Efficient Trajectory Similarity Computation with Contrastive Learning

Liwei Deng
University of Electronic Science and Technology of China
deng_liwei@std.uestc.edu.cn

Yan Zhao
Aalborg University
yanz@cs.aau.dk

Zidan Fu
University of Electronic Science and Technology of China
fuzidan@std.uestc.edu.cn

Hao Sun
Peking University
sunhao@stu.pku.edu.cn

Shuncheng Liu
University of Electronic Science and Technology of China
liushuncheng@std.uestc.edu.cn

Kai Zheng*
University of Electronic Science and Technology of China
zhengkai@uestc.edu.cn

ABSTRACT

The ubiquity of mobile devices and the accompanying deployment of sensing technologies have resulted in a massive amount of trajectory data. One important fundamental task is trajectory similarity computation, which is to determine how similar two trajectories are. To enable effective and efficient trajectory similarity computation, we propose a novel robust model, namely Contrastive Learning based Trajectory Similarity Computation (CL-TSim). Specifically, we employ a contrastive learning mechanism to learn the latent representations of trajectories and then calculate the dissimilarity between trajectories based on these representations. Compared with sequential auto-encoders that are the mainstream deep learning architectures for trajectory similarity computation, CL-TSim does not require a decoder and step-by-step reconstruction, thus improving the training efficiency significantly. Moreover, considering the non-uniform sampling rate and noisy points in trajectories, we adopt two type of augmentations, i.e., point dowm-sampling and point distorting, to enhance the robustness of the proposed model. Extensive experiments are conducted on two widely-used real-world datasets, i.e., Porto and ChengDu, which demonstrate the superior effectiveness and efficiency of the proposed model.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Trajectory Similarity Computation; Contrastive Learning; Efficiency

ACM Reference Format:
Liwei Deng, Yan Zhao, Zidan Fu, Hao Sun, Shuncheng Liu, and Kai Zheng. 2022. Efficient Trajectory Similarity Computation with Contrastive Learning.

*Corresponding author: Kai Zheng.

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

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557308

In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557308

1 INTRODUCTION

The prevalence of GPS-enabled devices and wireless communication technologies have generated massive trajectory data, which is denoted as a sequence of discrete locations describing the underlying route of a moving object over time [19]. Over the last decades, a rich variety of trajectory analytic tasks have been proposed, such as similar subtrajectory search [28], trajectory prediction [20] and trajectory clustering [1], among which trajectory similarity computation is one of the most fundamental component [11, 24, 37–40, 42]. Existing studies on trajectory similarity computation can be mainly divided into two groups, i.e., traditional methods and learning-based methods. The former methods, such as DTW [33], EDR [3], EDwP [23], etc, are mainly based on matching-and-measurement paradigm, i.e., finding the best alignment using dynamic programming and counting the distance between two aligned points, which cause that most of them have quadratic computational complexity $O(n^2)$ [19], where n is the number of samples in trajectories. Besides, these methods are often criticized for their lack of robustness, e.g., they suffer from the poor trajectory quality caused by non-uniform sampling rate [41] or noisy points [15].

The existing learning-based methods aim to reduce the computational complexity and/or improve the robustness. We divide them into two directions based on their purposes. The first direction, called neural approximation method, leverages powerful neural networks to approximate any existing trajectory measurement in hidden space [29, 31, 35]. Specifically, they train a neural network g to encode trajectories into hidden space while minimize the difference between the estimated similarity and the ground-truth, i.e., $|D_h(g(T_i), g(T_j)) - D(T_i, T_j)|$, where T_i and T_j are two trajectories, D_h is the dissimilarity (similarity) measurement (e.g., Euclidean distance) in hidden space, D is the trajectory dissimilarity (similarity) measurement (e.g., DTW) to be approximated. These methods does not require point alignment between two trajectories and thus are linear in the length of trajectories [31]. Although these methods accelerate the process of trajectory similarity computation, they may also suffer from robustness problems as the measurements to be approximated.

To solve the robustness problems, the methods in the second direction propose some brand new measurements through neural networks, which are robust for processing low-quality trajectory data while reduce the computational complexity [19, 30, 32]. The idea of these methods to reduce the computational complexity is similar to that of the approximation methods, i.e., they learn a neural network to convert trajectories into latent representations and then define similarity in the latent space. Unlike neural approximating methods, these methods mainly adopt a sequential auto-encoder architecture to unsupervisedly learn the mapping function due to lack of the guidance from an existing measurement. Moreover, to improve the robustness of the latent representation w.r.t poor quality, these methods adopt different strategies. For example, t2vec [19] leverages denoising sequential autoencoder and Trembr [30] incorporates information of road network and designs multiple tasks.

In this work, we will go further in the second direction and design a new measurement. Although previous methods can compute trajectory similarity efficiently, they suffer from low efficiency in training the encoding model. This is due to the inherent limitations of sequential auto-encoder architecture, where the decoding procedure and the step-by-step reconstruction are extremely time-consuming. For example, t2vec takes about 14 hours to train an epoch on 20 million trajectories with an average of 60 samples using Tesla K40 GPU [19]. Obviously, the training efficiency of t2vec is hardly acceptable in practice. Moreover, these methods try to learn the consistent representations for the trajectories from the same underlying route to solve the quality problem. In other words, for the trajectories even with different sampling rates and noisy points from the same underlying route, the representations should be the same. However, we argue that this goal cannot be achieved by sequential auto-encoders because their objective is to reconstruct the trajectories instead of the underlying route. Taking t2vec as an example, suppose there are two different trajectories, i.e., T_i and T_j, sampled from the same underlying route, in which $\boldsymbol{t}_i$ and $\boldsymbol{t}_j$ are the corresponding output representations of the encoder by inputting the destructive trajectories $T_i^{'}$ and $T_j^{'}$. To reconstruct trajectories T_i and T_j, $\boldsymbol{t}_i$ and $\boldsymbol{t}_j$ should be different even T_i and T_j are derived from the same underlying route. Therefore, obtaining the consistent representations is very difficult (if possible) for sequential auto-encoder.

To solve the problems above, we propose a novel robust model based on Contrastive Learning for Trajectory Similarity computation (CL-TSim). We follow the common paradigm that firstly learns the representations of trajectories and then uses Euclidean distance to calculate the similarity between trajectories in the encoding space. To learn the consistent representations of trajectories, an intuitive thought is directly minimizing the distance between $\boldsymbol{t}_i$ and $\boldsymbol{t}_j$. However, this operation cannot be implemented due to the agnostic of which trajectories are from the same underlying route. To circumvent this problem, we observe that two trajectories T_i and T_j should be two views of the same underlying route, where T_j is sampled from T_i. Based on this observation, we first preprocess the trajectory T_i to get the augmented trajectory T_j, in which down-sampling and distorting augmentations are used to fit the trajectory characteristics including the non-uniform sampling rate and noisy points. Then we simultaneously encode them into hidden space and maximize agreement between them. To avoiding constant solution, i.e., all the representations are the same, the pair-based negative sampling strategy [5] is adopted. Moreover, following the contrastive learning architecture, CL-TSim only contains an encoder and a projector, where the encoder is to encode the original trajectories to learn their representations, and the projector is to map the representations into a metric space where the loss function works. Compared to sequential auto-encoders, it does not require a decoder and step-by-step reconstruction, which can significantly reduce the training time.

The major contributions of this paper are as follows:

- We propose a robust and efficient model, CL-TSim, based on contrastive learning for trajectory similarity computation. To the best of our knowledge, it is the first study to apply contrastive learning framework to the task of trajectory similarity computation.
- We learn the consistent representations of trajectories by applying trajectory data augmentations, i.e., point down-sampling and point distorting, under the framework of contrastive learning, in which point down-sampling and distorting are designed to model the non-uniform sampling rate and noisy points of trajectories.
- We conduct extensive experiments on two widely-used real-world trajectory datasets. The results demonstrate the superiority of the proposed model in terms of effectiveness and efficiency. Specifically, compared to the method, i.e., t2vec, CL-TSim is around 20x faster in training time (on one epoch) and performs better in accuracy, i.e., lower mean rank and higher precision.
- We conduct extensive ablation study to investigate the impact of each component of the proposed model. We also release the codes in Github[1] for the purpose of reproducibility.

2 RELATED WORKS

2.1 Trajectory Similarity Computation

Trajectory similarity computation, the key component for many applications, has been studied in decades. Before the prevalence of deep learning, all methods fall into the matching-and-measurement paradigm, such as DTW [33], LCSS [26], etc, in which point matching between two trajectories is first adopted while a distance function is leveraged to measure the dissimilarity between points. In these methods, the distance between two trajectories is accumulated from distance between points. Due to the process of point matching, these methods result in $O(n^2)$ computational complexity, which cannot be endured in practice. For example in our experiments, given 1K query trajectories, vanilla DTW [33] costs around 10 hours to find the most similar trajectory from 100K trajectories[2]. Different from the previous methods, t2vec [19] is a deep-learning-based technique discarding the matching-and-measurement paradigm, which largely reduces the inference time of trajectory similarity computation while performs better on self-similarity and cross-similarity tasks. However, it takes a significant amount of time in training and is not robust enough because of the limitation of the sequential auto-encoder architecture. To solve these problems,

[1] https://github.com/LIWEIDENG0830/CL-TSim

[2] Due to the limitation of space, we refer the readers to the survey [24] for comprehensive study of trajectory similarity computation.

we adopt the contrastive learning architecture with two types of trajectory data augmentations in this work.

2.2 Contrastive Learning

Contrastive learning is a prevalent unsupervised learning architecture in computer vision. The main idea of this architecture is maximizing the similarity between two augmentations of one image while keeping dissimilarity between different images. Therefore, the main components in contrastive learning are data augmentations and dissimilar operations. In terms of images augmentations, amounts of intuitive augmentations, such as cropping and resizing [5], color distortion [17], rotation [13], cutout [12], etc, can be employed to improve the accuracy of image classification. For the dissimilar operations, we should note that this operation is important because only maximizing agreement between representations of two augmentations will cause a constant solution (also called collapsing solution), i.e., all the representations of images are same, which will make the model useless for downstream applications. To avoid collapsing solution, many models such as SimCLR [5], Moco [7], and Simsiam [8] have been proposed, in which SimCLR adopts negative sample pairs, Moco levarages momentum encoder, and Simsiam adopts an advanced stop-grad operator. Inspired by the success of contrastive learning in image similarity calculation, we apply it in trajectory similarity computation in this work. Specifically, two trajectory augmentations are considered to deal with the poor quality problems of trajectories. The negative sample pairs [5] are adopted for preventing the collapsing solution due to its simplicity and effectiveness.

3 PRELIMINARIES

We proceed to present necessary preliminaries and then define the problem to be addressed.

Definition 1 (Underlying Route). *Underlying route is a continuous spatial curve (in the longitude and latitude domain) generated by a moving object.*

Definition 2 (Trajectory). *A trajectory of a moving object, denoted as T, is a finite sequence of points sampled from the underlining route with the form of* $T = ((x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n))$, *where* (x_i, y_i) *stands for the longitude and latitude information of a sampled location at time stamp* t_i[3].

It is worthy noting that the underlying route only exists in theory because GPS-enabled devices cannot record the spatio-temporal positions continuously [19]. The actual data one can acquire and analyze are trajectories. Affected by sampling methods and devices, trajectories are often generated based on different sampling rates and contain noisy points.

Problem Definition. Given a set of trajectories, our problem is to design an efficient and robust model to compute the similarity between trajectories with the following goals:

1) Efficient representation learning: learn a representation $\boldsymbol{t}$ for each trajectory T efficiently, where $\boldsymbol{t}$ can reflect the underlying route of trajectory T for computing trajectory similarity; and

2) Model robustness: the trajectory similarity computation is robust to the data quality variance in trajectories. In other words, the similarity between two arbitrary trajectories, T_i and T_j, is consistent regardless of non-uniform sampling rates and noisy points, i.e., $d(T_i, T_j) \approx d(T_i^{'}, T_j^{'})$, where $d(T_i, T_j)$ is a trajectory similarity (distance) measurement between T_i and T_j determining how similar (dissimilar) two trajectories are, and $T_i^{'}$ ($T_j^{'}$) is the trajectory sampled from the same underlying route with T_i (T_j).

[3]In this paper, we only consider the spatial information.

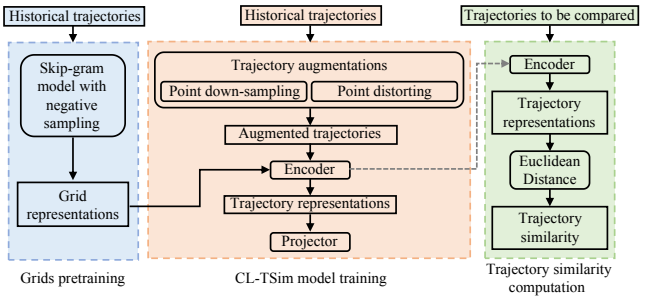


Figure 1: Framework Overview

4 FRAMEWORK AND METHODOLOGY

Bearing the above design goals, we propose a contrastive learning based model, namely CL-TSim, for trajectory similarity computation. We first give an overview of the framework and then provide specifics on each component in the framework.

4.1 Framework Overview

The framework consists of three components, i.e., pretraining, CL-TSim model training, and trajectory similarity computation, as shown in Figure 1.

The grids pretraining component uses the Skip-gram model [21, 22] with negative sampling to pretrain the representations of grids, which are leveraged by the encoder component to convert a trajectory into a sequence of vectors.

The CL-TSim model training component takes historical trajectories as input, which includes trajectory augmentations, an encoder, and a projector. More specifically, in trajectory augmentations, two trajectory sampling methods (i.e., point down-sampling and point distorting) are adopted to generate augmented trajectories, which aim to model non-uniform sampling rates and noisy points in trajectories. The encoder is used to learn the representations of trajectories by taking sequential information into consideration. Finally, in the projector, the representations are mapped into a metric space where a loss function works that is to maximize the similarity between two views of one underlying route while keeping dissimilarity between different underlying routes to train the model parameters.

After the model training, the trained encoder is transferred to the trajectory similarity encoder in the trajectory similarity computation component. When two new trajectories to be compared arrives, they are first encoded to a representation by the encoder, and then their distance is computed by Euclidean distance. The smaller the distance is, the more likely that the two trajectories are similar.

4.2 Grids Pretraining

Similar to the prior learning-based methods for trajectory similarity computation [14, 19, 30, 34, 36], we partition the whole space into non-overlapping and adjacent grids with equal size and treat each grid as a token. All the latitude-longitude points in the same grid will be mapped to the same token. Li et. al [19] empirically demonstrate the usefulness of this operation to overcome the problems of non-uniform and low sampling rates. After that, each trajectory in the training set is converted into a sequence of grids.

For better capturing the relationship among grids, we adopt the Skip-gram model [21, 22] with negative sampling, which has been proven to be an effective and efficient methods in many literatures [4], to pretrain the representations of grids, i.e., $\boldsymbol{E} \in \mathbb{R}^{N \times d}$, where N is the number of grids. Specifically, given a sequence of grids, e.g., $(g_0, g_1, g_2, \cdots, g_n)$, the objective of the Skip-gram with negative sampling is to maximize the average log probability as follows.

$$\frac{1}{N_1}\sum_{i=1}^{N_1}\sum_{-w\le j\le w, j\neq 0} log\ (\boldsymbol{g}_j^T \boldsymbol{g}_c) + \sum_{k=1}^{K} E_{g_k\sim P_{neg}(g)}[log\ \sigma(-\boldsymbol{g}_{g_k}^{'\ T}\boldsymbol{g}_c)] \tag{1}$$

where $\boldsymbol{g_j}$ (i.e., $\boldsymbol{E}[j,:]$) is the representation of g_j, w is the window size defining the positive pairs, and g_k is the negative sample of g_c from distribution $P_{neg}(g)$. After pretraining, we freeze the representations of grids, which means that these representations are not updated when other parameters are trained. This is mainly because that the proposed loss function (cf. Equation 6) cannot explicitly reflect the spatial information. As the training goes on, the spatial information may be discarded to some extent. We empirically demonstrate this statement in the experimental part (cf. Sections 5.6 and 5.9).

4.3 CL-TSim Model Training

Trajectory augmentations. Although there are many intuitive augmentations in images, such as cropping and resizing [5], color distortion [17], rotation [13], cutout [12], etc., it is non-trivial to define the augmentations for trajectory data (that is totally different from images), which can reflect the characteristics of trajectories to some extent. Following prior work [19], we adopt two type of trajectory augmentations, i.e., point down-sampling and point distorting, in which point down-sampling is to model the non-uniform sampling rate of trajectories and point distorting is to fit the characteristic of noisy points in trajectories.

Before formally introducing the point down-sampling augmentation, we firstly analyze the non-uniform sampling rate problem. This problem typically exists in two situations. The first situation is that two moving objects are equipped with GPS devices that have different sampling rates. The second one is that trajectories are generated by a moving object with different sampling rates, which may be caused by missing samples or equipment errors. Considering these two situations, we design two point down-sampling augmentations. Specifically, suppose $T = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$ is an existing trajectory in a dataset. The first point down-sampling augmentation, called *uniformly down-sampling augmentation*, can be described as follows.

$$T^{u_1} = ((x_1, y_1), (x_{k+1}, y_{k+1}), (x_{2k+1}, y_{2k+1}) \cdots) \tag{2}$$

where T^{u_1} is the augmented trajectory from T, and k is a hyper-parameter that define the re-sampling interval. In this augmentation, we aim to simulate the trajectories that have different sampling rates as described in the first situation. For the second point down-sampling augmentation, called *randomly down-sampling augmentation*, we obtain the augmented trajectory T^{u_2} by randomly dropping a portion of points in T, where the dropping rate is denoted as r_d that is the ratio between the number of dropped points and that of all the points in T. This augmentation is used to simulate the second situation.

Moreover, noisy points commonly exist in all trajectories due to the impact of sampling devices. We hypothesize that these noises are random and subject to normal distribution. To make latent representations be robust to these noisy points, an augmentation named *point distorting augmentation* is adopted [19]. Suppose (x_i, y_i) is the point to be distorted. We first convert this point into Mercator coordinate, e.g., $(\hat{x}_i, \hat{y}_i)$, in which the distance between points is measured in meters, so that we can easily set a satisfying perturbation on the points. Then this augmentation can be described as follows.

$$\begin{aligned} \hat{x}_i &= \hat{x}_i + r_s \cdot d_{x_i}, \quad d_{x_i} \sim N(0,1) \\ \hat{y}_i &= \hat{y}_i + r_s \cdot d_{y_i}, \quad d_{y_i} \sim N(0,1) \end{aligned} \tag{3}$$

where $r_s \cdot d_{x_i}$ ($r_s \cdot d_{y_i}$) denotes the random distorting for $\hat{x}_i$ ($\hat{y}_i$), and d_{x_i} (d_{y_i}) is a random number generated from the standard normal distribution $N(0,1)$. Next, r_s is a small radius that can be set by observations from a real application or specified by users, which is empirically set to 50 (meters) in our experiments.

For different types of sampling methods such as uniform down-sampling augmentation and point distorting augmentation, we combine them on a trajectory to get the augmentation. For the same type of sampling methods such as uniform and randomly down-sampling augmentation, we do not simultaneously adopt them. Then the augmented trajectory and the original trajectory can be treated as two views for the underlying route described by the original trajectory.

Encoder. After obtaining the two trajectories from the same underlying route, we firstly map the points in trajectories into grids. Therefore, each trajectory after augmentations is converted into a sequence of grids. Then we encode the them into Euclidean space by an encoder f. Considering the training efficiency, we choose Long Short-Term Memory (LSTM) [16] instead of complex neural networks as the basic network structure for the encoder. Suppose there are two trajectories, e.g., the original trajectory T and its corresponding augmented trajectory T^a. The encoding procedure in a python style can be presented as follows.

$$\begin{aligned} \boldsymbol{t}_e &= Emb(T, \boldsymbol{E}) \\ \boldsymbol{t}_f &= LSTM(\boldsymbol{t}_e) \end{aligned} \tag{4}$$

where $\boldsymbol{E}$ is the pretrained grid representations, Emb is an operation that converts the grids into vectors, and $\boldsymbol{t}_f$ is the latent representation of trajectory T. Similarly, we can obtain the representation $\boldsymbol{t}_f^a$ for trajectory T^a.

Projector. Previous studies [5, 7] show that directly maximizing agreement between representations of two views obtained by the encoder may degrade the performance. Following these studies, we add a projector after encoder, as shown in Figure 1. In practice, two

linear layers with ReLU are adopted. Supposing that $\boldsymbol{t}_f$ is the input, the projection can be expressed as follows.

$$\begin{aligned} \boldsymbol{t}_m &= ReLU(\boldsymbol{W}_m \boldsymbol{t}_f + \boldsymbol{b}_m) \\ \boldsymbol{t}_p &= \boldsymbol{W}_p \boldsymbol{t}_m + \boldsymbol{b}_p \end{aligned} \tag{5}$$

where $\boldsymbol{W}_m \in \mathbb{R}^{d \times d}$, $\boldsymbol{b}_m \in \mathbb{R}^d$, $\boldsymbol{W}_p \in \mathbb{R}^{d \times d}$, and $\boldsymbol{b}_p \in \mathbb{R}^d$ are parameters to be trained.

Training. Only maximizing agreement of the representations of trajectory augmentations will cause collapsing solution, which means that all the representations of trajectories are the same due to lacking dissimilarity computation among different underlying routes. To solve this problem, we adopt a simple but effective method, namely pair-based negative sampling [5], for preventing constant solution. Specifically, we denote the representations of a minibatch of trajectories and their augmented trajectories as $\boldsymbol{S}$ and $\boldsymbol{S}^a$, respectively. Based on that, the loss function with pair-based negative sampling for a positive pair $(\boldsymbol{t}_i, \boldsymbol{t}_i^a)$ can be expressed as follows.

$$l_i = -\log \frac{\exp(sim(\boldsymbol{t}_i, \boldsymbol{t}_i^a)/\tau)}{\sum_{\boldsymbol{t}_k \in \boldsymbol{S} \cup \boldsymbol{S}^a} \exp(\mathbb{I}_{[k \neq i]} sim(\boldsymbol{t}_i, \boldsymbol{t}_k)/\tau)} \tag{6}$$

where $\boldsymbol{t}_k$ is the representation of other trajectories in the same batch, Sim is the normalizing Euclidean distance, i.e., $sim(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^T \boldsymbol{v} / \|\boldsymbol{u}\| \, \|\boldsymbol{v}\|$, $\mathbb{I}_{[k \neq i]}$ is an indicator function that equals to 1 iff $k \neq i$, and τ denotes a temperature parameter. The pair-based negative sampling is an implicit method, which means that we do not need to explicitly sample negative trajectory pairs from a training dataset. Taking a positive pair $(\boldsymbol{t}_i, \boldsymbol{t}_i^a)$ as an example, $\boldsymbol{t}_k \in \boldsymbol{S} \cup \boldsymbol{S}^a$ are treated as negative samples for $\boldsymbol{t}_i$ and $\boldsymbol{t}_i^a$, where $k \neq i$. With the decrease of the loss, the distance between negative pairs and the similarity between positive pairs increases, which can make trajectory representations distinguishable to prevent collapsing solution.

Moreover, as shown in the loss function, our model does not implicitly implement the dissimilarity among trajectories from the same underlying route, i.e., the representations of two different trajectories T_i and T_j sampled from the same route could be the same. However, the sequential auto-encoder implicitly maintains the dissimilarity between T_i and T_j for better reconstruction even they are from the same underlying route. This demonstrates that our model has the ability to learn more consistent representations than the sequential auto-encoder.

4.4 Trajectory Similarity Computation

After training, we discard the projector part and adopt the encoder to convert trajectories to be compared into latent representations for trajectory similarity computation. Specifically, suppose T_i and T_j are two trajectories whose dissimilarity is to be determined. Their dissimilarity can be computed as follows.

$$\begin{aligned} \boldsymbol{t}_{f_i} &= LSTM(Emb(T_i, \boldsymbol{E})) \\ \boldsymbol{t}_{f_j} &= LSTM(Emb(T_j, \boldsymbol{E})) \\ D_h(T_i, T_j) &= ||\boldsymbol{t}_{f_i} - \boldsymbol{t}_{f_j}||_2 \end{aligned} \tag{7}$$

where $\boldsymbol{t}_{f_i} \in \mathbb{R}^d$ and $\boldsymbol{t}_{f_j} \in \mathbb{R}^d$ are latent representations of trajectory T_i and trajectory T_j, respectively. Next, $||\cdot||_2$ presents 2-norm of vectors, and $D_h(T_i, T_j)$ is the dissimilarity value between T_i and T_j.

Table 1: Statistics of Datasets.

Dataset	#Points	#Trips	Mean Length
Porto	74,269,739	1,233,766	60
ChengDu	60,048,544	1,164,451	51

5 EXPERIMENTS

5.1 Datasets and Baselines

We evaluate the performance of CL-TSim on two widely-used datasets, i.e., Porto[4] and ChengDu[5], compared with six representatives, i.e., DTW [33], EDR [3], LCSS [26], OWD [9], Frechet [2], and t2vec[6] [19]. We set the same sampling rate, i.e., 15 seconds, for two datasets. We partition each dataset into a training set and a testing set based on the starting timestamp of trajectories, where the first 1 million trajectories are used for training, and the remaining ones are used for testing. Statistics of the datasets are shown in Table 1.

5.2 Hyper-parameter Settings

Following previous work [19], we set the size of grid to 100 meters. To pretrain grid representations, we adopt a python library, called Gensim[7], to implement the Skip-gram model, in which the number of negative samples, learning rate, and epochs are set to 20, 0.025, and 5, respectively. For CL-TSim, we set the hidden size and the number of layers of LSTM to 128 and 1, respectively. The temperature τ, the batch size, initial learning rate, weight decays, and maximum epochs for training are set to 0.07, 128, 1e-4, 1e-4, and 40, respectively. We choose Adam as our optimizer and cosine annealing learning rate scheduler to dynamically adjust the learning rate. In the training stage, for each sample, we randomly choose k, down-sampling rate, and distorting rate from {2, 3, 4, 5}, {0, 0.2, 0.4, 0.6, 0.8}, and {0, 0.2, 0.4, 0.6, 0.8}, respectively, to generate the augmented trajectory, in which r_d and r_t are not set to 0 at the same time. For t2vec, we follow the original paper [19] to set the hyper-parameters and train the model. Specifically, we set the embedding dimension of grids, the hidden size of LSTM, and the number of layers of LSTM to 256, 128, and 3, respectively. Moreover, we also adopt the Skip-gram to pretrain the representations of t2vec. For EDR and LCSS, we set the threshold to 100 meters. For other baselines, there are no requirements for settings of hyper-parameters. In the following experiments, our results are reported based on these parameters, unless explicitly specified otherwise.

5.3 Evaluation Platform

Our method is implemented in Python and Pytorch, and trained using a GeForce GTX 1080 Ti GPU. The baseline methods are also written in Python. The platform runs the Ubuntu 16.04 operating system with 48-cores Intel(R) CPU E5-2650 v4 @ 2.20GHz 256GB RAM.

5.4 Metrics

We evaluate the above methods in terms of two aspects, i.e., self-similarity and cross-similarity [19].

[4] http://www.geolink.pt/ecmlpkdd2015-challenge
[5] https://gaia.didichuxing.com
[6] https://github.com/boathit/t2vec
[7] https://radimrehurek.com/gensim/

Table 2: Mean Rank and Precision w.r.t. $|Q_2 \cup \mathcal{D}_1|$.

		MR					Precision				
	DB size	20K	40K	60K	80K	100K	20K	40K	60K	80K	100K
Porto	DTW	23.757	47.192	69.068	93.995	118.033	0.271	0.203	0.170	0.144	0.124
	EDR	13.619	26.164	39.281	56.278	71.031	0.654	0.609	0.574	0.545	0.548
	LCSS	48.88	95.527	140.332	196.526	243.01	0.487	0.428	0.416	0.394	0.388
	OWD	5.470	10.006	14.473	20.676	25.855	0.782	0.727	0.700	0.675	0.659
	Frechet	3.669	6.646	9.714	14.143	18.010	0.802	0.753	0.720	0.693	0.677
	t2vec	1.292	2.408	3.611	5.103	6.351	0.858	0.806	0.780	0.751	0.738
	CL-TSim	**0.124**	**0.263**	**0.380**	**0.541**	**0.668**	**0.952**	**0.931**	**0.913**	**0.904**	**0.899**
ChengDu	DTW	10.632	21.140	31.784	42.958	54.052	0.271	0.190	0.143	0.121	0.106
	EDR	2.956	5.417	7.445	10.562	12.782	0.737	0.684	0.663	0.630	0.608
	LCSS	20.994	41.120	58.525	81.226	102.147	0.497	0.447	0.420	0.376	0.356
	OWD	1.483	2.903	4.274	5.767	7.339	0.767	0.714	0.674	0.648	0.627
	Frechet	1.806	3.561	5.244	7.092	8.992	0.623	0.525	0.493	0.444	0.411
	t2vec	1.333	2.611	3.711	5.017	6.445	0.756	0.695	0.667	0.639	0.615
	CL-TSim	**0.207**	**0.388**	**0.536**	**0.732**	**0.925**	**0.913**	**0.865**	**0.846**	**0.829**	**0.818**

Self-similarity. Given a set of trajectories, we randomly choose m trajectories and n trajectories, denoted as Q and $\mathcal{D}$, respectively. Then we define two empty set, denoted as Q_1 and Q_2. For each trajectory in Q, we create two sub-trajectories (called twins trajectories) by alternately taking points from it, and add the first sub-trajectory into Q_1 while the other is added into Q_2. Then for each trajectory in Q_1, called a query trajectory, we retrieve the most similar trajectory in $Q_2 \cup \mathcal{D}$, called a database trajectory. Obviously, the trajectories in Q_2 should be ranked before $\mathcal{D}$, since they are generated from the same trajectories with Q_1. Supposing that T_i is a query trajectory in Q_1, and T_j is the corresponding twins trajectory in Q_2, we calculate the similarity between T_i and $Q_2 \cup \mathcal{D}_1$, sort the trajectories based on the similarity, and denote the rank of T_j as r_i[8]. Based on that, two widely-used metrics [10], i.e., precision P and mean rank MR, are adopted for evaluation, which can be calculated as follows.

$$P = \frac{\sum_{T_i \in Q_1} p_i}{|Q_1|} \tag{8}$$

$$MR = \frac{\sum_{T_i \in Q_1} r_i}{|Q_1|} \tag{9}$$

where p_i equals to 1 iff r_i equals to 0, otherwise p_i equals to 0. A larger P or smaller MR value means a better self-similarity performance.

Cross-similarity. A good similarity measure should be able to preserve the similarity between two different trajectories, regardless of the data sampling strategies [18, 19]. An evaluation criterion from prior studies [19, 25, 27], namely Cross Distance Deviation (CDD), is also adopted to evaluate the performance. The calculation of CDD can be expressed as follows.

$$CDD = \frac{|d(T'_a(r_d), T'_b(r_d)) - d(T_a, T_b)|}{d(T_a, T_b)} \tag{10}$$

where T_a and T_b are two distinct trajectories with the original rate, $T'_a(r_d)$ is T_a's variant obtained by randomly dropping (or distorting) sample points with dropping (or distorting) rate r_d, and $T'_b(r_d)$ is T_b's variant obtained in the same way with $T'_a(r_d)$. To evaluate CDD, we randomly select a set of trajectories, denoted as $\mathcal{D}'$, from the testing set, and adopt the dropping (or distorting) strategy with different ratios. A smaller CDD value indicates that the evaluated similarity (i.e., distance) is much closer to the ground truth.

[8]The rank r_i equals to 0, when T_j ranks the first.

5.5 Overall Comparison

Effect of Database Sizes. Firstly, we study the effect of the database sizes, i.e., $|Q_2 \cup \mathcal{D}|$, as shown in Table 2, where $|Q_1|$ is fixed at 1K. As the database size grows, the performances of all methods deteriorate. Among all the traditional trajectory measurements, Frechet performs the best. The learning-based methods, i.e., t2vec and CL-TSim, perform better than others, which demonstrates the effectiveness of deep-learning techniques. Moreover, CL-TSim always outperforms the other competitors significantly regardless of database size, even when $|Q_2 \cup \mathcal{D}|$ equals to 100K. CL-TSim gives a low mean rank and high precision for the queries.

Robustness. To demonstrate the robustness of methods with non-uniform sampling rates and noises, we vary the down-sampling rate r_d and distorting rate r_t from 0.2 to 0.6 to down-sample and distort each trajectory in Q_1, $Q_2 \cup \mathcal{D}$, and $\mathcal{D}'$, where $|Q_1|$, $|Q_2 \cup \mathcal{D}|$, and $|\mathcal{D}'|$ are fixed at 1K, 20K, and 1K, respectively. We first conduct self-similarity experiments on Porto as shown in Tables 3 and 4. We observe that CL-TSim and t2vec outperforms all the traditional methods, and CL-TSim consistently performs the best among all the methods regardless of r_d and r_t. We also conduct cross-similarity experiments on Porto in Tables 5 and 6. When we vary the down-sampling rates, t2vec and CL-TSim consistently achieve the best compared with others. However, when varying the distorting rate, the performance of traditional methods (except for OWD) is better than those of t2vec and CL-TSim, and CL-TSim outperforms t2vec. Through the whole robustness experiments, we observe that the two learning-based methods, i.e., CL-TSim and t2vec, are more robust in self-similarity than in cross-similarity compared with other competitors. Moreover, comparing the two learning-based methods, we notice that CL-TSim is more robust than t2vec in most cases, which demonstrates that CL-TSim can learn more consistent representations of trajectories.

Table 3: Mean Rank and Precision w.r.t. Down-sampling Rate on Porto.

	MR					Precision				
r_d	0.2	0.3	0.4	0.5	0.6	0.2	0.3	0.4	0.5	0.6
DTW	27.386	38.482	58.356	98.378	170.379	0.282	0.266	0.228	0.196	0.136
EDR	118.18	248.633	471.154	784.239	1078.699	0.566	0.536	0.461	0.403	0.345
LCSS	48.406	63.742	56.979	63.494	83.233	0.445	0.446	0.434	0.396	0.387
OWD	5.610	6.716	9.931	16.354	24.684	0.740	0.698	0.649	0.581	0.482
Frechet	7.443	11.220	15.472	22.51	28.95	0.783	0.663	0.578	0.516	0.447
t2vec	1.406	1.610	1.768	3.029	3.907	0.839	0.831	0.827	0.788	0.755
CL-TSim	**0.252**	**0.319**	**1.497**	**1.795**	**3.366**	**0.943**	**0.917**	**0.893**	**0.881**	**0.817**

Table 4: Mean Rank and Precision w.r.t. Distorting Rate on Porto.

	MR					Precision				
r_t	0.2	0.3	0.4	0.5	0.6	0.2	0.3	0.4	0.5	0.6
DTW	23.82	23.895	23.964	24.095	23.975	0.268	0.271	0.267	0.262	0.265
EDR	13.028	12.349	12.795	12.045	10.582	0.673	0.669	0.671	0.680	0.671
LCSS	48.470	50.905	50.972	46.857	44.856	0.488	0.486	0.492	0.494	0.504
OWD	5.129	5.556	5.589	5.571	5.492	0.772	0.751	0.747	0.713	0.715
Frechet	3.604	3.635	3.617	3.696	3.622	0.803	0.803	0.801	0.798	0.799
t2vec	1.161	1.179	1.385	1.277	1.328	0.850	0.828	0.850	0.841	0.840
CL-TSim	**0.119**	**0.114**	**0.160**	**0.169**	**0.174**	**0.950**	**0.948**	**0.937**	**0.944**	**0.947**

Table 5: CDD w.r.t. the Down-sampling Rate on Porto.

Rate	0.2	0.3	0.4	0.5	0.6
DTW	0.1828	0.2743	0.3670	0.4638	0.5564
EDR	0.1457	0.2319	0.3256	0.4297	0.5315
LCSS	0.1369	0.1152	0.1867	0.2362	0.2341
OWD	0.3051	0.4616	0.6784	0.9821	1.4008
Frechet	0.0329	0.0417	0.0498	0.0544	0.0560
t2vec	0.0163	0.0232	**0.0282**	**0.0341**	**0.0404**
CL-TSim	**0.0110**	**0.0196**	0.0341	0.0579	0.0945

Table 6: CDD w.r.t. the Distorting Rate on Porto.

Rate	0.2	0.3	0.4	0.5	0.6
DTW	0.0014	0.0018	0.0023	0.0026	0.0029
EDR	**0.0005**	**0.0007**	**0.0009**	**0.0010**	**0.0011**
LCSS	0.0011	0.0015	0.0018	0.0021	0.0023
OWD	0.2071	0.2453	0.2680	0.2898	0.2998
Frechet	0.0010	0.0013	0.0017	0.0019	0.0021
t2vec	0.0151	0.0193	0.0221	0.0248	0.0265
CL-TSim	0.0050	0.0077	0.0084	0.0094	0.0107

Table 7: Effect of Grid Representations.

Metrics	P&F	R&NF	R&F
MR	0.668	2.315	325.42
Precision	0.899	0.794	0.494

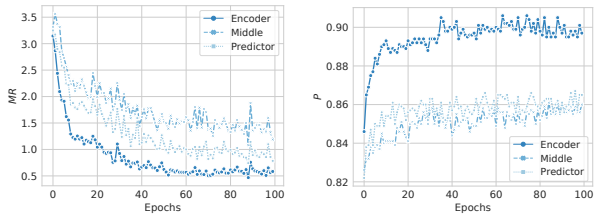
5.6 Effect of Main Components

In this subsection, we conduct self-similarity experiments on Porto to test the effect of each main component, in which $|Q_1|$ and $|Q_2 \cup \mathcal{D}|$ are fixed at 1K and 100K, respectively. When we evaluate the impact of a component, the other components or hyper-parameters are set by default.

Effect of Grid Representations. The input type of CL-TSim is different from previous contrastive learning methods, such as SimCLR [5], in which images with real value containing much information are inputs. Our inputs are discrete tokens, which need to be converted to representations by looking up an embedding table. Different initialization of grid representations usually lead to different performances. We denote three variants, i.e., P&F, P&NF, and R&F, to test the effect, in which P&F, P&NF and R&F represent pretrain-and-freeze, pretrain-and-freeze-free and random-and-freeze, respectively. The results are shown in Table 7. Compared with R&F, P&F performs better with a large margin since random initialization and freeze break the relationship among grids. Compared with P&NF, P&F performs better, which demonstrates that the loss function (cf. Equation 6) for training model cannot maintain spatial information well. As the training goes on, the pretrained grid representations will be modified and the spatial information will be dropped to some extent.

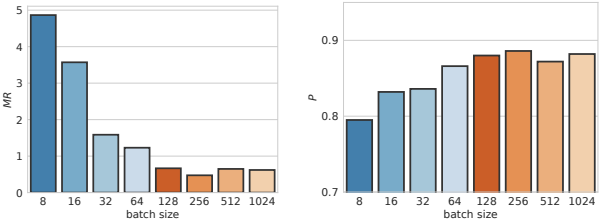
Effect of Representation Types. As shown in prior studies [5, 6], treating outputs of different layers as the latent representations, i.e., outputs of encoder $\boldsymbol{t}_f$, outputs of projector $\boldsymbol{t}_p$, outputs of middle of projector $\boldsymbol{t}_m$, usually have different performance. We use the same trained model but get the latent representations of trajectories by different parts of the model. Figure 2 shows the effect of epochs on *MR* and *precision*. From these results, we can see that $\boldsymbol{t}_f$ outputted by the encoder can get the best performance, and $\boldsymbol{t}_p$ is slightly better than $\boldsymbol{t}_m$, which demonstrates that $\boldsymbol{t}_f$ is more representative and should be used as representation of trajectories. We also observe that CL-TSim converges after around 40 epochs. This explains why we set the max epoch at 40 in other experiments.

Effect of Batch Sizes. Due to the pair-based negative sampling used in the loss function, the batch size will affect the number of negative samples [5–8]. We vary the batch size in the training phase from 8 to 1024. The results is shown in Figure 3. We can see



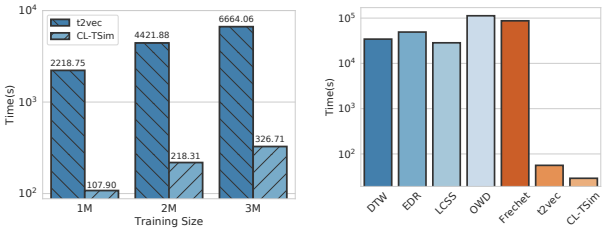
(a) MR (b) Precision

Figure 2: Effect of Representation Types



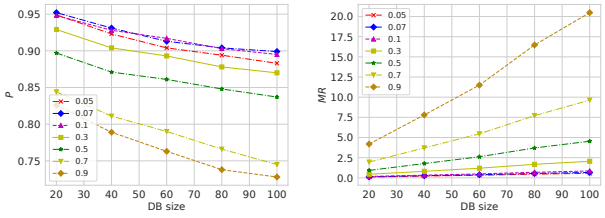
(a) MR (b) Precision

Figure 3: Effect of Batch Sizes



(a) Training time (b) Testing time

Figure 4: Model Efficiency



(a) MR (b) Precision

Figure 5: Effect of the temperature τ with respect to the DB size, i.e., $|Q_2 \cup \mathcal{D}_1|$.

that when the batch size is small, e.g., 8, the performance is the worst because there are few samples to help the model to generate distinguishable representations of trajectories, and it is easy for the model to collapse. As the batch size increases, the results of *mean rank* and *precision* will be better. When the batch size equals to 256, the results reach the best. After that, due to the increase of the probability that sampling trajectories from the same underlying route in a batch are wrongly treated as negative samples, the results will slightly degrade with the increase of the batch size.

5.7 Efficiency

As efficiency is important for real-world applications, we study the training time and testing time on Porto. The experimental settings are the same with that of the *Effect of Main Components*.

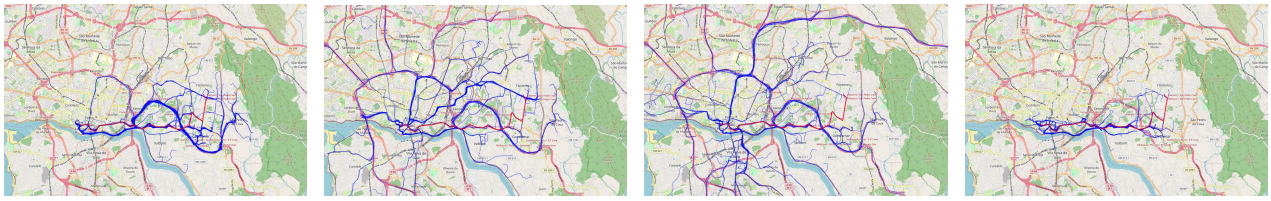
Training Time. To fairly compare the training efficiency, we present the average training time of one epoch under different training sizes (i.e., 1M, 2M, and 3M). The results are shown in Figure 4. We can see that CL-TSim is around 20x faster than t2vec for training one epoch. Next, t2vec will cost around 9 hours while CL-TSim only cost around 1 hours to achieve convergency, which demonstrates the efficiency of the proposed model.

Testing Time. We also present the testing time of each method in Figure 4, in which the time of t2vec and CL-TSim contain two parts, i.e., encoding 101K trajectories into representations using GeForce GTP 1080 Ti and evaluating the similarity among trajectories. Due to the low efficiency of baselines, we run them with 20 multi-processors implemented on Multiprocessing library of python while evaluate t2vec and CL-TSim on one core of CPU. We can see that CL-TSim is much faster than others with a large margin. Specifically, compared with t2vec, CL-TSim costs less testing time because t2vec has to use more complex neural networks (i.e., three Bi-LSTM layers with hidden size of 128) to maintain its effectiveness (i.e., smaller or larger hidden size used in t2vec will decrease the performance reported in [19]) while we just adopt one LSTM layer with hidden size of 128 in CL-TSim.

5.8 Hyper-Parameters Sensitivity

Effect of temperature τ. We first study the effect of temperature τ in Equation 6, which is important in the contrastive learning paradigm. Totally, we conduct experiment in terms of self-similarity to show the effect of τ. Firstly, we vary τ from 0.05 to 0.9 and the DB size from 20K to 100K at the same time. Then we report the MR and Precision. The results are shown in Figure 5. We can see that with the increase of the DB size, the performance decrease, i.e., MR increases and Precision decreases. Besides, we can observe that the performances are competitive when τ is set to 0.05 or 0.07, which are obviously better than the other settings, e.g., τ=0.1. Moreover, settting τ to 0.07 can achieve the best in most cases.

Effect of hidden size d. Then, we vary the hidden size from 16 to 256 and the DB size from 20K to 100K and report the performance in terms of self-similarity as shown in Figure 7, in which each line presents each value of DB size. For example, the red dotted line presents the performance when the hidden size varies from 16 to 256 and the DB size equals to 20K. Thus, we can see that the performance become better with the increase of the hidden size. This observation is obvious especially when the hidden size increases from 16 to 64. After that with the further increase of the hidden size, the decrease of MR metric is very slight and the increase of precision metric is limited. However, the training time of the proposed model largely increase, e.g., training one epoch on 1M trajectories costs around 150s when the hidden size equals to 256 while only costs around 107s when the hidden size equals to 128. Therefore, we set the hidden size to 128 in the other experiments by simultaneously considering the balance between effectiveness and efficiency.

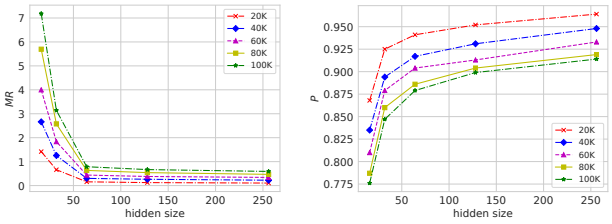


(a) CL-TSim (b) t2vec (c) R&NF (d) OWD

Figure 6: Visualization on Porto. The red trajectory is the query trajectory. These blue lines are 100 trajectories which are the most similar to the query trajectory returned by different models.

Table 8: Performance Comparison for Different Methods on DTW, EDR, OWD, Frechet distances.

	DTW		EDR		OWD		Frechet	
Methods	Overlap@20	Overlap@50	Overlap@20	Overlap@50	Overlap@20	Overlap@50	Overlap@20	Overlap@50
t2vec	1.899	4.991	2.351	5.138	2.608	5.400	3.442	7.526
CL-Tsim	**2.855**	**8.683**	**3.667**	**8.960**	**4.571**	**11.343**	**6.480**	**15.743**



(a) MR (b) Precision

Figure 7: Effect of the hidden size d with respect to the DB size, i.e., $|Q_2 \cup \mathcal{D}_1|$.

5.9 Visualization

Previous tasks, i.e., self-similarity and cross-similarity, mainly show the robustness of a measurement to non-uniform sampling rate and noisy points. Considering an extreme situation that one neural network can map trajectories from the same underlying route into the same representations while the distance between different underlying routes is randomly assigned. In this situation, the performance of the neural network on self-similarity and cross-similarity tasks will be perfect. However, it may violate the common sense (spatial proximity), i.e., trajectories located in the adjacent streets should be intuitively more similar than trajectories that are far apart from each other.

To demonstrate that CL-TSim obeys this common sense, we randomly sample a trajectory from the Porto testing set and treat it as a query trajectory. Then we compute similarity between the query and the remaining trajectories in the testing set and return 100 trajectories that are the most similar to the query trajectory. The results are visualized in Figure 6. From this figure, we can see that CL-TSim can preserve spatial information since the top-100 similar trajectories obtained by it are around in the query trajectory. Moreover, we observe that if $\boldsymbol{E}$ is not initialized by pretrained representations, the results will be more diverse, which shows the importance of pretraining operation. Moreover, we observe that OWD tends to return the trajectories that are parallel to the query trajectory since it measures the distance between trajectories as the average minimal distance from all sample points of query trajectory to the line-approximated trajectory [24].

Besides, we can see that the obtained trajectories of CL-TSim are more compact than those of t2vec. To demonstrate this observation in a quantitive way, we follow the evaluation of neural approximation methods [31] to calculate the overlaps of the most similar k trajectories between neural network-based method and traditional method. Specifically, we randomly choose 1K trajectories as query and 100K trajectories as data from the Porto testing set. Then we calculate the mean of the number of overlaps among the most similar k trajectories between neural network-based measurement and traditional measurement. The spatial proximity is naturally guaranteed by traditional methods. If the first neural network-based measurement has larger number of overlaps (i.e., a higher mean value) with all the traditional methods than the second neural network-based measurement does, we can claim that the first measurement performs better than the second measurement in preserving spatial proximity. The results are shown in Table 8. Through this table, we observe that CL-TSim has a higher mean value, which further demonstrate the superiority of CL-TSim to preserve spatial proximity compared with t2vec.

6 CONCLUSION

In this paper we propose a novel trajectory similarity computation model based on contrastive learning, called CL-TSim. In order to achieve high effectiveness and efficiency, we address a few challenges by employing a contrastive learning mechanism to learn the latent representations of trajectories and calculate the similarity between trajectories, where two augmentation strategies including point down-sampling and point distorting are developed that target robust trajectory similarity computation. Extensive experiments based on real trajectories datasets confirm the superiority of our proposed model over the state-of-the-art approaches in terms of precision, mean rank, and training efficiency.

7 ACKNOWLEDGEMENT

This work is partially supported by NSFC (No. 61972069, 61836007 and 61832017), and Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021).

REFERENCES

[1] Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. 2018. Subtrajectory Clustering: Models and Algorithms. In *SIGMOD*. 75–87.

[2] Helmut Alt and Michael Godau. 1995. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.* 5 (1995), 75–91.

[3] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and Fast Similarity Search for Moving Object Trajectories. In *SIGMOD*. 491–502.

[4] Meng Chen, Yan Zhao, Yang Liu, Xiaohui Yu, and Kai Zheng. 2022. Modeling spatial trajectories with attribute representation learning. *TKDE* 34, 4 (2022), 1902–1914.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. *ArXiv* abs/2002.05709 (2020).

[6] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. Big Self-Supervised Models are Strong Semi-Supervised Learners. *ArXiv* abs/2006.10029 (2020).

[7] Xinlei Chen, Haoqi Fan, Ross B. Girshick, and Kaiming He. 2020. Improved Baselines with Momentum Contrastive Learning. *ArXiv* abs/2003.04297 (2020).

[8] Xinlei Chen and Kaiming He. 2020. Exploring Simple Siamese Representation Learning. *ArXiv* abs/2011.10566 (2020).

[9] Chi-Yin Chow, Mohamed F. Mokbel, and Walid G. Aref. 2009. Casper*: Query processing for location services without compromising privacy. *ACM Trans. Database Syst.* 34, 4 (2009), 24:1–24:48. https://doi.org/10.1145/1620585.1620591

[10] Yue Cui, Hao Sun, Yan Zhao, Hongzhi Yin, and Kai Zheng. 2021. Sequential-knowledge-aware Next POI Recommendation: A Meta-learning Approach. *TOIS* (2021).

[11] Liwei Deng, Hao Sun, Rui Sun, Yan Zhao, and Han Su. 2022. Efficient and Effective Similar Subtrajectory Search: A Spatial-aware Comprehension Approach. *TIST* 13, 3 (2022), 35:1–35:22.

[12] Terrance Devries and Graham W. Taylor. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. *ArXiv* abs/1708.04552 (2017).

[13] Spyros Gidaris, Praveer Singh, and N. Komodakis. 2018. Unsupervised Representation Learning by Predicting Image Rotations. *ArXiv* abs/1803.07728 (2018).

[14] R. H. Güting and M. Schneider. 2005. Realm-based spatial data types: The ROSE algebra. *VLDBJ* 4 (2005), 243–286.

[15] Suining He and S. Chan. 2016. Tilejunction: Mitigating Signal Noise for Fingerprint-Based Indoor Localization. *TMC* 15 (2016), 1554–1568.

[16] S. Hochreiter and J. Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9 (1997), 1735–1780.

[17] Andrew G. Howard. 2014. Some Improvements on Deep Convolutional Neural Network Based Image Classification. *CoRR* abs/1312.5402 (2014).

[18] Guanyao Li, Chih-Chieh Hung, Mengyun Liu, Linfei Pan, W. Peng, and S. Chan. 2021. Spatial-Temporal Similarity for Trajectories with Location Noise and Sporadic Sampling. In *ICDE*. 1224–1235.

[19] Xiucheng Li, Kaiqi Zhao, G. Cong, Christian S. Jensen, and Wei Wei. 2018. Deep Representation Learning for Trajectory Similarity Computation. In *ICDE*. 617–628.

[20] Shuncheng Liu, Han Su, Yan Zhao, Kai Zeng, and Kai Zheng. 2021. Lane Change Scheduling for Autonomous Vehicle: A Prediction-and-Search Framework. In *KDD*. 3343–3353.

[21] Tomas Mikolov, Kai Chen, G. Corrado, and J. Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR*.

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, G. Corrado, and J. Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. 3111–3119.

[23] Sayan Ranu, P Deepak, A. Telang, Prasad Deshpande, and S. Raghavan. 2015. Indexing and matching trajectories under inconsistent sampling rates. In *ICDE*. 999–1010.

[24] Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. 2019. A survey of trajectory distance measures and performance evaluation. *VLDBJ* 29 (2019), 3–32.

[25] Han Su, Kai Zheng, Haozhou Wang, Jiamin Huang, and Xiaofang Zhou. 2013. Calibrating trajectory data for similarity-based analysis. In *SIGMOD*. 833–844.

[26] M. Vlachos, D. Gunopulos, and G. Kollios. 2002. Discovering similar multidimensional trajectories. In *ICDE*. 673–684.

[27] Haozhou Wang, Han Su, Kai Zheng, S. Sadiq, and Xiaofang Zhou. 2013. An Effectiveness Study on Trajectory Similarity Measures. In *ADC*. 13–22.

[28] Zheng Wang, Cheng Long, G. Cong, and Yiding Liu. 2020. Efficient and effective similar subtrajectory search with deep reinforcement learning. In *VLDB*, Vol. 13. 2312 – 2325.

[29] Peilun Yang, Hanchen Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2021. T3S: Effective Representation Learning for Trajectory Similarity Computation. *ICDE*, 2183–2188.

[30] Tao yang Fu and Wang-Chien Lee. 2020. Trembr: Exploring Road Networks for Trajectory Representation Learning. *TIST* 11 (2020), 10:1–10:25.

[31] Di Yao, G. Cong, Chao Zhang, and Jingping Bi. 2019. Computing Trajectory Similarity in Linear Time: A Generic Seed-Guided Neural Metric Learning Approach. In *ICDE*. 1358–1369.

[32] Di Yao, Chao Zhang, Zhihua Zhu, Jianhui Huang, and Jingping Bi. 2017. Trajectory clustering via deep representation learning. *IJCNN* (2017), 3880–3887.

[33] Byoung-Kee Yi, H. Jagadish, and C. Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. In *ICDE*. 201–208.

[34] Daqing Zhang, N. Li, Z. Zhou, C. Chen, Lin Sun, and Shijian Li. 2011. iBAT: detecting anomalous taxi trajectories from GPS traces. In *UbiComp*. 99–108.

[35] Hanyuan Zhang, Xinyu Zhang, Qize Jiang, Baihua Zheng, Zhenbang Sun, Weiwei Sun, and Changhu Wang. 2020. Trajectory Similarity Learning with Auxiliary Supervision and Optimal Matching. In *IJCAI*. 3209–3215.

[36] Yifan Zhang, An Liu, Guanfeng Liu, Zhixu Li, and Qing Li. 2019. Deep Representation Learning of Activity Trajectory Similarity Computation. In *ICWS*. 312–319.

[37] Yan Zhao, Shuo Shang, Yu Wang, Bolong Zheng, Quoc Viet Hung Nguyen, and Kai Zheng. 2018. REST: A Reference-based Framework for Spatio-temporal Trajectory Compression. In *KDD*. 2797–2806.

[38] Yan Zhao, Shuo Shang, Yu Wang, Bolong Zheng, Quoc Viet Hung Nguyen, and Kai Zheng. 2018. Rest: A reference-based framework for spatio-temporal trajectory compression. In *SIGKDD*. 2797–2806.

[39] Kai Zheng, Yan Zhao, Defu Lian, Bolong Zheng, Guanfeng Liu, and Xiaofang Zhou. 2019. Reference-based framework for spatio-temporal trajectory compression and query processing. *TKDE* 32, 11 (2019), 2227–2240.

[40] Kai Zheng, Yan Zhao, Defu Lian, Bolong Zheng, Guanfeng Liu, and Xiaofang Zhou. 2019. Reference-based framework for spatio-temporal trajectory compression and query processing. *TKDE* 32, 11 (2019), 2227–2240.

[41] Kai Zheng, Yu Zheng, Xing Xie, and Xiaofang Zhou. 2012. Reducing Uncertainty of Low-Sampling-Rate Trajectories. In *ICDE*. 1144–1155.

[42] Hao Zhou, Yan Zhao, Junhua Fang, Xuanhao Chen, and Kai Zeng. 2019. Hybrid route recommendation with taxi and shared bicycles. *Distributed and Parallel Databases* (2019), 1–21.