Can Adversarial Training benefit Trajectory Representation? An Investigation on Robustness for Trajectory Similarity Computation

Quanliang Jing
Shuo Liu
jingquanliang@ict.ac.cn
liushuo22@mails.ucas.ac.cn
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

Xinxin Fan
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
fanxinxin@ict.ac.cn

Jingwei Li
Department of Computer Science and Engineering, University at Buffalo, SUNY
Buffalo, New York, USA
jli379@buffalo.edu

Di Yao
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
yaodi@ict.ac.cn

Baoli Wang
Microsoft Search Technology Center Asia
Beijing, China
baoli.wang@microsoft.com

Jingping Bi
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
bjp@ict.ac.cn

ABSTRACT

Trajectory similarity computation as the fundamental problem for various downstream analytic tasks, such as trajectory classification and clustering, has been extensively studied in recent years. However, how to infer an accurate and robust similarity over two trajectories is difficult due to some trajectory characteristics in practice, e.g. non-uniform sampling rate, nonmalignant fluctuation, noise points, etc. To circumvent such challenges, we in this paper introduce the adversarial training idea into the trajectory representation learning for the first time to enhance the robustness and accuracy. Specifically, our proposed method AdvTraj2Vec has two novelties: i) it perturbs the weight parameters of embedding layers to learn a robust model to infer an accurate pairwise similarity over each two trajectories; ii) it employs the GAN momentum to harness the perturbation extent to which an appropriate trajectory representation can be learned for the similarity computation. Extensive experiments using two real-world trajectory datasets Porto and Beijing validate our proposed AdvTraj2Vec on the robustness and accuracy aspects. The multi-facet results show that our AdvTraj2Vec significantly outperforms the state-of-the-art methods in terms of different distortions, such as trajectory-point addition, deletion, disturbance, and outlier injection.

CCS CONCEPTS

• **Information systems** → *Location based services.*

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10.
https://doi.org/10.1145/3511808.3557250

KEYWORDS

deep representation learning, trajectory similarity Computation, adversarial learning

ACM Reference Format:
Quanliang Jing, Shuo Liu, Xinxin Fan, Jingwei Li, Di Yao, Baoli Wang, and Jingping Bi. 2022. Can Adversarial Training benefit Trajectory Representation? An Investigation on Robustness for Trajectory Similarity Computation. In *Proceedings of the 31st ACM Int'l Conference on Information and Knowledge Management (CIKM '22), Oct. 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557250

1 INTRODUCTION

A massive amount of trajectory data has been increasingly produced with the real-world applications of many smart devices, like diversities of IoT sensors, location-aware mobile ends, GPS devices, etc. Accordingly, how to mine the latent valuable information from these trajectories has become the fundamental research spot in recent years. In many trajectory-oriented domains, such as clustering, classification, anomaly detection, and regular prediction, the trajectory similarity computation (TSC) as the core technique plays the basic role to perform various studies. For instance, the TSC can be utilized to find the potentially infected people who have a similar trajectory with a COVID-19 infector. Beyond, TSC can be applied to other tasks, such as sport-data analysis of athletes[25], tracking migration patterns of animals [15], or urban planning [8].

At present, there mainly exist two categories of metrics on TSC: pairwise trajectory-point distance and deep learning-generated vector distance. The former usually requires a pairwise matching of trajectory points and accumulates the distance information to indicate the similarity over two trajectories, such as the representative methods Dynamic Time Warping (DTW)[28], Edit distance with Real Penalty[6], Hausdorff distance[3], Fréchet distance[2] and Edit distance on real sequences (EDR)[7]. However, these methods are

time-consuming and sensitive to noise points. The latter is based on deep learning, that is to say, each trajectory can be embedded into a vector, then compute the pairwise vector-distance as the similarity. The existing methods are t2vec[14], T3S[26], NEUTRAJ[27], etc. Compared to the pairwise trajectory-point-based TSC, the deep learning-generated vector-based TSC has a lower time cost. This is because the TSC becomes linear once the trajectory representation/embedding is completed. Nevertheless, most of the latter methods did not take the robustness factor into account except t2vec, i.e. when there exist somewhat fluctuations at some points or some outlier/anomalous points. Even though the t2vec has a consideration on the robustness through addition of limited data, its performance on similarity computation accuracy declines significantly. Our experiment shows that when each trajectory point is perturbed, and the maximum perturbation amplitude is 75m, the performance of t2vec will drop by 16%. The performance of t2vec will also decrease by 12% when we add anomalous points in the trajectory. In general, the TSC using a deep learning-generated vector has a superior performance than that using the traditional pairwise trajectory-points[14, 22], and our work aiming to enhance the robustness property falls into this category.

For the robustness concern, we think there exist several facets that could bring the challenges to TSC in terms of non-uniform sampling rate, nonmalignant fluctuation, and unexpected noise.

Non-Uniform Sampling Rate. A real-world trajectory can be depicted by different sets of data points, as shown in Figure 1(a) and Figure 1(b), arising from the non-uniform sample rate. For example, a taxi driver may flexibly alter the sampling rates given the consideration on power consumption and other personal driving habits [29], which will cause the inconsistent trajectory description/points even on the same road segment. Furthermore, even though the sampling rate is approximately uniform, the collected data points might be different yet due to the changeful speeds. These different trajectory descriptions for the same road segment incur difficulty for TSC, given that the subtle incoordination will trigger the deep learning to spawn an enormous change in the vector output.

Nonmalignant Fluctuation. Fluctuation in trajectory fingerprints is sometimes nonmalignant due to some natural reasons. For example, when traffic congestion takes place on the main road, the driver may depart the main road to the auxiliary road, and vice versa. In this situation, the TSC ought to tolerate this fluctuation trajectory and still recognize it similar to the non-fluctuation state, as shown in Figure 1(a) and Figure 1(c). That is to say, the deep learning-generated vectors for the fluctuated trajectory and non-fluctuated trajectory should have a close distance. Given the fact that deep learning is a blackbox and consists of massive parameters, hence even a little fluctuation in the input trajectory, the model will yield an obviously-changed output vector. Thereby, this kind of nonmalignant trajectory fluctuation would deteriorate the performance of the present-day deep learning pattern.

Noise Points. Considering the today's location device is usually a mobile wireless communication endpoint, it is prone to be interfered unexpectedly in communication channels, or even encountered some mischievous communication-layer attacks, such as jamming attacks, man-in-the-middle attacks, spoofing attacks, etc. Therefore, the noise points are indispensably fingerprinted for these wireless location devices whatever caused unexpectedly or

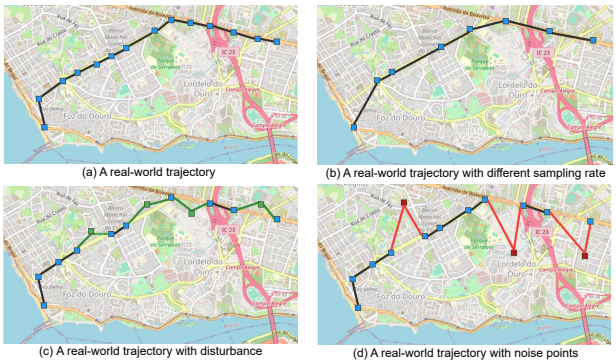

Figure 1: An illustration of different trajectory scenarios.

intendedly. They will definitely produce a side-effect on the deep learning model for the trajectory presentation learning.

With the purpose of overcoming the challenges above, we in this paper propose an adversarial learning-based trajectory representation model AdvTraj2Vec to enhance the robustness and accuracy of TSC in terms of the considerations on the inconsistent sampling rate, nonmalignant fluctuation, and unexpected and intended noise. Normally, the adversarial training is primarily applied in the computer vision (CV) field to enhance the neural network's robustness by combating various adversarial/evasion attacks targeted at image examples. Concretely, the adversarial training in CV is to employ the adversarial/perturbed examples and original examples to jointly retrain the neural network or merely use the adversarial examples without the original examples. However, due to the convenience of trajectory data processing, the space where the trajectory is located is often divided based on grid [12]. The continuous coordinates (i.e., longitude, latitude) are mapped into discrete tokens, therefore, the original trajectory becomes a sequence containing a series of discrete tokens. Given the premise that the trajectory tokens are integer values, it makes no sense to perturb the example data, thereby, our work would perform adversarial training from another viewpoint, i.e. perturb the embedding parameters rather than the trajectory data itself. Another critical problem is the perturbation amplitude, and an appropriate perturbation can promote the neural network. Hence, we adopt the idea of the generative adversarial network (GAN) to harness the perturbation amplitude.

To summarize, we conclude three main contributions as follows:

- We propose an adversarial training-based trajectory representation model AdvTraj2Vec[1] to enhance the robustness and accuracy for TSC with respect to the practical occurrences of non-uniform sampling rate, nonmalignant fluctuation, and unexpected or intended noise.
- Taking account of the perturbation amplitude, we propose a GAN-guided function to compute an appropriate amplitude for perturbation on the trajectory embedding, by which two extreme-caused harmful effects by too large or too small amplitudes will be alleviated.
- Multi-facet experiments are performed using two commonly-used real-world trajectory datasets Porto and Beijing, and

[1] Our source code and model are available at https://github.com/xiran2018/AdvTraj2Vec

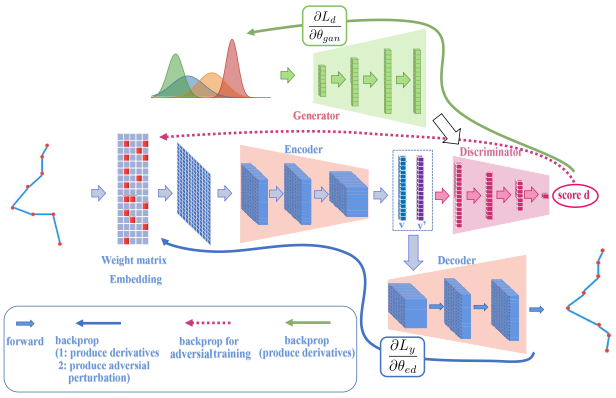

Figure 2: Architecture of the AdvTraj2Vec. It consists of two primitives. One is Encoder-Decoder which is leveraged to learn a robust representation of trajectory with adversarial training. The other is GAN module, which is used to leash perturbation amplitude on weight parameters in embedding layers in a proper extent.

the experimental results validate that our proposed AdvTraj2Vec can significantly outperform the state-of-the-art TSC methods in robustness and accuracy.

Organization. The remaining of our work is organized as follows. Section II presents the formal definitions and states the problem. We detail our AdvTraj2Vec model in Section III, report the experiment evaluation in IV, comparatively analyze the related work in Section V, and finally conclude this paper in VI.

2 PROBLEM DEFINITION

2.1 Definitions

Definition 1. *(UNDERLYING ROUTE)* An underlying route is a theoretical concept with recording the locations continuously for a moving object, it indicates the object's exact path (a successive spatial curve).

Definition 2. *(TRAJECTORY)* A trajectory is a sequence of sample points from the underlying route of a moving object.

In practice, an underlying route can be represented by a suite of trajectories. It concretely depends on the specifics of the moving object and the sampling strategies. For a referral trajectory, although different trajectory sets are acquired under diverse sampling rates, they still represent the same underlying route, that is to say, each fingerprinted trajectory can be considered as one representative of the same underlying route.

2.2 Problem Statement and Challenges

Problem Statement. Given a trajectory repository, our work aims to learn a representation $\boldsymbol{v} \in \boldsymbol{R}^n$ for each trajectory T, and the representation can appropriately elucidate the underlying route of this trajectory T for the pairwise similarity computation in a robustness way against several affections, such as non-uniform sampling rates, nonmalignant fluctuations, and noise points, etc.

Challenges. The encoder-decoder, as the popular learning framework, has been witnessed to be successful to handle sequence data in the procedure of TSC. However, when the uncertainty emerges stemming from non-uniform sampling rate, fluctuation, and point noise, the straightforward encoder-decoder cannot learn proper embedding vectors for such unexpectedly/intendedly-perturbed trajectories. To circumvent this problem, we proposed an adversarial training framework to enhance the robustness and accuracy for the trajectory similarity inference. The main novelties are twofold: i) embedding parameter-oriented adversarial training, which aims to tolerate more uncertainty cases, further for robustness promotion; ii) harnessing the amplitude of perturbation, which aims to provide insight into the extent to which the perturbation is constrained properly. Through twofold novelties, our proposed framework can learn an appropriate embedding vector for each trajectory for the robust TSC problem.

3 THE PROPOSED MODEL

3.1 Framework Overview

Two modules reside at the heart of our framework as shown in Figure 2: i) adversarial training-guided encoder-decoder. Herein the encoder is to encode the input sequence T into a fixed-dimension embedding vector $\boldsymbol{v}$, while the decoder is to decode the sequence upon the encoding representation. With the addition of adversarial training into this encoder-decoder primitive, the weight parameters of embedding vectors are perturbed in a proper range to enhance the robustness of the model; and ii) GAN-controlled perturbation amplitude. How to decide the extent to which the proper perturbation is yielded during the learning course is the critical point. Here we utilize the GAN to harness the perturbation with the guarantee that the amplitude is neither too large nor too small, achieving the generalization of the spawned model.

3.2 Adversarial Training-Guided Encoder-Decoder

Trajectory-Oriented Encoder-Decoder Procedure. As for the sequence characteristic of trajectory, the continuous coordinates ought to be mapped into discrete tokens beforehand, then they serve as input data to the encoder primitive. Herein we still use the routine spatial-date processing strategy to produce tokens, specifically, the trajectory space is partitioned into equal-size cells [13], and each cell is recognized as a token. It is worth noting that all sample points falling into the same cell will be deemed as the same token.

Normally, the encoder-decoder primitive must maximize the conditional probability $\boldsymbol{P}(R|T)$, implying that it can find the most likely underlying route R for the given trajectory T. The input of such encoder-decoder primitive is the trajectory sequence with a low sampling rate, While the output is a trajectory sequence with a high sampling rate.

In order to train a decent encoder-decoder primitive, we sample two trajectories from one trajectory with different sampling rates. The trajectory with a lower sampling rate is called T_a, and the trajectory with a higher sampling rate is called T_b. However, both the two fingerprint trajectories are the representation of the same underlying route. These observations cause us to replace the objective of maximizing $\boldsymbol{P}(R|T_a)$ with the objective of maximizing

$P(T_b|T_a)$. The encoder embeds T_a into its representation $\boldsymbol{v}$, and the decoder will try to recover its counterpart T_b with relatively high sampling rate conditioned on $\boldsymbol{v}$.

Adversarial Training towards Embedding Weight Parameters. Recall that a trajectory is elucidated by partitioning it into a suite of equal-size cells (a.k.a. tokens), and all the points falling into the same cell are recognized as the same token. Currently, each token is presented as an integer number, in other words, the input to the model must a positive integer. Thereby, we think that it makes no sense to perturb such token-symbol values, e.g. change a token value from 7 to 7.2. Because even the integer value representing the trajectory point is perturbed to become a floating point number to enhance the robustness of the model. But when the model is actually used, the inputs to the model are all integer values. That is to say, the model trained in this way cannot enhance the robustness for the actual input. Moreover, it would be difficult to create adversarial examples for the clean trajectory examples. This is because the perturbed trajectories remain agnostic whether such perturbation can still maintain the label unchanged in the case of no human intervention, providing that the label/purport is subject to the trajectory context [19].

Upon the above restriction of perturbation on actual trajectory data, we naturally switch our attention from trajectory-data space to the trajectory-embedding space, i.e. attempt to perturb the weight parameters of embedding layers of the encoder primitive, rather than the traditional actual data variation. Specifically, we utilize a gradient-based manner to produce norm-bounded adversarial perturbation on the weight parameter of embedding layers. Clearly, such embedding-level adversarial manipulation can be strictly stronger than that of token-level perturbation.

We denote the sequence of an input trajectory as $\boldsymbol{T} = [t_1, t_2, ..., t_n]$, the embedding matrix as $\boldsymbol{E}$, the encoder as a function $\boldsymbol{v} = f_\theta(\boldsymbol{X})$, and the decoder as a function $\boldsymbol{y} = f_d(f_\theta(\boldsymbol{X}))$ where $\boldsymbol{X} = \boldsymbol{ET}$ is the trajectory embeddings, $\boldsymbol{v}$ is the output of the encoder, $\boldsymbol{y}$ is the output of the decoder, and θ denotes all the encoder-decoder-related parameters including the embedding matrix $\boldsymbol{E}$.

The adversarial training-guided trajectory embedding procedure is detailed as follows. First, we compute the loss for input trajectory $\boldsymbol{T}$ and acquire the gradient g by backpropagation. Second, we define the following formula to execute adversarial perturbation $\boldsymbol{\delta'}$ for the embedding matrix (weigh parameters of embedding layers) according to the gradient information of $\boldsymbol{E}$.

$$\boldsymbol{\delta'} = \epsilon \cdot (\boldsymbol{g_e}/||\boldsymbol{g_e}||_2), \tag{1}$$

where the gradient of $\boldsymbol{E}$ is denoted as $\boldsymbol{g_e}$. The value of $\boldsymbol{E}$ is updated along the direction of the gradient to maximize the loss. The parameter ϵ is artificially determined. Third, we add $\boldsymbol{\delta'}$ to the embeddings such that the prediction becomes $\boldsymbol{y'}$=$f_d(f_\theta((\boldsymbol{E}+\boldsymbol{\delta'})\boldsymbol{T}))$. To preserve the semantics, we constrain the norm of $\boldsymbol{\delta'}$ within a reasonable range, with the purpose of warranting the model's resultant prediction cannot change largely after the perturbation. Afterwards, we can get the encoder output $\boldsymbol{v'}$.

3.3 GAN-Controlled Perturbation Amplitude

As shown in formula (1), the coefficient ϵ reflects the amplitude of the perturbation, which causes a straightforward and critical affection on the adversarial training efficiency. That is to say, excessive disturbance (too large perturbation) will have side effects on the model representation. However, the very tiny perturbation may not work. Thereby, we in this paper employ the GAN to leash the perturbation range in an adequate way, by which a rational perturbation range can be guaranteed.

As we know, the GAN consists of two parts, i.e. Generator and Discriminator. In our work, the Generator is used to capture the distribution of the encoding representation $\boldsymbol{v}$. At first, the inputs of the Generator are random seed z under the normal distribution in the course of GAN training, and the outputs are the same-dimension vectors $\boldsymbol{f}$ as $\boldsymbol{v}$. On the other side, the Discriminator is a binary classifier with the purpose of judging whether its input is "real data" or not, i.e. the original vector representation. Specifically, the inputs are $\boldsymbol{v}$ and $\boldsymbol{f}$, along with a ground truth label representing positive and negative samples. In particular, the ground truth labels of $\boldsymbol{v}$ and $\boldsymbol{f}$ are 0 and 1 respectively. The Discriminator would output a score s_d for each input to reflect the amplitude of trajectory perturbation. A big score s_d means the input of the Discriminator is unlikely to be a "real data", in other words, the perturbation amplitude for the embedding matrix $\boldsymbol{E}$ is too large, rendering an enormous dissimilarity of the inputs.

In the procedure of adversarial training, the Discriminator aims to judge how similar between encoder output $\boldsymbol{v'}$ and encoding representation $\boldsymbol{v}$. In order to look for an appropriate perturbation, we herein set a threshold a to leash the amplitude. Concretely, the Discriminator output score s_d only works for the trajectories that its perturbation amplitude is larger than the threshold a by function h, thus we have the following formula

$$h(s_d) = \begin{cases} 1 & s_d < a \\ s_d & s_d >= a. \end{cases} \tag{2}$$

Accordingly, we modify the previously-defined adversarial perturbation and multiplied it by the function $h(s_d)$ to decrease the gradient for the limitation on the perturbation amplitude adaptively.

$$\boldsymbol{\delta} = h(s_d) \cdot \boldsymbol{\delta'} = h(s_d) \cdot \epsilon \cdot (\boldsymbol{g_e}/||\boldsymbol{g_e}||_2). \tag{3}$$

3.4 Loss Function Elaboration

Given a sampled trajectory presented by the sampled-point set $\{t_b^{(i)}\}_{i=1}^N$, and N denotes the number of sampled points. We create a pair of trajectory-point sets (t_a, t_b), wherein t_b is the original trajectory and t_a is obtained by randomly dropping sample points from t_b with dropping rate r_1. We deem that such a non-uniform trajectory set t_a after downsampling would be in practice an uneven trajectory with a low sampling rate. In order to keep the basic path of the down sampling trajectory unchanged as a rule of thumb, the start point and end point of t_b are retained in t_a. In the training process, we normally need to maximize the joint probability of all (t_a, t_b) pairs with the encoder-decoder model to prevent downsampling trajectory set t_b from deviating the original trajectory set t_a to a large extent:

$$\text{maximize} \prod_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{P}(\mathrm{t}_{\mathrm{b}}^{(\mathrm{i})}|\mathrm{t}_{\mathrm{a}}^{(\mathrm{i})}). \tag{4}$$

The sequence encoder-decoder is trained through achieving the loss function-constrained objective optimization as mathematically defined in Equation 5. In order to make the loss function reflect the spatial proximity, we in this paper use the different weights to leash the cell as did in t2vec[14]. That is to say, while the target unit y_t is tried to decode from the decoder part, a weight is assigned to each cell. The weight of cell $u \in V$ is inversely proportional to its spatial distance from the target cell y_t. The closer the cell is to y_t, the greater the weight are assigned. Correspondingly, the loss of spatial proximity perception is defined as follows:

$$\mathcal{L} = -\sum_{t=1}^{|y|}\sum_{u\in V}\omega_{uy_t}\left(W_u^\top h_t - \sum_{v\in V}\exp(W_v^T h_t)\right), \tag{5}$$

$$\omega_{uy_t} = \frac{\exp(-||u-y_t||_2/\theta)}{\sum_{v\in V}\exp(-||v-y_t||_2/\theta)}, \tag{6}$$

where h_t is the hidden state of the decoder, $W^\top$ is the projection matrix that projects h_t into the cells, and $W_u^\top$ is the u-th row. ω_{uy_t} is the spatial proximity weight of cell u when decoding target y_t, and $||u-y_t||_2$ represents the distance between cell center coordinates. $\theta > 0$ is a spatial distance penalty parameter, which encourages the model to learn the cells near the output target cell y_t.

GAN Loss. The GAN is leveraged to conduct the amplitude of trajectory perturbation. First, the Discriminator is trained, then the Generator is followed. They are cooperative to conduct the adversarial training process with proper iteration/convergence rounds.

The overall learning process is shown in Algorithm 1:

1) Random sampling is carried out from normal distribution to generate the fake representation of the trajectories.
2) The real trajectory representation from encoder $\boldsymbol{v}$ and the representation from generator $\boldsymbol{z}$ are fed into the Discriminator which can score whether the data is true or false.
3) We train the Generator with the data recollected from the normal distribution. Note that the Generator is trained based on the Discriminator.
4) The GAN model after training is introduced into the process of adversarial training. The disturbance will be dynamically adjusted based on the output of the Discriminator during the adversarial training, as shown in Line21.
5) The above process will be repeated during training. The Discriminator can effectively judge the similarity between the real trajectory representation $\boldsymbol{v}$ and the representation generated by the generator, so as to adaptively adjust the amplitude of the adversarial disturbance. The encoder-decoder model and the GAN model will interact to reach an approximate convergence state during the training of the model. At this moment, the trajectory representation output by the encoder will tend to be stable.

Algorithm 1: "AdvTraj2Vec Algorithm"

Input: perturbation bound ϵ, ascent steps k, trajectory database $\boldsymbol{D}$

1 **for** *epoch=1…N_{ep}* **do**
2 **for** *k steps* **do**
3 Sample mini-batch of m examples $\{\boldsymbol{T^{(1)}}, ..., \boldsymbol{T^{(m)}}\}$ from $\boldsymbol{D}$;
4 Get the representation $\{\boldsymbol{v^{(1)}}, ..., \boldsymbol{v^{(m)}}\}$ encoded by the encoder $\boldsymbol{v^{(i)}} = f_\theta(\boldsymbol{T^{(i)}})$;
5 Sample mini-batch of m samples $\{\boldsymbol{z^{(1)}}, ..., \boldsymbol{z^{(m)}}\}$ from normal distribution;
6 Update the Discriminator: $\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\boldsymbol{v}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$;
7 **end**
8 Sample mini-batch of m noise samples $\{\boldsymbol{z^{(1)}}, ..., \boldsymbol{z^{(m)}}\}$ from normal distribution;
9 Update the Generator: $\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$;
10 **for** *minibatch $B \subset \boldsymbol{D}$* **do**
11 Get the decoder output y;
12 $y = f_d(f_\theta(\boldsymbol{EB}))$;
13 Update gradients to update $\boldsymbol{\delta'}$;
14 $\boldsymbol{g_e}, \boldsymbol{g} \leftarrow \boldsymbol{E}_{(Z,y)\in B}[\nabla_\theta f_d(f_\theta(\boldsymbol{EB})), y]$;
15 $\boldsymbol{\delta'} \leftarrow \epsilon \cdot (\boldsymbol{g_e}/||\boldsymbol{g_e}||_2)$;
16 Get the encoder output $\boldsymbol{v'}$;
17 $\boldsymbol{v'} = f_\theta((\boldsymbol{E} + \boldsymbol{\delta'})\boldsymbol{B})$;
18 Using the Discriminator to score;
19 $s_d \leftarrow D(\boldsymbol{v'})$;
20 Limit the amplitude of the disturbance;
21 $\boldsymbol{\delta} \leftarrow h(s_d) \cdot \epsilon \cdot (\boldsymbol{g_e}/||\boldsymbol{g_e}||_2)$;
22 Get the limited return gradient g_{adv} and update g;
23 $\boldsymbol{g_{adv}} \leftarrow \boldsymbol{E}_{(Z,y)\in B}[\nabla_\theta L(f_\theta((\boldsymbol{E} + \boldsymbol{\delta})\boldsymbol{B})), y]$;
24 $\boldsymbol{g} \leftarrow \boldsymbol{g} + \boldsymbol{g_{adv}}$;
25 Update all the parameters in the model;
26 **end**
27 **end**

4 EXPERIMENTS

Prior to mentioning the experiment detail for the validation of our proposed method, we first declare that our multi-facet experiments aim to answer the following three concerned questions:

• **EQ1**: What is the performance when traditional TSC meets the adversarial training?

• **EQ2**: How is the robustness of AdvTraj2Vec compared to the state-of-the-art TSC baselines?

• **EQ3**: What is the respective impact of each module of our proposed AdvTraj2Vec on the robustness and accuracy?

Table 1: Dataset statistics

Dataset	Points	Trips	Mean length
Porto	74269739	1233766	60
Beijing	316744	8214	80

4.1 Experimental Settings

Datasets. Our experiments are conducted on two real-world datasets captured in Porto and Beijing. The dataset in Porto[17] contains 1.7 million taxi trajectories with a period of 12 months from July 2013 to June 2014. In consideration of the usefulness of the datasets, we delete those trajectories less than 30 in length as did in t2vec, and finally fingerprint 1.2 million trajectories for our experiment. The dataset in Beijing[30], also called Geolife, contains 17621 trajectories collected from April 2007 to August 2012. We also choose those trajectories that satisfy at least 30 in length, and a less than 20-second time interval between consecutive sampling points, such operation produces 8214 trajectories. In addition, the datasets are also divided into two parts, i.e. training data and test data. For the Porto dataset, the training data consists of 800,000 trajectories, and the remaining are used for testing data. For the Geolife dataset, the first 4928 trajectories are used for training data, and the remaining are used for testing data. The statistics on the two cities' trajectory datasets are shown in Table 1.

In order to train a robust and accurate TSC method, at first we usually need down-sample the original trajectory with a dropping rate in the interval [0, 0.2, 0.4, 0.6], then perform the distortion with the probability range [0, 0.2, 0.4, 0.6] following the strategy in t2vec, and finally each trajectory $\mathbf{T_i}$ would generate 16 data pairs $(\mathbf{T_j}, \mathbf{T_i})$.

Baseline Methods. We employ six widely-used baselines to validate the performance of our approach, these baselines can be classified into two categories: i) the traditional method based on pairwise-point alignment. The representative works include EDR[7], LCSS[23], and EDwP [18]. LCSS and EDR are the two most widely used trajectory similarity measures in spatiotemporal data analysis. EDwP is currently a relatively advanced method for measuring the similarity of non-uniform and low sampling rate trajectories.; and ii) the trajectory representation-learning methods. The vanilla RNN (vRNN)[9] and T2vec[14] are the two representatives. vRNN is compared with our model as the basic method of deep learning. T2vec is the state-of-the-art method for measuring similarity among the deep learning methods. In addition, we also compared with the common set representation (CMS) method just as t2vec. CMS computes the similarity of two trajectories based on their common set after they have been mapped to cells.

Additionally, with the purpose of a microscopic observation on our proposed method, we exhibit two kinds of ablations of AdvTraj2Vec: i) the GAN-controlled perturbation amplitude module in AdvTraj2Vec is removed to test the effectiveness of harnessing the extent to which the trajectory perturbation is well-matched to promote the robustness. We denote this variant as AdvTraj2Vec-noGAN; and ii) we use the multi-iteration perturbation fashion in our model, denoted as AdvTraj2Vec-MIP, to test the effectiveness of AdvTraj2Vec. This method will execute perturbation multiple times for each round of adversarial training.

Evaluation Metrics. At present, given that the performance evaluation for TSC is still a challenging problem due to the scarcity of ground truth dataset, some works hence [14, 18, 21] suggested some other reasonable metrics to alternatively perform accuracy evaluation, i.e. leverage the most-similar search to assess the precision. Our experiment follows such performance metrics as well.

The dataset used in our experiments is generated using the following strategy. First, we select 10,000 and 100 trajectories from the Porto and Beijing test datasets respectively, denoted as $\mathbf{Q}$, and create two sub-trajectories for each trajectory by alternately extracting some points, denoted as T_a and T'_a. Then, we choose another m trajectories, denoted as $\mathbf{P}$, and process each trajectory as the above operation, denoted as T_b and T'_b. Afterwards, we implement the most-similar search queries, and each trajectory in T_a will obtain the top-k most similar trajectories from database $T'_a \bigcup T'_b$. We calculate the rank of T'_a to indicate the quality/performance of the method. In principle, the better the method behaves, the lower the calculated rank would be.

4.2 Performance Evaluation

4.2.1 Performance with Adversarial Training. To answer the **EQ1** above, we comparatively analyze the performance of AdvTraj2Vec with adversarial training and other baseline methods without - deep/adversarial training under the dataset setting that enlarges the size of $\mathbf{P}$ from 20,000 to 100,000 for Porto and from 200 to 1,000 for Beijing to better facilitate experiment evaluation. Table 2 exhibits the mean response/answering rank for 1,000 queries under Porto and Beijing datasets.

Seen from the experimental results, we can have three important observations: i) obviously, there exists a clear gap between the pairwise trajectory-point-based TSC methods (EDR, LCSS, CMS, vRNN, EDwP) and deep learning-generated trajectory representation methods (t2vec, AdvTraj2Vec). This indicates the methods in conjunction with deep learning would significantly outperform the traditional methods. We think the behind reason lies in the basic fact that the deep learning-based trajectory representation methods could learn a more appropriate embedding vector to represent one trajectory with the capability of learning its latent characteristics, compared to the traditional pairwise trajectory-points-based methods; ii) the mean rank declines to a different extent for all the methods as the size of another trajectory set $\mathbf{P}$ increases. This result is in a normal case when the massive extra (somewhat uncorrelated) trajectories are added to the test dataset. However, the different-level performance just reflects the exact robustness and accuracy of such methods even with a deterioration tendency; and iii) in general, our proposed AdvTraj2Vec has the best performance with a large margin compared to other baselines. For example, the mean rank improves 18.9% and 9.0% for Porto and Beijing compared with the best-baseline t2vec while the number of $\mathbf{P}$ is 20,000 and 200 respectively. Meanwhile, AdvTraj2Vec outperforms 28.4% and 26.4% while the size of $\mathbf{P}$ reaches up to 100,000 and 1000.

This observation further unveils that the adversarial learning with harnessed perturbation on weight parameters in the embedding layers can indeed enhance the performance to a large extent in this similar trajectory search task. We believe our method with adversarial training can be extended to other TSC-related tasks.

4.2.2 Robustness with Trajectory Distortions. To answer **EQ2**, i.e. the robustness verification, we deliberately figure out four kinds of trajectory distortions by design as shown in Table 3 in terms of sampling points addition, deletion, disturbance, and outlier injection. The trajectory transformations are controlled by three indicators: ratio, sampling rate and distance, herein the ratio represents how

Table 2: Mean rank versus the database size using Porto and Beijing datasets.

	Porto					Beijing				
Methods	20k	40k	60k	80k	100k	200	400	600	800	1000
EDR	26.89	49.12	73.32	105.2	132.09	89.5	109.9	128.7	140.6	155.9
LCSS	32.17	60.25	96.75	141.29	160.27	65.4	118.93	135.98	170.09	189.3
CMS	63.81	105.76	159.32	226.17	286.78	74.78	120.87	173.43	229.54	292.76
vRNN	31.03	59.72	97.35	131.79	159.2	63.21	115.79	134.89	165.66	178.23
EDwP	6.49	12.14	17.28	22.34	27.79	3.51	5.99	6.77	8.24	8.97
T2vec	2.28	3.47	4.68	6.17	7.39	1.22	1.33	1.48	1.62	1.74
AdvTraj2Vec	**1.85**	**2.62**	**3.42**	**4.46**	**5.29**	**1.11**	**1.19**	**1.21**	**1.24**	**1.28**

Table 3: Trajectory transformations.

Transformation type	Transformation
Point shift	Add sampling points
	Delete sampling points
	Disturb sampling points
Noise	Inject outlier

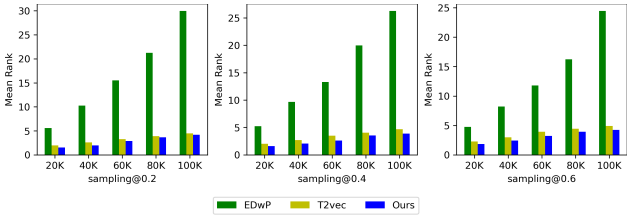

Figure 3: Mean rank calculation in the case of addition of trajectory points with different ratios.

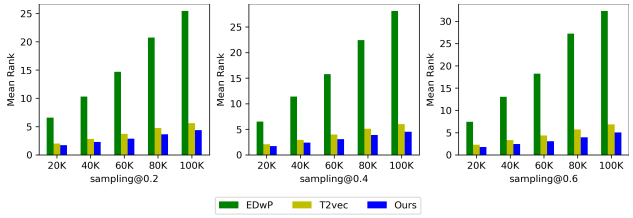

Figure 4: Mean rank calculation in the case of deletion of trajectory points with different ratios.

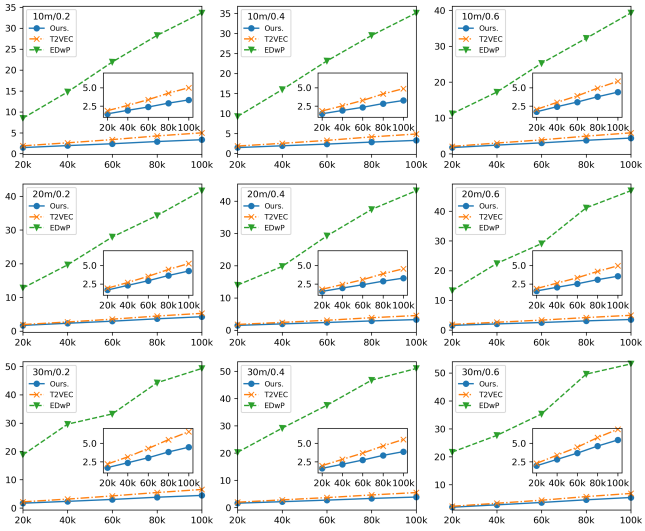

Figure 5: Mean rank calculation in the case of distance disturbance with different ratios.

many points are transformed in a trajectory; the sampling rate determines how many trajectories are transformed and the distance specifies how far one trajectory point can be deviated from the original location.

Robustness with Trajectory Point Addition. We verify the robustness of AdvTraj2Vec by deliberately adding some points to the trajectory. Specifically we can add a point p^s between two consecutive points p_i and p_{i+1} and set its coordinate as the mean value of the coordinate p_i and coordinate p_{i+1}. Moreover, the number of points needed to add is determined by the indicator ratio. In our experiment, we set the ratios as 50% and the sampling rate as 20%, 40% and 60% respectively. Figure 3 shows the mean rank under the three sampling rates. From the experimental results, we can see that EDwP performs the worst with a large margin, stemming from the fact its performance merely depends on the quality of the dataset, i.e. the matching caliber of sampling points over two trajectories. For the deep learning methods, AdvTraj2Vec has better performance than t2vec, and the robustness of our proposed AdvTraj2Vec becomes more apparent as the amount of test data increases. For example, when the size is 100k, the performance of our AdvTraj2Vec is improved by 6.7%, 17.0%, 13.8% respectively, corresponding to the three sampling rates 20%, 40% and 60%.

Robustness with Trajectory Point Deletion. Inversely, we in this case remove some points in each trajectory using the ratios 20%, 40% and 60%. Figure 4 shows the mean rank, from which we can see that our AdvTraj2Vec still has the best performance as the ratio of point deletion enlarges. For instance, the performance of our model improved 23.4%, 26.4%, 29.8%, 31.2%, 26.8% when the ratio is 0.6 under dataset sizes 20k, 40k, 60k, 80k and 100k, compared to the best baseline method t2vec.

Robustness with Trajectory Disturbance. Keeping the three ratios unchanged, i.e. 20%, 40% and 60%, we perturb the points in each trajectory through deviate the distance 10m, 20m, 30m from the original point location respectively. We obtain the experiment

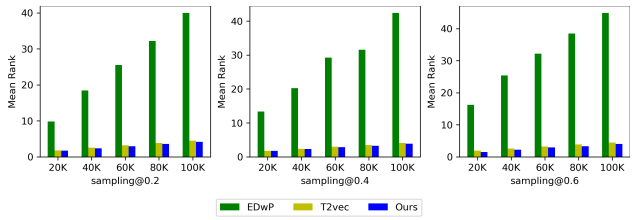

Figure 6: Mean rank calculation under outlier injection with different ratios.

result as shown in Figure 5. From the figure, we can see that our AdvTraj2Vec performs significantly better than t2vec. In particular, as the size of the test dataset increases, the gap becomes larger and larger. This is because as the number of query trajectories increases, the TSC between different trajectories would be strictly affected by the fluctuation/deviation points. That is to say, t2vec would become more sensitive to the perturbation of trajectory samples with the routine deep learning fashion, leading to poor performance on such disturbed trajectory datasets. Nevertheless, our proposed AdvTraj2Vec, proving that adversarial training is introduced to enhance the robustness, has optimized the model (neural network) parameters during the training process, thus it has better robustness.

Robustness with Outlier Injection. Remaining the sampling rate same as the previous setting, i.e. 20%, 40% and 60%, the point ratio is set to 25%. We inject outliers with the distance 75m to the original trajectory w.r.t. our statistics that the average distance between trajectory points is 75m as well. The experimental results are exhibited in Figure 6, from which we can see that, similar to other cases, our model achieves better performance regardless of the data size. This is because the outlier injection may bring in incorrect inference while disregarding such noise samples during the course of deep learning in a regular way, especially trained on a clean dataset. Differently, our AdvTraj2Vec alternatively handle this noise-data case through perturbing the weight parameters in the embedding layers, thereby, our method can better adapt to the noise data-injection scenario.

To sum up, the four groups of experimental results verify the adversarial training idea can validly enhance the robustness of our proposed AdvTraj2Vec in terms of trajectory points addition, deletion, disturbance, or even uncorrelated outlier injection.

4.2.3 Ablation Study. To answer **EQ3**, the functionalities of two radical components are investigated: i) when there does not exist the GAN module to control perturbation amplitude; ii) when the multi-iteration perturbations can be applicable into the course of perturbation. We execute the ablation experiments and present the results in Table 4, from which we can observe that AdvTraj2Vec-noGAN behaves worse than AdvTraj2Vec, e.g. the performance drops by 11% when the dataset size is 100K. This implies the perturbation should not be in the wild, but rather in the control. That is to say, the harness functionality of GAN plays an important role to guide the perturbation in a proper pitch. During the model training course, the output of the model ought to be recognized the same as the original trajectory after adversarial perturbation on

Table 4: The ablation experimental results on Porto dataset.

	20K	40K	60K	80K	100K
AdvTraj2Vec-noGAN	1.979	2.893	3.865	4.942	5.832
AdvTraj2Vec-MIP	2.214	3.317	4.532	6.022	7.224
AdvTraj2Vec	**1.85**	**2.62**	**3.42**	**4.46**	**5.29**

Table 5: Time cost for model training on Porto dataset.

	AdvTraj2Vec	T2vec
Time	8.10h	6.22h

the trajectory. After the multi-iteration perturbations, we can see the experimental results of AdvTraj2Vec-MIP are not good compared to AdvTraj2Vec, even worse than AdvTraj2Vec-noGAN, e.g., the accuracy in AdvTraj2Vec-MIP are reduced by 11.8%, 14.6%, 17.2%, 21.8% and 23.9% compared to that in AdvTraj2Vec-noGAN in terms of 20k, 40k, 60k, 80k and 100k. This point shows that multi-iteration perturbation cannot further accelerate the effect of adversarial training, which reversely verifies that the perturbation should be constrained appropriately for each trajectory.

4.2.4 Scalability. We compare the training time of T2vec and AdvTraj2Vec. The training data for the experiment is 800k trajectories. It can be seen from the table 5 that the training time of AdvTraj2Vec is slightly longer than that of T2vec. The reason is that AdvTraj2Vec needs to continuously update the model based on adversarial training and the discriminator to generate better parameters.

4.3 Parameter Sensitivity Study

4.3.1 Selection of Threshold a. The threshold a specifies the perturbation amount. In this subsection, we would investigate the effect of the value of a using the dataset Porto. As defined in the formula (2), a larger threshold means a bigger perturbation. In our experiment, the size of query dataset $|T'_a \bigcup T'_b|$ is fixed to 100,000 in the testing phase, and the mean rank results are drawn in Figure 7.

From the curve, we can see the performance is poor when the threshold a is small, that is to say, the allowed perturbation on the embedding parameters is too small to make the adversarial training play an enhanced effect. As the threshold enlarges gradually, the performance becomes better and better, and arrives at the lowest point (best performance) at a=0.7. However, when the threshold

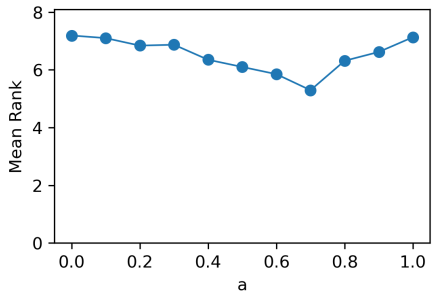

Figure 7: The impact of the threshold a on the model.

Table 6: The impact of the hidden layer size of GAN.

	20K	40K	60K	80K	100K
3	1.94	2.72	3.73	4.80	5.49
9	1.92	2.73	3.53	4.80	5.39
12	**1.85**	**2.62**	**3.42**	**4.46**	**5.29**
15	1.85	2.73	3.43	4.47	5.30
18	1.94	2.98	3.75	4.47	5.53

increases consistently, the performance begins to deteriorate, which indicates the allowed perturbation might be too large. The lowest point (a.k.a. the smallest mean rank) represents a reasonable trajectory perturbation to the original trajectory for the adversarial training. Therefore we set the threshold equal to 0.7 as the condition to decide how much perturbation should be executed.

4.3.2 The Influence of the number of Hidden Layers of GAN. We further investigate the effect of the size of hidden layers under the task of trajectory search with the threshold a= 0.7. The experimental results are shown in Table 6. We find that when the size of hidden layers increases from 3, 9, to 12, the performance gradually improves, this tendency reflects that the larger size of hidden layers could enhance the recognition on trajectory representation. Nevertheless, the performance has a slight deterioration while consistently increasing the size of hidden layers, that is to say, there may exist a somewhat overfitting phenomenon if too many hidden layers are employed. Thereby, we set the size of the hidden layer in GAN primitive as 12 to enable it equipped with a good recognition capability on trajectory representation.

5 RELATED WORK

5.1 Adversarial learning

Adversarial training, aiming to combat against adversarial attacks, is mainly utilized to improve the robustness of model inference in CV through co-training original clean examples and adversarial examples generated by the pixel perturbation or retraining the adversarial examples merely. We opine that the essence of promoting robustness is due to the data augmentation, namely, the adversarial examples enforce the deep learning models to learn attack-resilient parameters to withstand various adversarial attacks, in this way, the model would enhance the performance robustness.

Our work alternatively attempts to perturb the weight parameters of the embedding layer instead of trajectory points. Furthermore, in order to harness the magnitude of weight modification, we also employ the GAN-spawned distortion to guide the concrete modification, and keep the perturbation neither too large nor too small.

5.2 Trajectory Similarity Computation

The problem of TSC has been extensively calibrated in terms of spatiotemporal-distance measures and trajectory-sequence distance measures. The former regards the trajectory's both spatial and temporal attributes, while the latter only takes into account the spatial information. This paper focuses on the sequence measures.

For trajectory-sequence distance measure, this line of research can be further divided into two subcategories: i) the pairwise trajectory-point distance, and ii) deep learning-generated vector distance. In the first subcategory, DTW[28] was the first work to treat with the local time shift issue, and it searched for the smallest aligned-points distance over two trajectories. Moreover, the longest common subsequence (LCSS) [23] measure and the edit-distance-based measure [1, 5, 6, 10] only employ partial-match metrics for TSC. However, the above matching strategy only selects the optimally-matched data points in the trajectory. It ignores the trajectory-noise existence caused by whatever unexpected determinant [20, 24], that is to say, these methods did not take the robustness into account. In order to solve this robustness problem, Edit distance with projections (EDwP)[4, 11, 16, 18] were proposed, such that EDwP used dynamic interpolation to match trajectory points. The difference lies in that our work strengthens the robustness by adversarial learning on the representation of the trajectory, rather than modifying the matching strategy of trajectory points.

The second subcategory is to utilize deep learning to infer the trajectory similarity through computing the distance of the learned embeddings[14, 26, 27]. T2vec[14] was the representative attempt to embed trajectories into vectors for similarity inference using unsupervised methods. It enhanced robustness by modifying or generating data. On the one hand, the number of trajectories generated is limited. On the other hand, the modification strategies are based on the independent perturbation of a single point without considering the contextual relationship of the trajectory. So the modification may not fit to advance the robustness. Furthermore, T3S[26] and NEUTRAJ[27] are supervised methods to learn trajectory representations. Nevertheless, these DL-based TSC methods did not encompass the robustness issue, which implies that even a slight change in trajectory or a different trajectory scenario can deduce a big deterioration in accuracy.

6 CONCLUSION

To the best of our knowledge, we introduce the adversarial training idea into the trajectory similarity computation domain for the first time, and the extensive experiments also validate our work with the comparison to other baselines. There are mainly two novelties in our proposed AdvTraj2Vec. First, unlike the regular operation, it does not perturb the original trajectory data in the course of adversarial training, but rather disturbs the weight parameters of the embedding layer to enhance the robustness and accuracy. Second, as aforementioned, the perturbation amplitude plays a critical role during the adversarial training, thus, we employ the GAN primitive to adjust the extent to which an appropriate perturbation would be spawned. Upon two realistic trajectory datasets Porto and Beijing, multi-facet experiments verify that our proposed AdvTraj2Vec can achieve superior performance on the robustness and accuracy than the state-of-the-art TSC methods.

Our work sheds light on trajectory data analytics by introducing adversarial training, however, we think there may exist several interesting questions worthwhile to be explored in the future. Firstly, we can extend our work into a specific measurement, for example, NEUTRAJ. Secondly, we can apply our AdvTraj2Vec to other tasks, such as anomaly detection and clustering.

REFERENCES

[1] Osman Abul, Francesco Bonchi, and Mirco Nanni. 2008. Never walk alone: Uncertainty for anonymity in moving objects databases. In *2008 IEEE 24th international conference on data engineering*. Ieee, 376–385.

[2] Helmut Alt and Michael Godau. 1995. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications* 5, 01n02 (1995), 75–91.

[3] Stefan Atev, Grant Miller, and Nikolaos P Papanikolopoulos. 2010. Clustering of vehicle trajectories. *IEEE transactions on intelligent transportation systems* 11, 3 (2010), 647–657.

[4] Wei Cao, Zhengwei Wu, Dong Wang, Jian Li, and Haishan Wu. 2016. Automatic user identification method across heterogeneous mobility data sources. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 978–989.

[5] Jae-Woo Chang, Rabindra Bista, Young-Chang Kim, and Yong-Ki Kim. 2007. Spatio-temporal similarity measure algorithm for moving objects on spatial networks. In *International Conference on Computational Science and Its Applications*. Springer, 1165–1178.

[6] Lei Chen and Raymond Ng. 2004. On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. 792–803.

[7] Lei Chen, M Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 491–502.

[8] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. 2011. Discovering popular routes from trajectories. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 900–911.

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[10] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1542–1552.

[11] Zhenni Feng and Yanmin Zhu. 2016. A survey on trajectory data mining: Techniques and applications. *IEEE Access* 4 (2016), 2056–2067.

[12] Ralf Hartmut Güting and Markus Schneider. 1995. Realm-based spatial data types: The ROSE algebra. *The VLDB Journal* 4, 2 (1995), 243–286.

[13] RH Güting and M. Schneider. 1997. *Realm-based spatial data types: The ROSE algebra*. Spatial Data Types for Database Systems.

[14] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S Jensen, and Wei Wei. 2018. Deep representation learning for trajectory similarity computation. In *2018 IEEE 34th international conference on data engineering (ICDE)*. IEEE, 617–628.

[15] Zhenhui Li, Jiawei Han, Ming Ji, Lu-An Tang, Yintao Yu, Bolin Ding, Jae-Gil Lee, and Roland Kays. 2011. Movemine: Mining moving object data for discovery of animal movement patterns. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 4 (2011), 1–32.

[16] Huiping Liu, Cheqing Jin, and Aoying Zhou. 2016. Popular route planning with travel cost estimation. In *International Conference on Database Systems for Advanced Applications*. Springer, 403–418.

[17] Luís Moreira-Matias, João Gama, Michel Ferreira, João Mendes-Moreira, and Luis Damas. 2016. Time-evolving OD matrix estimation using high-speed GPS data streams. *Expert systems with Applications* 44 (2016), 275–288.

[18] Sayan Ranu, Padmanabhan Deepak, Aditya D Telang, Prasad Deshpande, and Sriram Raghavan. 2015. Indexing and matching trajectories under inconsistent sampling rates. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 999–1010.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin. 2018. Semantically Equivalent Adversarial Rules for Debugging NLP models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

[20] Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. 2020. A survey of trajectory distance measures and performance evaluation. *The VLDB Journal* 29, 1 (2020), 3–32.

[21] Han Su, Kai Zheng, Haozhou Wang, Jiamin Huang, and Xiaofang Zhou. 2013. Calibrating trajectory data for similarity-based analysis. In *Proceedings of the 2013 ACM SIGMOD international conference on management of data*. 833–844.

[22] David Alexander Tedjopurnomo, Xiucheng Li, Zhifeng Bao, Gao Cong, Farhana Choudhury, and AK Qin. 2021. Similar Trajectory Search with Spatio-Temporal Deep Representation Learning. *ACM Transactions on Intelligent Systems and Technology (TIST)* 12, 6 (2021), 1–26.

[23] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. 2002. Discovering similar multidimensional trajectories. In *Proceedings 18th international conference on data engineering*. IEEE, 673–684.

[24] Haozhou Wang, Han Su, Kai Zheng, Shazia Sadiq, and Xiaofang Zhou. 2013. An effectiveness study on trajectory similarity measures. In *Proceedings of the Twenty-Fourth Australasian Database Conference-Volume 137*. 13–22.

[25] Zheng Wang, Cheng Long, Gao Cong, and Ce Ju. 2019. Effective and efficient sports play retrieval with deep representation learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 499–509.

[26] Peilun Yang, Hanchen Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2021. T3S: Effective Representation Learning for Trajectory Similarity Computation. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2183–2188.

[27] Di Yao, Gao Cong, Chao Zhang, and Jingping Bi. 2019. Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach. In *2019 IEEE 35th international conference on data engineering (ICDE)*. IEEE, 1358–1369.

[28] Byoung-Kee Yi, Hosagrahar Visvesvaraya Jagadish, and Christos Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. In *Proceedings 14th International Conference on Data Engineering*. IEEE, 201–208.

[29] Kai Zheng, Yu Zheng, Xing Xie, and Xiaofang Zhou. 2012. Reducing uncertainty of low-sampling-rate trajectories. In *2012 IEEE 28th international conference on data engineering*. IEEE, 1144–1155.

[30] Yu Zheng, Xing Xie, Wei-Ying Ma, et al. 2010. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.