motion history of any kind of moving object. Trajectory T is represented by a sequence of trajectory sample points.

Definition 2 (Trajectory Similarity): Trajectory similarity is a concept that measures the distance between two trajectories. $d(T_1, T_2)$ denotes the distance between two trajectories T_1 and T_2. The larger the value is, the less similar the two trajectories are.

Trajectory similarity computation asks for whether the measure considers the spatial attribute only, i.e., treating two compared trajectories as sequences, or considers both spatial and temporal information in the trajectory. Spatial information means the sequence order of trajectory. Temporal information is time-related information. For distance measures that only measure the spatial distance between trajectories, we name them sequence-only distance measures. For distance measures that measure both the spatial and temporal distance between trajectories, we name them spatiotemporal distance measures.

Since we specify the above two kinds of similarity measures, there are two ways we formulate the similarity problem—whether to involve the timestamp. In our study, we do not focus on the timestamp and concentrate on the location of each sample point.

B. Problem Description

A trajectory means a sequence of points recording the trace of a moving object. Usually, a record consists of the timestamp and location. In this article, we ignore timestamp and only focus on the location. The data set we adopt is the taxi driving record, so a trajectory $T = [X_1, \ldots, X_t, \ldots]$ in our method means a sequence of (longitude and latitude) tuples, where X_t is the tth record of (longitude and latitude) in trajectory T.

Trajectories are usually treated as multidimensional (2-D or 3-D in most cases) time-series data, such as the animals migration or vehicle routes; hence existing distance measures for 1-D time series (e.g., stock market data) can be applied directly or with minor extension. Typical examples include the distance measures based on DTW, edit distance, and LCSS. Such distance measures were originally designed for traditional time series but now have been extensively adopted for trajectory similarity computation. However, with the rapid development of GPS-embedded devices, there are more and more methods to collect trajectory data from them. Trajectory data is not a simple multidimensional extension of time series, but also has some unique characteristics, which should be considered when designing effective similarity computation.

1) Asynchronous Observations: Unlike time-series databases, in trajectory databases, there is usually no central and synchronized mechanism to control the timing of collecting location data. Moving objects, such as GPS-embedded vehicles, may have different strategies when they need to report their locations to a central repository, such as time-based, distance-based, and prediction-based strategies. Even worse, they might suspend the communication with a central server for a while and resume later. The overall result is that the lengths and timestamps of different trajectories are not the same.

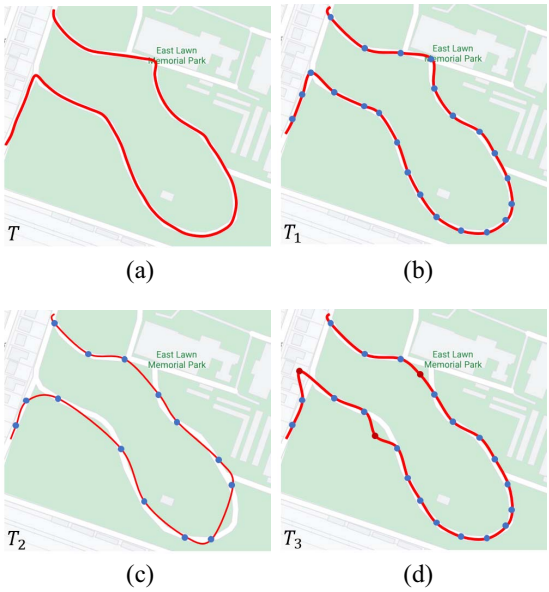


Fig. 2. Different trajectory sampling strategies. (a) Original route. (b) Good sampling. (c) Nonuniform sampling. (d) Noise.

2) More Data Quality Issues: In the traditional time-series databases, there are usually high-quality data due to stable and quality-guaranteed sources to collect the data. Financial data may be one of the most precise time-series data and almost error-free. In environmental monitoring applications, data readings from sensors also have little noise. In contrast, trajectory data are faced with more quality issues, since they are generated by individuals in a complex environment. Actually, such situation is quite common. GPS devices have measurement precision limits. For example, due to the devices' inherent limitation, it may fail to grasp the exact location information. What is more, when bypassing the rough area, it is hard for the devices to locate the accurate position. They all lead to the generation of the noise. The devices may lower the frequency of recording out of energy consideration, which leads to the nonuniform and low sampling rates.

As seen in Fig. 2, an original route T is shown in Fig. 2(a), and three trajectories T_1, T_2, and T_3 sampled from it are displayed in Fig. 2(b)–(d), respectively. T_1 is a good sampling because it can accurately describe the original route T. Compared to T_1, T_2 holds the low and nonuniform sampling rate. So it can barely represent the original route and distinguishes T_1 computationally. T_3 holds the same sampling rate with T_1 but has three unmatched points pairs. The traditional pair-match similarity computation methods roughly compute the distance pair by pair and identify them as different routes mistakenly. To this end, it is hard to tell which one represents the true route because of the "noise" points.

To cope with the statements above, we have come up with a method to cover it. Without loss of generality, we consider 2-D trajectories as we present before. For a trajectory similarity function $f(\cdot, \cdot)$ measuring trajectories T_i and T_j, we aim to learn a trajectory similarity function $g(\cdot, \cdot)$ through our deep

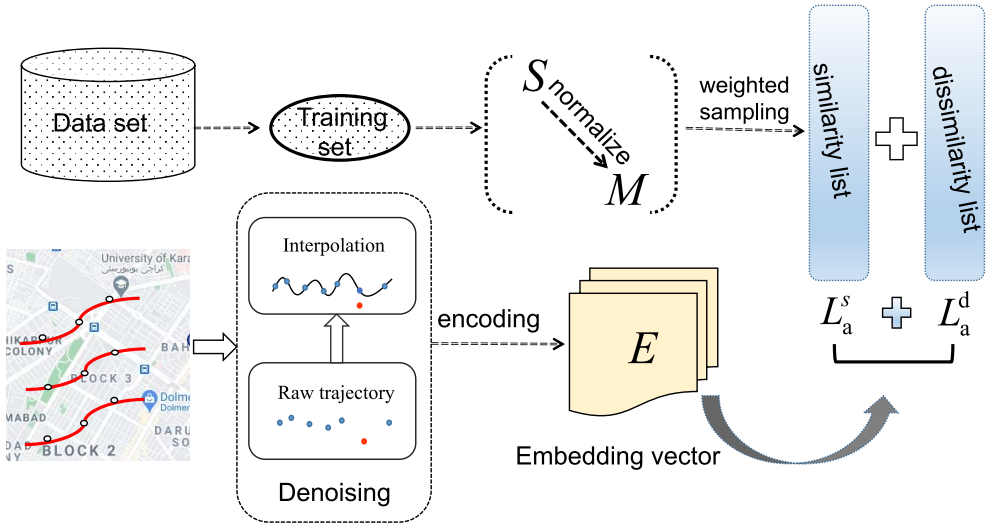


Fig. 3. Architecture of our method.

representation learning to approach $f(\cdot,\cdot)$, and accelerate the computing procedure without losing the accuracy.

IV. Proposed Method

As shown in Fig. 3, our proposed approach for trajectory similarity computation with three phases: 1) data preprocessing to eliminate noise for every pending trajectory; 2) deep representation learning encode the trajectories and combine a kind of memory-augmented RNN unit which NEUTRAJ [18] proposed to memorize the correlation between trajectories; and 3) learning to rank (L2R) to learn the similarity between trajectories, which randomly samples N trajectories as the training set from the data set. Then, we compute the $N \times N$ symmetric matrix S with the predefined similarity function $f(\cdot,\cdot)$, and normalize it to a standard similarity matrix M. The goal is to learn the similarity function $g(\cdot,\cdot)$ based on the matrix S through the L2R model.

1) *Data Preprocessing:* Due to the ubiquity of noise in the trajectory data, we apply Lagrange Interpolation to preprocess the trajectories. For simplicity, we determine the noise by comparing adjacent points and then exploit interpolation to substitute them.
2) *Deep Representation Learning:* We simply divide the space into several parts for embedding consideration. For raw input trajectory T, we adopt the recurrent neural network to encode the trajectory tuple sequence and take the last hidden state of the GRU units as the embedding vector of T. In order to learn the similarity of the training set exactly, we import memory-augmented RNN [18] to memorize the similarity between processed trajectories.
3) *List-Wise Ranking Loss With Weighted Sampling:* Since we encode the trajectory as an embedding vector and memorize the similarity between any two trajectories, we built an L2R to learn the similarity of the training set. For any anchor trajectory T_a, we do not need to capture all the similarities between T_a and any other trajectories in the training set in case of overfitting. We adopt a unique sampling strategy to sample n similar trajectories and n dissimilar trajectories to formulate the list-wise ranking loss to learn the similarity.

A. Data Preprocessing

Due to the ubiquity of noise in collected trajectories, we attempt to preprocess the data to avoid the negative effect of noise. Our certain solution is to address noise and then replace it through Lagrange interpolation.

To locate noise, we simply compare the neighboring points by setting a distance threshold. Due to the limitation of the mobile ability and road network, the object cannot move too far in a certain time (e.g., a car should be positioned on the road, and the distance between two adjacent sampling points is limited). We name it a noise point if a point locates in an impossible position or the distance exceeds threshold. We consider the first sampling point not noise by default. We then interpolate the points on either side of the noise point by Lagrange interpolation. Details are as follows.

Interpolation is a kind of curve fitting method, which aims to find a polynomial curve to best fit the discrete points. In an interval $[a, b]$, given n discrete points (x_k, y_k), $k = 1, 2, \ldots, n$, find the corresponding $\boldsymbol{y}$ for every $\boldsymbol{x}$. $u(x)$ is defined in $[a, b]$, and

$$u(x_k) = y_k \qquad k = 1, 2, \ldots, n. \tag{1}$$

Then, we need to find $v(x)$, for the n discrete points

$$v(x_k) = u(x_k) \qquad k = 1, 2, \ldots, n. \tag{2}$$

We name $v(x)$ an interpolation of $u(x)$ in $[a, b]$. In most cases, the polynomial curve is applied when the fitting operation happens. The polynomial curve is much more convenient to conduct for the given discrete points. The sine-like polynomial curve is more stable due to being closer to reality.

According to the definition, we know that the interpolation would include all the given discrete points without abandoning any of them. In common trajectory data set, most trajectories are collections of only dozens of latitude/longitude coordinates. For the anchor noise, sampling points of either side

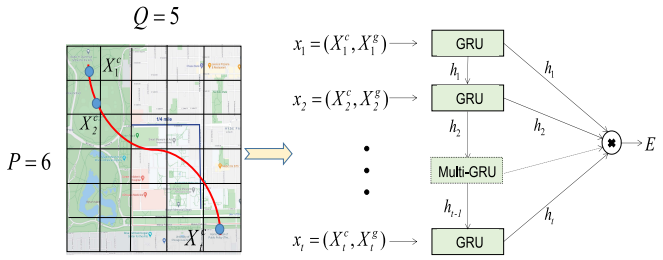


Fig. 4. Sequence encoder.

are needed. We should fully exploit the information of noise' context.

So in this article, we adopt *Lagrange interpolation*, i.e., a sort of polynomial interpolation. In mathematics, Lagrange interpolation finds a polynomial that happens to take the observed values at each observed point. That means, it can provide a polynomial function that happens to cross all the given points in a 2-D plane.

Let set $\boldsymbol{D}_n$ be the collection of angular coordinates of the points (x_k, y_k), $k = 1, 2, \ldots, n$, $\boldsymbol{D}_n = \{1, 2, \ldots, n\}$, there are n polynomials $p_j(x)$, $j \in \boldsymbol{D}_n \quad \forall k \in \boldsymbol{D}_n$

$$p_k(x) = \prod_{i \in \boldsymbol{B}_k} \frac{x - x_i}{x_k - x_i} \tag{3}$$

where $\boldsymbol{B}_k = \{i | i \neq k, i \in \boldsymbol{D}_n\}$. It is obvious the degree of $p_k(x)$ is $n - 1$. Finally, we will get the Lagrange interpolation

$$\text{Lag}_n(x) = \sum_{j=1}^{n} y_j p_j(x). \tag{4}$$

As the operation above, the Lagrange interpolation has been conducted. Concretely, for the anchor noise point, we exploit its context to conduct Lagrange interpolation fitting curve and replace the noise locally. After preprocessing raw trajectory data, we then encode trajectories with deep representation learning.

B. Deep Representation Learning

As shown in Fig. 4, to encode a trajectory as a representation vector, we first divide the 2-D Euclidean space into $P \times Q$ parts. Then any trajectory $T = [X_1^c, \ldots, X_t^c, \ldots]$ can be mapped into a sequence $T^g = [X_1^g, \ldots, X_t^g, \ldots]$ where $X_t^g = (x_t^g, y_t^g)$ specifies the part at the tth position. At each t-time step, we store the embedding vector of cell (p, q). Combining with the memory tensor proposed by Yao *et al.* [18], we can scan the surround of (p, q) and retrieve the information for later forward computing. We take the last hidden state of GRU unit as the embedding vector of the trajectory. At each time step t, input $X_t = (X_t^c, X_t^g)$ and previous hidden state $\mathbf{h}_{t-1}$ and output $\mathbf{h}_t$ to the next recurrent step

$$(X_t^c, X_t^g, \mathbf{h}_{t-1}) \longrightarrow \mathbf{h}_t. \tag{5}$$

In the GRU unit, memory-augmented GRU employs a gating mechanism to control the operation on the cell. The recurrent step is performed as follows:

$$(\mathbf{r}_t, \mathbf{s}_t, \mathbf{u}_t)^T = \text{sigmoid}(\mathbf{W}_g \cdot [X_t^c, \mathbf{h}_{t-1}] + \mathbf{b}_g) \tag{6}$$

Algorithm 1 Deep Representation Learning

Require: A raw trajectory $T = [X_1^c, ..., X_t^c, ...]$;
Ensure: $\mathbf{E}$: Embedding of T;
1: Divide space into small cells, and map T into a sequence $T^g = [X_1^g, .., X_t^g, ...]$;
2: **for** each time step $t \in T$ **do**
3: Input $X_t = (X_t^c, X_t^g)$ and last hidden state h_{t-1} to GRU;
4: Calculate a temporal state $\tilde{\mathbf{h}}_t$ according to equation (6)(7);
5: Retrieve information of previous trajectories from memory and calculate cell state according to equation (8);
6: Store the cell state into the Memory;
7: Calculate hidden state h_t according to equation (10);
8: **end for**
9: Take the last hidden state h as embedding $\mathbf{E}$;

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c \cdot [X_t^c, \mathbf{r}_t \cdot \mathbf{h}_{t-1}] + \mathbf{b}_c) \tag{7}$$

where $(\mathbf{r}_t, \mathbf{s}_t, \mathbf{u}_t, \tilde{\mathbf{h}}_t)$ are reset gate, spatial gate, update gate, and temporal state of cell, and $\mathbf{W}_g \in R^{4d \times (d+2)}$,$\mathbf{W}_c \in R^{d \times (d+2)}$, $\mathbf{b}_g$ and $\mathbf{b}_c$ are weight bias, and d is the hidden state size.

To get the hidden state $\mathbf{h}_t$, the unit exploits gating operation to generate the medium cell state at the current time step t.

$$\mathbf{c}_t = \mathbf{u}_t \cdot \tilde{\mathbf{h}}_t + \mathbf{s}_t \cdot \text{read}(\mathbf{u}_t \cdot \tilde{\mathbf{h}}_t, X_t^g, \mathbf{M}) \tag{8}$$

$$\text{write}(\mathbf{c}_t, \mathbf{s}_t, X_t^g, \mathbf{M}) \tag{9}$$

where $\hat{\mathbf{c}}_t$ and $\mathbf{c}_t$ are medium cell state, and $\mathbf{M}$ is the memory tensor accompanied with two operations: 1) *read* and 2) *write*. Based on the medium cell date and the gating operation, the last hidden state $\mathbf{h}_t$ to the next recurrent unit can be generated by

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{u}_t) \cdot \mathbf{h}_{t-1} + \mathbf{u}_t \cdot \mathbf{c}_t \tag{10}$$

where $(\mathbf{h}_{t-1}, \mathbf{h}_t)$ have the same shape: R^d. The whole procedure of embedding acquisition can be seen in Algorithm 1.

All embeddings parts are initialized with 0 before training, then the GRU unit performs the following three steps to generate the last hidden state $\mathbf{h}_t$ as trajectory's embedding vector: 1) the GRU unit takes X_t^c, and previous hidden state $\mathbf{h}_{t-1}$ as input to obtain three gates and medium state $\tilde{\mathbf{h}}_t$, as shown in (6) and (7); 2) the unit produces cell state of the current time step $\mathbf{c}_t$ based on the $\mathbf{u}_t$, $\mathbf{s}_t$, $\tilde{\mathbf{h}}_t$ and the inputs X_g^c, as shown in (8); and 3) by (10), the unit generates the last hidden state $\mathbf{h}_t$ and output it to the next recurrent step.

C. Learning to Rank With Weighted Sampling

For any input trajectory, the final hidden state of our GRU encoder is used as the trajectory representation, i.e., trajectories are embedded into d-dimensional space. Giving a set of training trajectories, we randomly sample N trajectories and generate an $N \times N$ symmetric similarity matrix M With the predefined similarity function $f(\cdot, \cdot)$. As the magnitude of raw

distance may span a large range due to power-law distribution, we normalize M to a similarity matrix S as follows:

$$\mathbf{S}_{i,j} = \frac{\exp\left(-\alpha \cdot \mathbf{M}_{i,j}\right)}{\sum_{n=1}^{N} \exp\left(-\alpha \cdot \mathbf{M}_{i,n}\right)} \tag{11}$$

where α is a tunable parameter controlling the similarity value distribution.

Based on the similarity matrix of the training data, our problem is to learn the exact inside similarity via the list-wise L2R model. Formally, let two d-dimensional vectors $\mathbf{E}_i$ and $\mathbf{E}_j$ denote encoding vectors of T_i and T_j, $i, j \in [1, \ldots, N]$, we aim to learn the similarity between $\mathbf{E}_i$ and $\mathbf{E}_j$ based on the similarity matrix S by minimizing the distance between the similarity function $g(\mathbf{E}_i, \mathbf{E}_j)$ and predefined function $f(T_i, T_j)$. Among all the N^2 pairs from the training data set, the importance of different pairs varies and we also pick K-related pairs with reasonable weight to fit the similarity. We set the loss function of the learning procedure as

$$\min \sum_{k=1}^{K} w_k \cdot \left(f\left(T_i, T_j\right) - g\left(\mathbf{E}_i, \mathbf{E}_j\right)\right)^2 \tag{12}$$

where w_k is the weight of pair k. Then, we construct the weight-based L2R model to sample a set of similarity list Γ_a^s as well as dissimilarity list Γ_a^d for anchor trajectory T_a.

To preserve the ranking order in both similar and dissimilar pairs. First, we randomly select a few trajectories from the training data set as anchor trajectories. For an anchor trajectory, we take the corresponding row from the similarity matrix S as the importance vector $\mathbf{I}_a$. With such vector $\mathbf{I}_a$, we can sample n different trajectories as the similarity list $\Gamma_a^s = \{T_1^s, \ldots, T_n^s\}$ in decreasing order. On the contrary, we can get $\mathbf{1}-\mathbf{I}_a$ as the importance weight to sample another n different trajectories as the dissimilarity list $\Gamma_a^d = \{T_1^d, \ldots, T_n^d\}$ in increasing order.

After sampling, we first generate the trajectory embeddings through the deep representation learning. Then, we will obtain $2n$ pairs for anchor trajectories T_a and define the list-wise similarity on the similarity list and dissimilarity list. For similarity list, we compute similarity $g(\mathbf{E}_a, \mathbf{E}^s)$ between T_a' embedding and every embedding in this list, where $\mathbf{E}_a$ is the embedding of T_a, $\mathbf{E}^s$ is one of the embeddings in similarity list and $g(\mathbf{E}_a, \mathbf{E}^s) = \exp[-\text{Euclidean}(\mathbf{E}_a, \mathbf{E}^s)]$, so we can get n values after such operation. The same to the dissimilarity list, we also compute the similarity between T_a's embedding and every embedding in the dissimilarity list under the same function. We get another n values. The $2n$ values are used to optimize the loss function.

For every n sampled trajectories, we set their ranking weights as $\mathbf{r} = (1, 1/e, \ldots, 1/e^l, \ldots, 1/e^n)$ and normalize the weights by $\sum_{l=1}^{n} r_l$. For the n similar pairs, their weights decrease with the ranking order. Using the list-wise ranking model, we define the loss for similar pairs of T_a as

$$L_a^s = \sum_{l=1}^{n} \mathbf{r}_l \cdot \left(g\left(\mathbf{E}_a, \mathbf{E}_l^s\right) - f(T_a, T_l^s)\right)^2 \tag{13}$$

where T_l^s is the lth trajectory in the similarity list and $\mathbf{E}_l^s$ is the embedding vector of T_l^s, $f(T_a, T_l^s)$ is the ground-truth similarity between trajectories T_a and T_l^s.

Since learning the similarity from the dissimilarity list is meaningless. We focus on the margin loss to separate dissimilar trajectories from the anchor trajectory

$$L_a^d = \sum_{l=1}^{n} \mathbf{r}_l \cdot \left[\text{ReLU}\left(g(\mathbf{E}_a, \mathbf{E}_l^d) - f(T_a, T_l^d)\right)\right]^2 \tag{14}$$

where T_l^d is the lth trajectory in the dissimilarity list, $\mathbf{E}_l^d$ is its embedding vector, and $\text{ReLU}(X) = \max(0, x)$. If $g(\mathbf{E}_a, \mathbf{E}_l^d) - f(\mathbf{E}_a, \mathbf{E}_l^d) < 0$, then $L_a^d = 0$, which means that dissimilar sample is faraway enough from the anchor trajectory and it should not affect the final result. If $g(\mathbf{E}_a, \mathbf{E}_l^d) - f(\mathbf{E}_a, \mathbf{E}_l^d) > 0$, then $L_a^d > 0$, which means the embeddings should be adjusted to enlarge the distance of the dissimilar sample of the anchor. Finally, for the training data set, the total loss is defined as follows:

$$\mathbf{L} = \sum_{a \in [1, \ldots, N]} \left(L_a^s + L_a^d\right). \tag{15}$$

In the training process, we update the parameters with back-propagation through time algorithm to optimize the model.

D. Time Complexity

The time complexity of our method to compute the similarity include two parts: 1) the deep representation learning and 2) list-wise L2R. For the deep representation part, the GRU unit can be trained completely unsupervised with a gradient descent algorithm.

For a trained model, the higher order term of complexity in classic recurrent units is $|c| = (m + 1) * d^2$, where m is the number of gates and d is a constant indicating dimension. Thus, it only requires $O(n + |c|)$ to embed a trajectory with length n into a vector. After training the list-wise ranking part offline, the complexity of similarity computation between two embedding vectors is a constant. Therefore, the time complexity of calculating the similarity between two trajectories is $O(n + |c|))$, which is linear in the length of the trajectories.

V. Experiment

In this section, we report the results of a series of experiments conducted to evaluate the performance of the proposed method using a large-scale vehicle trajectory data set. We first describe the settings of experiments, including data sets, comparative methods, and evaluation metric. Then, we report and discuss the experimental results.

A. Experimental Setup

Data Set: Our experiments are conducted on a real-world taxi data set.[1] As seen in Table III, it is collected in the city of Beijing, China and contains over 1.7 thousands trajectories. Each taxi reports its location at 15 s intervals, and the total

[1] http://www.geolink.pt/ecmlpkdd2015-challenge

TABLE III
DATA SET STATISTICS

Dataset	Trips	Mileage	Driving Time
Beijing	17,621	1,200,000km	48,000h

mileage is over 1.2 million kilometers. To reduce the contingency, we remove the trajectories which are less than 30 points.

Benchmarking Methods: We compare our method with four most commonly used methods for top-k trajectory similarity search problem, namely, DTW [4], EDR [19], ERP [7], and NEUTRAJ [18]. EDR is one of the most widely adopted trajectory measures in spatiotemporal data analyses. It adopts the edit distance to capture inside information, which is based on the point-to-point match to some certain extent. EDR seeks the matched points pairs as much as possible, abandoning the regularity of the points sequence in the x-axis, leading to inaccuracy. DTW measures the similarity by warping the sequence to best achieve feature to feature aligned. But in the trajectory similarity computation, the warping operation may distort the original features of the points sequence, which would also result in the inaccuracy. ERP is a metric thus can cope with some indexing problems of other methods and supports local time shifting, which satisfies the triangle inequality. As for NEUTRAJ, it proposes a kind of memory-augmented LSTM unit and neural metric learning to accelerate the trajectory similarity computation.

Evaluation Platform: Our method is implemented in Python 2.7 and pytorch running in Ubuntu 14.01 with an Intel Core i7-7700 CPU and 1080-Ti GPU.

B. Performance Evaluation

In this experiment, each method aims to find the exact top-k results based on the calculated similarity. The lack of ground-truth data set makes it a challenging problem to evaluate the accuracy of trajectory similarity. In this experiment, we tackle this problem from a novel aspect. First, we construct a ground-truth result set by computing the k-NN list for a randomly chosen query trajectory with a predefined similarity function. Then, we recompute the k-NN list for the same query trajectory with different comparison methods and calculate the experiment metric based on the ground-truth k-NN list and the recomputed k-NN list. To analyze the performance of our method and other methods, we set many contrast experiments in accuracy and efficiency.

We use the widely used top-k hitting ratio to evaluate the proposed similarity computation method for vehicle trajectory, which examines the overlap percentage of the top-k results and the ground truth. we randomly choose q_t trajectories as query and 10 000 trajectories as the target database from the test data set. We apply each method to find the k-nearest-neighbors (knn) of each query trajectory from the target database as its ground truth. We report the hitting ratio for both top-10 and top-50 searches for all experiments. For every trajectory T, sample an N-length list of trajectories and compute the similarity between T and the list with a predefined

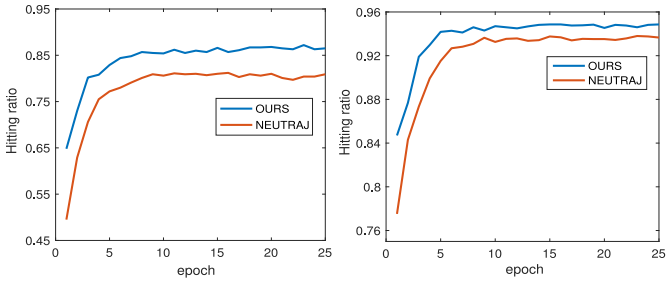


Fig. 5. Comparison between our method and NEUTRAJ. (a) Top-10. (b) Top-50.

similarity function. Then, sort the list according to the similarity values. Generate a top-k list by picking the top-k ranked trajectories from the list. Let $\#hit@k$ denote a single test case as either the value is 1 if the similar trajectory t_i appears in the top-k results, or else the value is 0. The overall top-k hitting ratio is defined by averaging all test cases

$$\text{top}-k = \frac{\Sigma \#hit@k}{|TE|} \tag{16}$$

where $|TE|$ is the number of all test cases.

1) Experiment in Accuracy: By generating the predefined ground truth with the Hausdorff distance and setting 1,800 trajectories as the target search database from the test data set, we compare the performance of our method and NEUTRAJ by increasing the number of epochs from 1 to 25, the results are shown in Fig. 5. From this figure, we have the following observations.

1) Our method and NEUTRAJ converge rapidly in nearly ten epochs for both top-10 and top-50 hitting ratio comparing.
2) The proposed method that utilizes a list-wise ranking learning model to learn the trajectory similarity always outperforms NEUTRAJ.

For example, the top-10 hitting ratio of NEUTRAJ is 80.9% when epoch is 12, while 85.5% for our method for the same epoch that means the performance is improved by 5.69% compared with NEUTRAJ. The results suggest that the proposed Lagrange interpolation method can effectively denoise trajectories, and the proposed L2R model by exploiting exponentially decreasing weights for both similarity and dissimilarity lists can boost trajectory similarity calculation. In fact, a small improvement in trajectory similarity calculation is likely to lead to a significant increase with such a huge vehicle trajectory data set. Therefore, the performance improvement of our method is significant, which clearly demonstrates the effectiveness of the proposed method.

We further investigate the performance with three different ground-truth measures (i.e., DTW, Fréchet, and SSPD [31]) for our method and NEUTRAJ, the results are reported in Fig. 6. Note we artificially suspend an experiment if it converges already for time-saving considering. From this figure, we observe the following.

1) Our method performs better than NEUTRAJ with all the ground-truth measures, for instance, the top-50 hitting ratio of our method is 93.56% when epoch is equal to 15 and 90.88% for NEUTRAJ for the same epoch with

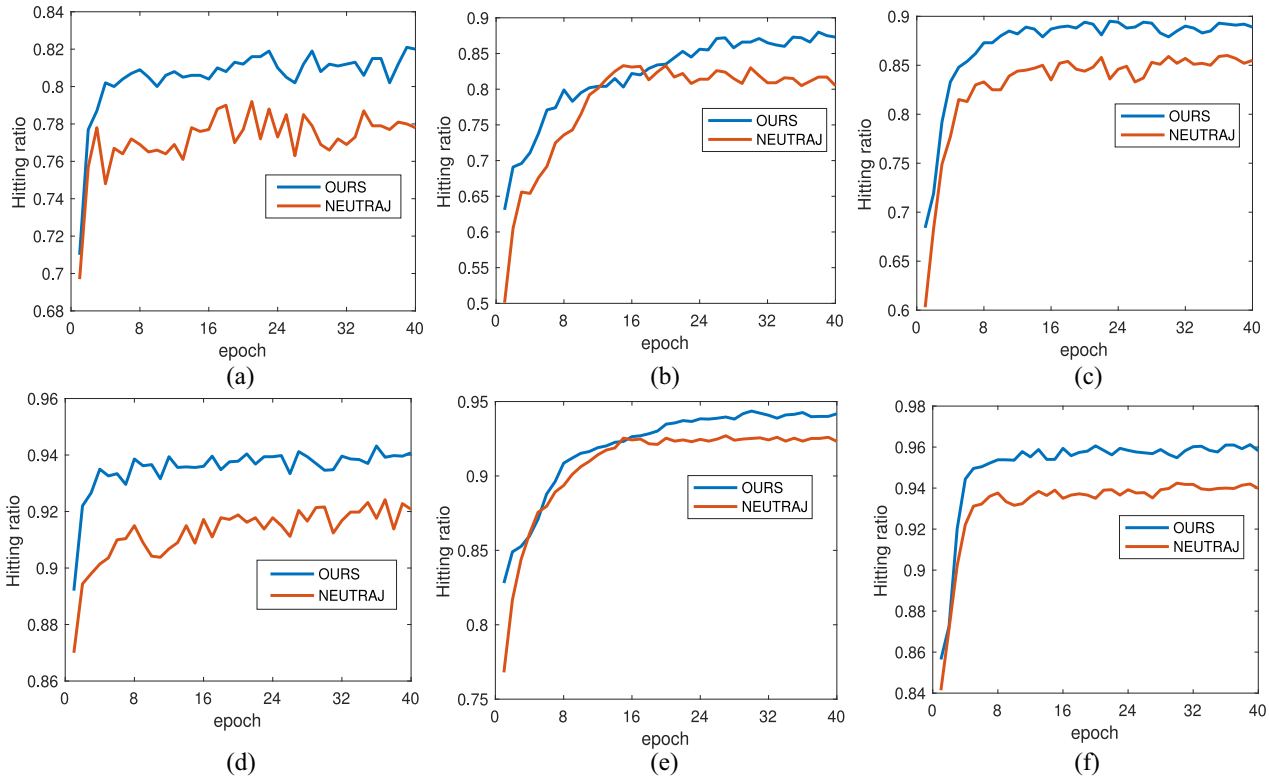


Fig. 6. Comparison between our method and NEUTRAJ. (a) DTW (Top-10). (b) Frechet (Top-10). (c) SSPD (Top-10). (d) DTW (Top-50). (e) Frechet (Top-50). (f) SSPD (Top-50).

TABLE IV
COMPARISON BETWEEN OUR METHOD AND OTHERS IN DIFFERENT GROUND-TRUTH METHODS WITH TEST SIZE 100, 500, AND 1000

Ground truth			Hausdorff				Fréchet				SSPD			
			DTW	EDR	ERP	OURS	DTW	EDR	ERP	OURS	DTW	EDR	ERP	OURS
Test trajectories	100	Top-10	75.8	89.4	83.7	**94.8**	71.8	90.3	86.2	**94.5**	69.3	85.3	73.7	**89.1**
		Top-50	69.4	81.7	79.1	**86.5**	63.3	84.3	82.2	**86.7**	76.3	89.3	83.4	**96.1**
	500	Top-10	69.8	81.4	79.6	**86.4**	63.4	84.3	82.1	**87.2**	69.1	90.2	83.5	**96.0**
		Top-50	75.9	89.7	83.9	**94.8**	71.7	90.4	86.1	**94.6**	76.7	85.6	73.5	**89.0**
	1000	Top-10	70.2	82.1	79.4	**86.5**	63.0	84.1	82.4	**86.8**	69.2	89.7	83.8	**96.1**
		Top-50	75.6	89.6	84.0	**94.0**	71.6	90.6	96.4	**94.6**	76.9	85.2	73.8	**89.1**

using DTW as a ground-truth measure, which means that our method has improved performance by 2.9% compared to NEUTRAJ. The result suggests again that the L2R model by exploiting exponentially decreasing weights for both similarity and dissimilarity lists can improve the similarity calculation performance.

2) Both our method oscillates more and the speed of convergence is as same as NEUTRAJ when the ground-truth measure is Fréchet, but our method still performs better when the model training converges. From Fig. 6(a) and (d), when the ground-truth measure is changed to DTW, our method converges quicker and achieves higher accuracy than NEUTRAJ on both top-10 and top-50.

We also compare the performance of our method with other traditional similarity measures (DTW, ERP, and EDR) on the massive vehicle trajectory data set, as shown in Table IV. We can observe the following from this table.

1) For all comparative experiments using different ground-truth measures, the proposed method by encoding a trajectory as a vector via deep representation learning always outperforms the baseline methods (DTW, ERP, and EDR), which first form a pairwise matching for the sample points of two trajectories then identify the best alignment using dynamic programming. For 100 test trajectories, our method outperforms DTW by 19.3%, ERP% by 12.5%, and EDR% by 5.8% by utilizing SSPD as a ground-truth measure in terms of the top-50 hitting ratio, respectively. This result suggests that using learning for creating representations of trajectories can improve the performance of trajectory similarity computation than traditional methods by means of pairwise point-matching.
2) For traditional similarity measures, EDR achieves much better performance than DTW and ERP in terms of both top-10 hitting ratio and top-50 hitting ratio. The results suggest that EDR is robust to noise, shifts and scaling of data that commonly occur due to sensor failures.

2) Experiment in Efficiency: We investigate the efficiency of the proposed method in this section. Generally, the time complexity of the proposed method consists of two parts, i.e.,

TABLE V
TIME COST FOR ONLINE TRAJECTORY SIMILARITY COMPUTATION

Method	100	500	1000
		Hausdorff	
DTW	2.310s	11.329s	21.443s
ERP	8.980s	46.014s	77.219s
EDR	3.143s	17.318s	34.283s
OURS	**0.056s**	**0.078s**	**0.146s**
		Fréchet	
DTW	2.014s	9.346s	21.653s
ERP	3.152s	15.135s	30.137s
EDR	3.275s	13.541s	29.521s
OURS	**0.059s**	**0.120s**	**0.193s**
		SSPD	
DTW	1.534s	7.501s	16.437s
ERP	9.431s	39.315s	81.255s
EDR	3.174s	14.359s	31.254s
OURS	**0.051s**	**0.095s**	**0.179s**

TABLE VI
HYPERPARAMETER STUDY ON THE ACTIVATE FUNCTION, THE NUMBER OF NEURONS PER LAYER, AND THE RECURRENT UNIT

		top-10	top-50
Activate Fuction	Leaky_relu	0.824	0.931
	Relu	0.823	0.933
	Tanhshrink	0.825	0.933
	Tanh	0.823	0.939
Num of Neuron	unit=128	0.823	0.939
	unit=228	0.832	0.947
	unit=428	0.831	0.941
	unit=628	0.815	0.938
	unit=828	0.813	0.934
Recurrent Unit	GRU	0.826	0.940
	LSTM	0.815	0.943

the deep representation learning and list-wise L2R. The deep representation learning part can be trained completely unsupervised with a gradient descent algorithm in the offline phase. For a trained model, it only requires linear time to encode a trajectory into a vector. Therefore, the time complexity of calculating the similarity between two trajectories is linear in the length of trajectories. Then, we report the run time of the graph embedding process for extracting the object's social similarity in Table V. From this table, we can observe time complexity is generally linear in terms of the number of test trajectories as expected, which shows the proposed approach is scalability to the large data set.

C. Hyperparameter Study

To further estimate the accuracy of our method, we adjust the parameter settings to promote our model. We study the influence of the hyperparameter toward our model in the following three ways:

1) activation function;
2) the number of neurons per layer;
3) recurrent unit.

The experiment results are as seen in Table VI.

1) Activate Function: We consider various activate functions which we expect to improve our model. Under the top-10 hitting ratio indicator, four candidate functions perform no significant difference. But under the top-50 hitting ratio indicator, activate function *tanh* stands out. Albeit to a small degree, *tanh* performs better than *leaky_relu*, other than the others.

2) Number of Neurons Per Layer: When other factors remain the same, increasing the number of neurons per layer introduces complexity. Increasing the number of neurons does not always bring benefits. Under the top-10 hitting ratio indicator, the model performs unstably worse when the number of neuron units reaches 828. On the contrary, when the neuron number is 228 and 428, the model performs better than others and more stably. The same as top-10 hitting ratio, under the top-50 hitting ratio indicator, the model does not fit well when the number of units is 828. Apparently, when the number is 228, the model performs the best of all.

3) Recurrent Unit: As illustrated in the previous section, we adopt GRU as recurrent unit in our deep representation model and reach the remarkable experiment result. We consider another recurrent unit—LSTM and adopt it to replace GRU in order to promote the model. Under the top-10 hitting ratio indicator, the GRU unit performs apparently better than LSTM and the peak value is much larger. On the contrary, under the top-50 hitting ratio indicator, LSTM performs better and more stably. It somehow improves the model in a less degree.

In the above study, our model preserves the potential of self-promotion. In most cases, a suitable adjustment of hyperparameter setting improves the model to some extent. As we can figure out, the minor promotion is not usually stable but helpful.

VI. CONCLUSION

In this article, we propose an embedding-based method for trajectory similarity computation with linear time complexity which only costs some time to train a model offline in advance. The novelty of the proposed method lies in it only takes a linear time complexity to compute the similarity between a pair of trajectories, while most existing methods take quadratic computational complexity due to pairwise point matching. Specifically, the proposed model consists of two phases: 1) a deep representation learning used to encode trajectories as vectors and 2) an attention-based L2R model to learn the similarity between trajectories. Additionally, we utilize an interpolation-based approach to reduce noise and outliers by considering vehicle trajectory is physically constrained to the road network. We conduct extensive experiments on a real-world massive vehicle trajectory data set, and the results show our method outperforms state-of-the-art baselines significantly in terms of both accuracy and efficiency.

In future work, we plan to employ the trajectory similarity computation approach to explore more downstream tasks, such as trajectory clustering and popular routes search.

REFERENCES

[1] C. Chen, C. Chen, C. Xiang, S. Guo, Z. Wang, and B. Guo, "ToiletBuilder: A PU learning based model for selecting new public toilet locations," *IEEE Internet Things J.*, vol. 8, no. 9, pp. 7531–7545, May 2021.

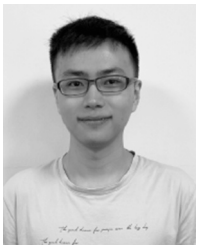
[2] C. Chen *et al.*, "iBOAT: Isolation-based online anomalous trajectory detection," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 2, pp. 806–818, Jun. 2013.
[3] C. Chen *et al.*, "Crowddeliver: Planning city-wide package delivery paths leveraging the crowd of taxis," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 6, pp. 1478–1496, Jun. 2017.
[4] B.-K. Yi, H. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *Proc. 14th Int. Conf. Data Eng.*, 1998, pp. 201–208.
[5] J.-F. Hangouet, "Computation of the hausdorff distance between plane vector polylines," in *Proc. Conf. AutoCarto*, 1995, pp. 1–10.
[6] H. Alt and M. Godau, "Computing the Fréchet distance between two polygonal curves," *Int. J. Comput. Geom. Appl.*, vol. 5, no. 1, pp. 75–91, 1995.
[7] L. Chen and R. T. Ng, "On the marriage of Lp-norms and edit distance," in *Proc. VLDB*, 2004, pp. 792–803.
[8] N. Ta, G. Li, Y. Xie, C. Li, S. Hao, and J. Feng, "Signature-based trajectory similarity join," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 4, pp. 870–883, Apr. 2017.
[9] H. Su, K. Zheng, H. Wang, J. Huang, and X. Zhou, "Calibrating trajectory data for similarity-based analysis," in *Proc. SIGMOD Conf.*, 2013, pp. 833–844.
[10] S. Shang, R. Ding, K. Zheng, C. S. Jensen, P. Kalnis, and X. Zhou, "Personalized trajectory matching in spatial networks," *VLDB J.*, vol. 23, no. 3, pp. 449–468, 2014.
[11] A. C. Sanderson and A. K. C. Wong, "Pattern trajectory analysis of nonstationary multivariate data," *IEEE Trans. Syst., Man, Cybern.*, vol. 10, no. 7, pp. 384–392, Jul. 1980.
[12] H. Li, J. Liu, K. Wu, Z. Yang, R. Liu, and N. Xiong, "Spatio-temporal vessel trajectory clustering based on data mapping and density," *IEEE Access*, vol. 6, pp. 58939–58954, 2018.
[13] C. Myers, L. Rabiner, and A. Rosenberg, "Performance tradeoffs in dynamic time warping algorithms for isolated word recognition," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 28, no. 6, pp. 623–635, Dec. 1980.
[14] T. Wang, L. Qiu, A. K. Sangaiah, A. Liu, M. Z. A. Bhuiyan, and Y. Ma, "Edge-computing-based trustworthy data collection model in the Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4218–4227, May 2020.
[15] T. Wang, H. Ke, X. Zheng, K. Wang, A. K. Sangaiah, and A. Liu, "Big data cleaning based on mobile edge computing in industrial sensor-cloud," *IEEE Trans. Ind. Informat.*, vol. 16, no. 2, pp. 1321–1329, Feb. 2020.
[16] T. Wang *et al.* "Privacy-enhanced data collection based on deep learning for Internet of Vehicles," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6663–6672, Oct. 2020.
[17] T. Wang, Y. Lu, J. Wang, H.-N. Dai, X. Zheng, and W. Jia, "EIHDP: Edge-intelligent hierarchical dynamic pricing based on cloud-edge-client collaboration for IoT systems," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1285–1298, Aug. 2021.
[18] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, 2019, pp. 1358–1369.
[19] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proc. SIGMOD Conf.*, 2005, pp. 491–502.
[20] E. Frentzos, K. Gratsias, and Y. Theodoridis, "Index-based most similar trajectory search," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, 2007, pp. 816–825.
[21] S. Sankararaman, P. Agarwal, T. Mølhave, J. Pan, and A. P. Boedihardjo, "Model-driven matching and segmentation of trajectories," in *Proc. 21st ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inf. Syst.*, 2013, pp. 234–243.
[22] H. Wang, H. Su, K. Zheng, S. Sadiq, and X. Zhou, "An effectiveness study on trajectory similarity measures," in *Proc. ADC*, 2013, pp. 13–22.
[23] S. Ranu, P. Deepak, A. Telang, P. Deshpande, and S. Raghavan, "Indexing and matching trajectories under inconsistent sampling rates," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 999–1010.
[24] K. Toohey and M. Duckham, "Trajectory similarity measures," *ACM SIGSPATIAL Special*, vol. 7, no. 1, pp. 43–50, 2015.
[25] M. Vlachos, D. Gunopulos, and G. Kollios, "Discovering similar multidimensional trajectories," *Proc. 18th Int. Conf. Data Eng.*, 2002, pp. 673–684.
[26] K. Zheng, Y. Zheng, X. Xie, and X. Zhou, "Reducing uncertainty of low-sampling-rate trajectories," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 1144–1155.
[27] C. Chen, Y. Ding, X. Xie, S. Zhang, Z. Wang, and L. Feng, "TrajCompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 2012–2028, May 2020.
[28] P. Banerjee, S. Ranu, and S. Raghavan, "Inferring uncertain trajectories from partial observations," in *Proc. IEEE Int. Conf. Data Min.*, 2014, pp. 30–39.
[29] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, 2018, pp. 617–628.
[30] C. F. Torres and R. Trujillo-Rasua, "The Fréchet/Manhattan distance and the trajectory anonymisation problem," in *Proc. DBSec*, 2016, pp. 19–34.
[31] J. Froehlich and J. Krumm, "Route prediction from trip observations," SAE Int., Warrendale, PA, USA, Rep. 2008-01-0201, 2008.

Yuanyi Chen received the B.Sc. degree from Sichuan University, Chengdu, China, in 2010, the M.Sc. degree from Zhejiang University City College, Hangzhou, China, in 2013, and the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 2017.

He is currently a Distinguished Research Fellow with the Department of Computer Science and Computing, Zhejiang University City College. He has published more than 20 technical papers in major international journals and conference proceedings. His research interests include the Internet of Things, mobile computing, and ubiquitous computing.

Peng Yu received the B.S. degree in computer science from Soochow University, Suzhou, China, in 2020. He is currently pursuing the master's degree with the College of Computer Science and Technology, Zhejiang University City College, Hangzhou, China.

His research topics include the Internet of Things and mobile-edge computing.



Wenwang Chen received the B.S. degree from the School of Mathematics, Tianjin University, Tianjin, China, in 2018. He is currently pursuing the master's degree with the College of Computer Science and Technology, Zhejiang University City College, Hangzhou, China.

His research topics include data mining and signal processing.

Zengwei Zheng received the B.S. and master's degrees in computer science and western economics from Hangzhou University, Hangzhou, China, in 1991 and 1998, respectively, and the Ph.D. degree in computer science and technology from Zhejiang University City College, Hangzhou, in 2005.

He is currently a Professor with the School of Computer and Computing Science and the Director of the Intelligent Plant Factory of Zhejiang Province Engineering Laboratory and the Hangzhou Key Laboratory for IoT Technology and Application, Zhejiang University City College. His research interests include wireless sensor networks, location-based service, Internet of Things, digital agriculture, and pervasive computing.

Prof. Zheng is a Senior Member of CCF.

Minyi Guo (Fellow, IEEE) received the B.Sc. and M.E. degrees in computer science from Nanjing University, Nanjing, China, in 1982 and 1986, respectively, and the Ph.D. degree in information science from the University of Tsukuba, Tsukuba, Japan, in 1998.

He is currently the Zhiyuan Chair Professor and the Chair of the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China.

Dr. Guo received the National Science Fund for Distinguished Young Scholars Award from NSFC in 2007.