Computing Trajectory Similarity in Linear Time: A Generic Seed-Guided Neural Metric Learning Approach

Di Yao[1,4], Gao Cong[2], Chao Zhang[3], Jingping Bi[1,4]
[1]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China*
[2]*School of Computer Science and Engineering, Nanyang Technological University, Singapore*
[3]*Computer Science Department, University of Illinois at Urbana-Champaign, Urbana, IL, USA*
[4]*University of Chinese Academy of Sciences, Beijing, China*
[1,4]{yaodi, bjp}@ict.ac.cn,[2]gaocong@ntu.edu.sg, [3]czhang82@illinois.edu

***Abstract*—Trajectory similarity computation is a fundamental problem for various applications in trajectory data analysis. However, the high computation cost of existing trajectory similarity measures has become the key bottleneck for trajectory analysis at scale. While there has been much research effort for reducing the complexity, they are specific to one similarity measure and often yield limited speedups. We propose NEUTRAJ to accelerate trajectory similarity computation. NEUTRAJ is generic to accommodate any existing trajectory measures and fast to compute the similarity of a given trajectory pair in linear time. Furthermore, NEUTRAJ is elastic to collaborate with all spatial-based trajectory indexing methods to reduce the search space. NEUTRAJ samples a number of seed trajectories from the given database, and then uses their pair-wise similarities as guidance to approximate the similarity function with a neural metric learning framework. NEUTRAJ features two novel modules to achieve accurate approximation of the similarity function: (1) a spatial attention memory module that augments existing recurrent neural networks for trajectory encoding; and (2) a distance-weighted ranking loss that effectively transcribes information from the seed-based guidance. With these two modules, NEUTRAJ can yield high accuracies and fast convergence rates even if the training data is small. Our experiments on two real-life datasets show that NEUTRAJ achieves over 85% accuracy on Fréchet, Hausdorff, ERP and DTW measures, which outperforms state-of-the-art baselines consistently and significantly. It obtains 50x-1000x speedup over bruteforce methods and 3x-500x speedup over existing approximate algorithms, while yielding more accurate approximations of the similarity functions.**

***Index Terms*—deep metric learning, trajectory similarity, linear time**

I. INTRODUCTION

Computing the similarity between two trajectories is a primitive that is fundamental to many searching and mining tasks for trajectory analysis. Various measures have been proposed to capture the intrinsic structural similarities between trajectories, including Dynamic Time Warping(DTW) [30], the Hausdorff distance [3], the Fréchet distance [2], Edit distance with Real Penalty(ERP) [9] and *etc*. These similarity measures have played a key role in leading to enormous success in anomaly detection [17], duplicate detection [26], trajectory clustering [6], and many other domains.

Unfortunately, the high computation cost of existing trajectory similarity measures has become the *de facto* bottleneck for trajectory analysis at scale. To compute the similarity between two trajectories, existing techniques often require to align the points in the two trajectories, accumulate the information among all aligned pairs, and finally produce the distance. Such a process incurs quadratic and even super-quadratic time complexity and limits many trajectory mining algorithms to scale to large datasets. For instance, it takes us more than 6.5 hours to compute the pair-wise Hausdorff distances for merely $\sim$8000 human GPS trajectories on a high-end server. As massive trajectory data are being collected at an unprecedentedly massive scale in many kinds of scenarios, fast trajectory similarity computation under different measures has become a pressing need.

The difficulties in fast trajectory similarity computation are two-fold. The first is the complicated nature of trajectory similarity computation. A pair of input trajectories may have completely different lengths, and the best alignment of the two trajectories is subject to flexible shifting and scaling instead of exact head-to-tail matching. As such, most existing techniques have to employ a scan-and-align mechanism for determining the best matching, and it is hard to decouple the matching process and reduce the time cost. The second is the variations across different similarity measures. Prevailing trajectory measures (*e.g.*, DTW, Hausdorff, Fréchet) differ a lot in their definitions and computation mechanisms. It is challenging to design a generic accelerating strategy that accommodates all the existing measures.

There have been considerable research efforts attempting to accelerate trajectory similarity computation for various tasks. Such efforts can be generally categorized into two lines. The first [10], [14], [29] is to reduce the involved number of computations at a global level. However, the techniques along this line are exclusively designed for the top-k similarity search task. Instead of reducing the computation complexity for an ad-hoc pair of trajectories, they focus on designing indexing and pruning strategies for a given trajectory database and reducing the number of computations for top-k similarity

search. As such, they cannot be applied for tasks that require the distances between all trajectory pairs such as trajectory clustering and anomaly detection. The second line aims at directly reducing the time complexity for trajectory similarity computation with approximate algorithms. Different strategies have been proposed for different measures, such as locality sensitive hashing (LSH) for the Fréchet distance [12] and the presorting strategies for DTW [16]. Unfortunately, the techniques are designed for one specific distance measure and not applicable to any other measure.

We propose a model that drastically accelerates trajectory similarity computation for any measures. Our proposed model, named NEUTRAJ, is an approximate approach based on neural metric learning. NEUTRAJ does not need any external information, it samples a pool of seed trajectories from the database and use their pair-wise similarities as guidance. Specifically, NEUTRAJ learns a neural network that jointly embeds input trajectories and approximates the distance function. It has the following attractive characteristics:

- **Generic:** Unlike previous methods that are specific to one trajectory similarity measure, NEUTRAJ is generic enough to support any existing measures. It can thus be used for accelerating most trajectory mining tasks based on different measures.
- **Fast:** Given an ad-hoc pair of trajectories, NEUTRAJ is able to compute their similarity in $O(L)$ time complexity, where L is the length sum of the pair. In practice, we observed it at least 50x faster than accurate bruteforce computation.
- **Accurate:** NEUTRAJ achieves superb approximation performance in practice. On two real trajectory datasets, NEUTRAJ achieves over 85% hitting ratio and less than 50m average error distance on top-10 similar trajectory search task for Fréchet, Hausdorff, DTW and ERP. Meanwhile, it obtains more than two times higher hitting ratios compared with state-of-the-art trajectory similarity approximation methods [12], [16].
- **Elastic:** NEUTRAJ embeds the trajectory without losing spatial information which makes it elastic to extend by other indexing and pruning strategies. In tasks that similarities of all pairs are non-essential, NEUTRAJ is able to cooperate with existing indexing methods [10], [14], [29] for reducing the computing space.

The core of NEUTRAJ is a deep metric learning framework that uses recurrent neural networks (RNNs) to generate trajectory embeddings. We sample a pool of seed trajectories from the database and compute their pair-wise similarities. With the computed seed similarities as guidance, we design a pair-wise loss to optimize the network for fitting seed similarities. NEUTRAJ features two novel modules that encourage the network to approximate the similarity function accurately: (1) *Spatial attention memory (SAM).* Vanilla RNNs along with its existing variants (GRU, LSTM) can only model one sequence without considering between-sequence correlations. Our designed SAM units memorize the information from previously processed trajectories with the attention mechanism, and capture the correlations between training trajectories to produce better trajectory embeddings. (2) *Distance-weighted ranking loss.* One difficulty of making use of the seed trajectories is the dilemma between efficiency and effectiveness. On one hand, training the network sufficiently would preferably iterate over all pairs of trajectories. On other hand, a full enumeration of all pairs of trajectories incurs expensive computation time. To address this dilemma, we propose a distance-weighted sampling strategy to focus on the more discriminative training pairs. Along with the weighted sampling strategy, distance-weighted ranking loss is a ranking loss that learns the parameters of the network to conform to the guidance from the seeds. With these two novel modules, NEUTRAJ can yield high accuracies and fast convergence rates even if the training data is small.

Our contributions can be summarized as follows:

1) We propose a neural metric learning method for accelerating trajectory similarity computation under different measures. To the best of our knowledge, NEUTRAJ is the first method that supports accelerating generic trajectory similarity measures, making it widely applicable to many applications.
2) We propose the spatial attention memory unit to model the correlation between spatially close trajectories based on an attention network and external memory tensor.
3) We design a weighted sampling and learning module that fully unleashes the power of seed trajectories. Compared with existing architecture, our learning module yields faster convergence rates and higher accuracies.
4) We conduct extensive experiments on two real trajectory datasets and four popular trajectory similarity measures. The results demonstrate that the proposed model consistently outperforms state-of-the-art baselines in both accuracy and efficiency.

II. RELATED WORK

In this section, we provide an overview of existing studies related to NEUTRAJ from three perspectives: (1) trajectory similarity computation; (2) deep metric learning; and (3) memory networks.

Trajectory Similarity Computation. Various techniques have been proposed to accelerate trajectory similarity computation, which can be broadly categorized into two categories. The first category uses indexing and pruning techniques to reduce the involved number of computations at a global level. Most techniques in this category employ tree-based index structures [10], [11], [16], [18], [26] , such as K-D tree or R-tree to organize the trajectory data in a hierarchy. Based on the index, bounding-box-based pruning techniques are employed to eliminate unnecessary computations. Thus sub-trajectories [10], [11] or point segments [14], [29] in a bounding box which are too faraway to belong to the top-k results are pruned. However, the techniques in this category are specifically designed for the top-k similarity search problem. They do not reduce the time complexity of computing the similarity between a pair of trajectories. Hence, they cannot be applied for tasks that

require the distances of all pairs such as trajectory clustering and anomaly detection.

The second category aims at designing approximate algorithms to speed up similarity computation for a pair of trajectories. Most techniques in this category treat each trajectory as a spatial curve and address the problem from the angel of computational geometry. Focusing on the Hausdorff distance, Farach-Colton *et al.* and Backurs *et al.* [4], [13] proposed embedding-based methods for approximating nearest neighbor search. Salvador *et al.* [1] proposed an approximate algorithm which can fast compute the DTW distance. Thanawin *et al.* [25] proposed a method which omits the square computation step to speed up DTW computation. Li *et al.* [19] proposed a new trajectory similarity measure based on road network and employ deep representation learning to approximate it. Very recently, Driemel *et al.* [12] proposed a locality sensitive hashing (LSH) based algorithm for fast computing the Fréchet and Hausdorff distances. Although these algorithms can achieve high computation efficiency, they suffer from two shortcomings. First, they rely on hand-crafted heuristics and could lead to unsatisfactory accuracies. In practice, we observed that these algorithms generate poor approximations in many cases. Second, they are all designed for one or two specific measures. It is hard to adapt these techniques for other similarity measures.

Deep Metric Learning. NEUTRAJ is related to the recent development of deep metric learning, which aims at learning a distance function that measures how similar two objects are based on neural networks. Bromley *et al.* [5] pioneered deep metric learning and proposed the classic Siamese network for signature verification. Qian *et al.* [24] used precomputed activation features to learn a feature embedding for classification. Pei *et al.* [23] extended the Siamese network to learn a similarity metric for sentences. Our method differs from the above models in two aspects. First, they are all designed for modeling one sequence independently, while ours employs the spatial attention mechanism to capture the correlations among all the trajectories. Second, they all use random sampling to generate training samples and could suffer from low convergence for trajectory data.

Memory Network. NEUTRAJ employs memory network to capture the relations between trajectories. The embryonic form of memory network is proposed in [15] to solve the tasks that need to memory long term information. [28] first employs a long-term memory component and define the read and write operation for questing answering(QA). Then [27] extends the architecture to recurrent neural network. [8], [28] extends the memory to a hierarchical structure. But these memory structures are designed without considering the spatial information and can not directly use for trajectory modeling.

TABLE I
NOTATIONS USED IN THIS PAPER

Notations	Description
$\mathscr{T}, \mathfrak{S}$	A trajectory database and a pool of N seeds trajectories which are random sampled from the database.
T	A trajectory in $\mathscr{T}$ which consist of a sequence of coordinate tuples.
$\mathbf{D}, \mathbf{S}$	The distance and similarity matrices of $\mathfrak{S}$ which have the same size: $N \times N$.
$\mathbf{M}$	The memory tensor which stores the spatial information of $P \times Q$ grids.
$\mathbf{E_i}, \mathbf{E_j}$	d-dimensional embedding vectors of T_i and T_j learnt by NEUTRAJ.
X_t	The input of NEUTRAJ at t-time step which contains the coordinate input X_t^c and grid input X_t^g.
$\mathbf{W}, \mathbf{U}, \mathbf{b}$	The linear weights and bias in SAM units.
$\mathbf{f_t}, \mathbf{i_t}, \mathbf{o_t}$	The forget, input and output gates in SAM-LSTM units.
$\mathbf{r_t}, \mathbf{u_t}$	The reset and update gates in SAM-GRU units.
$\mathbf{s_t}$	Novel spatial gate in SAM units which controls the $read$ and $write$ operations on $\mathbf{M}$.
$\mathbf{c_t}, \hat{\mathbf{c_t}}$	The cell state and intermediate cell state in SAM units at t-time step which store the information of the processed $t-1$ steps.
$\mathbf{h_t}, \mathbf{h_{t-1}}$	The hidden states in SAM units at time step t and $t-1$.
$\mathbf{G_t}$	The spatial information matrix of X_t with shape $(2w+1)^2 \times d$, where w is the scanning bandwidth.
$\mathbf{A}$	The spatial attention weight that reflects the similarity weights of $\hat{\mathbf{c_t}}$ over $\mathbf{G_t}$.
$\mathbf{c_t^{cat}}, \mathbf{c_t^{his}}$	The intermediate concatenated state and the final historical state in memory $read$ operation at t-time step.
T_a	The anchor trajectory which is sampled from the seeds for training NEUTRAJ.
$\mathscr{T}_a^s, \mathscr{T}_a^d$	n similar trajectories and n dissimilar trajectories of T_a which are sampled from the seeds for fitting the pair-wise similarities.
$\mathbf{S}_a^s, \mathbf{S}_a^d$	The ground truth similarities of both similar and dissimilar pairs of T_a
$\hat{\mathbf{S}_a^s}, \hat{\mathbf{S}_a^d}$	The similarities of T_a which are calculated by NEUTRAJ.

III. PRELIMINARIES

A. Problem Definition

We consider a trajectory database $\mathscr{T}$ and a trajectory similarity function $f(\cdot,\cdot)$. Each trajectory $T \in \mathscr{T}$ is a sequence of points recording the trace of a moving object. Although each sample point in a trajectory has a sampling time, we only focus on finding trajectories with similar shape, regardless of the time information. Without loss of generality, we consider two-dimensional trajectories. That is, each trajectory $T = [X_1^c, ..., X_t^c, ...]$ is a sequence of tuples where $X_t^c(x_t, y_t)$ is the $t-th$ location of the object. For any two trajectories $T_i, T_j \in \mathscr{T}$, the function $f(T_i, T_j)$ measures the similarity between T_i and T_j. Here, $f(\cdot,\cdot)$ could be the DTW similarity, the Hausdorff distance, the Fréchet distance, or any other trajectory similarity measure. We omit the detailed definitions of these measures due to the space limit.

Our considered problem is to compute the similarity for an ad-hoc pair of trajectories from $\mathscr{T}$ under the similarity function $f(\cdot,\cdot)$. However, for most prevailing similarity measures, computing the similarity between a pair of trajectories incurs quadratic or even super-quadratic time complexity. Hence, the research question is: how can we accelerate computing the similarities for any pair of trajectories in $\mathscr{T}$?

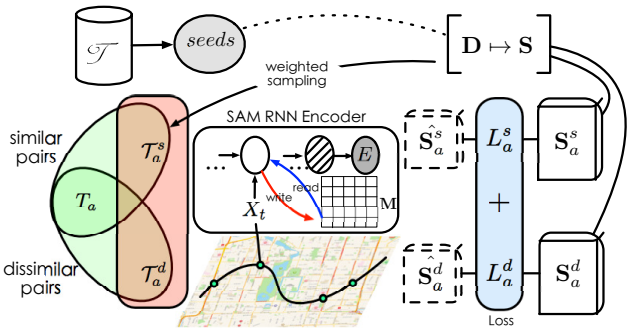


Fig. 1. Architecture of NEUTRAJ. Taking similar and dissimilar pairs of anchor T_{a_i} as input, NEUTRAJ first generates the embedding of each trajectory. And then fits the pair-wise similarity guided by the ground truth in $\mathbf{S}$.

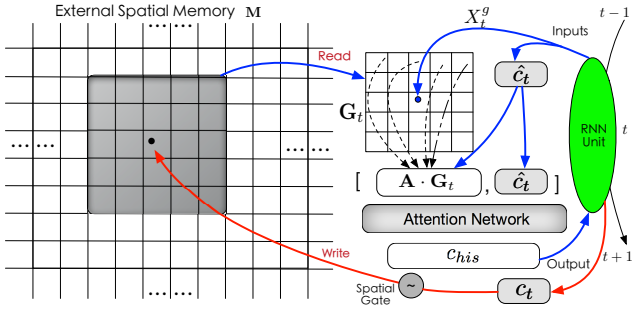


Fig. 2. Illustration of the proposed spatial attention memory(SAM) with scanning bandwidth $w = 2$. At each time step, SAM takes two inputs, the input grid X_t^g and the intermediate cell state $\hat{\mathbf{c}}_\mathbf{t}$. The reader first scans the memory $\mathbf{M}$ to get $(2*2+1)^2 = 25$ grid embeddings. Then it calculates and outputs the attention cell state $\mathbf{c}_\mathbf{t}^\mathbf{his}$. The writer (red lines) updates the $\mathbf{M}$ with the cell state $\mathbf{c_t}$ in the recurrent unit.

B. Overview of NEUTRAJ

At the high level, NEUTRAJ is a neural metric learning framework. It randomly samples N trajectories from $\mathscr{T}$ as the pool of seeds $\mathfrak{S}$ and computes a symmetric $N \times N$ distance matrix $\mathbf{D}$ for $\mathfrak{S}$. Then it transforms the original distance matrix into a normalized similarity matrix $\mathbf{S}$. Leveraging the matrix $\mathbf{S}$ as guidance, NEUTRAJ further learns a neural network, which maps arbitrary-length trajectories into low-dimensional space to capture their similarities. More formally, for any two input trajectories T_i and T_j $(i, j \in [1, ..., N])$, NEUTRAJ projects them to two d-dimensional vectors $\mathbf{E_i}$ and $\mathbf{E_j}$. The learned mapping should be similarity preserving, namely $f(T_i, T_j) \approx g(T_i, T_j)$ where $g(\cdot, \cdot)$ is the similarity between $\mathbf{E_i}$ and $\mathbf{E_j}$ in the embedding space. Figure 1 illustrates the architecture of NEUTRAJ. It consists of two major parts: spatial attention memory(SAM) augmented RNN encoder and seed-guided metric learning method.

SAM Augmented RNN Encoder. NEUTRAJ relies on recurrent neural networks (RNN) to model the trajectory and takes the last hidden state of RNN as the embedding vector. However, as aforementioned, vanilla RNNs and its variants (GRU, LSTM) capture the information of each sequence independently. For trajectory similarity computing, the correlations between trajectories are critical. It is important to leverage the information of spatially close trajectories previously seen to guide the metric learning process. Thus, we design a spatial attention memory module in NEUTRAJ. It employs a spatial memory tensor to store the spatial information of previously processed trajectories. The memory tensor underpins $read$ and $write$ operations over the entire space based on the soft attention mechanism, such that the information of previously seen trajectories can be encoded and retrieved on demand.

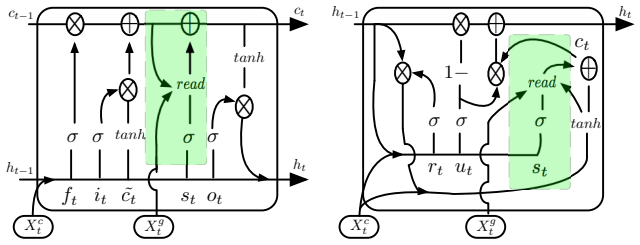
Seed-Guided Neural Metric Learning. Based on SAM augmented RNN, NEUTRAJ builds a seed-guided metric learning architecture to consume a pair of trajectories, and learns the network to approximate the similarity matrix $\mathbf{S}$. Existing metric learning methods employ random sampling to produce training pairs, which implies all trajectories are equally weighted. But this assumption dose not hold in trajectory metric learning as it ignores the spatial proximity between trajectories. Particularly, we develop a distance-weighted sampling procedure and a ranking loss objective to solve this problem. Unlike previous random sampling, the distance-weighted sampling focus on the more discriminative training pairs from the seed trajectories. With the weighted sampling strategy, each seed trajectory T_a is associated with one similar list $\mathscr{T}_a^s$ and one dissimilar list $\mathscr{T}_a^d$ in the pool. The ranking loss then learns the parameters of the network for fitting the similarities to $\mathbf{S}$ and preserving the ranking order in both similar and dissimilar pairs.

IV. SAM AUGMENTED RNN ENCODER

In this section, we introduce the Spatial Attention Memory (SAM) module that augments existing RNN architectures for trajectory encoding. Below, we first introduce the spatial attention memory structure. Then we present two fancy RNN units, SAM-augmented LSTM and SAM-augmented GRU, which augment existing recurrent neural networks with the SAM. Finally, we detailed the $read$ and $write$ operations of the memory in SAM-augmented recurrent units.

A. Grid-Based Memory Tensor

The SAM module is a grid-based memory network. As a prepossessing step, we partition the space into small grids. Then any trajectory $T = [X_1^c, ..., X_t^c, ...]$ can be mapped into a sequence $T^g = [X_1^g, ..., X_t^g, ...]$ where $X_t^g = (x_t^g, y_t^g)$ specifies the grid at $t - th$ position. Figure 2 shows the architecture of proposed SAM module. As shown, the core part of SAM is a memory tensor $\mathbf{M}$. The memory tensor $\mathbf{M}$ stores vector representations for all the grids in the space, which enable encoding and retrieving information for previously seen trajectories. Formally, assume the entire space is partitioned into $P \times Q$ grids, then dimensionality of the memory tensor is $P \times Q \times d$, where d is the hidden size of the recurrent unit. Each slice $(p, q, :)$ in $\mathbf{M}$ stores the embedding vector of grid (p, q) and all grid embeddings are initialized with 0 before training. As NEUTRAJ continuously processes the trajectories in the training data, the memory tensor $\mathbf{M}$ will be updated accordingly by memory-augmented recurrent units to encode the information in processed trajectories.



(a) SAM-augmented LSTM (b) SAM-augmented GRU

Fig. 3. Memory-augmented RNN units.

B. Memory-Augmented RNN Units

The SAM module allows for memorizing and retrieving information from processed trajectories. We leverage it to ameliorate standard RNN units. In this way, the RNN encoder captures the information from not only the current trajectory, but also those similar ones processed in history. In what follows, we show the architectures of two novel recurrent units that integrate the SAM module with two popular RNNs: LSTM and GRU.

1) SAM-Augmented LSTM: Figure 3(a) shows the architecture of SAM-augmented LSTM unit and the green parts are the novel SAM module. As shown, at each time step t, $X_t = (X_t^c, X_t^g)$ is fed to the unit which contains four gates: the forget gate $\mathbf{f_t}$, the input gate $\mathbf{i_t}$, the spatial gate $\mathbf{s_t}$ and the output gate $\mathbf{o_t}$. As in LSTM, these gates control the operations on the cell state $\mathbf{c_t}$ that stores core information of the recurrent unit. The recurrent step is performed as follows:

$$(\mathbf{f_t}, \mathbf{i_t}, \mathbf{s_t}, \mathbf{o_t}, \tilde{\mathbf{c_t}})^T = \mathbf{W_I} \cdot X_t^c + \mathbf{U_h} \cdot \mathbf{h_{t-1}} + \mathbf{b} \quad (1)$$

$$\hat{\mathbf{c_t}} = \sigma(\mathbf{f_t}) \cdot \mathbf{c_{t-1}} + \sigma(\mathbf{i_t}) \cdot \tanh(\tilde{\mathbf{c_t}}) \quad (2)$$

$$\mathbf{c_t} = \hat{\mathbf{c_t}} + \sigma(\mathbf{s_t}) \cdot read(\hat{\mathbf{c_t}}, X_t^g, \mathbf{M}) \quad (3)$$

$$\mathbf{h_t} = \sigma(\mathbf{o_t}) \cdot \tanh(\mathbf{c_t}) \quad (4)$$

$$write(\mathbf{c_t}, \mathbf{s_t}, X_t^g, \mathbf{M})) \quad (5)$$

where $\mathbf{W_I} \in 5d \times 2$, $\mathbf{U_h} \in 5d \times d$ and d is the hidden state size. All of the gates($\mathbf{f_t}, \mathbf{i_t}, \mathbf{s_t}, \mathbf{o_t}$), cell states($\tilde{\mathbf{c_t}}, \hat{\mathbf{c_t}}, \mathbf{c_t}$) and hidden states($\mathbf{h_t}, \mathbf{h_{t-1}}$) have the same shape: $d \times 1$.

In the above, the parameters $\{\mathbf{W_I}, \mathbf{U_h}, \mathbf{b}\}$ are the gate parameters of SAM-LSTM. To update the hidden state $\mathbf{h_t}$, the unit performs the following steps: (1) applying a linear transformation of the coordinate information X_t^c and the previous hidden state $\mathbf{h_{t-1}}$; (2) obtaining the intermediate cell state $\hat{\mathbf{c_t}}$ from the forget gate $\sigma(\mathbf{f_t})$ and the input gate $\sigma(\mathbf{i_t})$. The forget gate $\sigma(\mathbf{f_t})$ controls what part of information in previous cell state $\mathbf{c_{t-1}}$ will be forgot, while the input gate $\sigma(\mathbf{i_t})$ controls what part of information in current input $tanh(\tilde{\mathbf{c_t}})$ will be stored; (3) augmenting $\hat{\mathbf{c_t}}$ with historical information, which is calculated by the $read$ operation on the memory tensor $\mathbf{M}$, to derive update cell state $\mathbf{c_t}$. The spatial gate $\mathbf{s_t}$ controls the what part of spatial information in other processed trajectory will be added to generate $\mathbf{c_t}$; (4) the output gate $\mathbf{o_t}$ outputs the hidden state $\mathbf{h_t}$ and cell state $\mathbf{c_t}$ for the next step; (5) Finally, the unit updates the memory tensor $\mathbf{M}$. The detail of *read* and *write* operations are described in the next section.

2) SAM-Augmented GRU: Unlike LSTM, standard GRU does not have cell states. To cooperate SAM with GRU, we construct a pseudo cell state $\mathbf{c_t}$. It stores information for the current time step but does not output to the next time step. As shown in Figure 3(b), there are three gates in SAM-GRU: (1) the reset gate $\mathbf{r_t}$, (2) the update gate $\mathbf{u_t}$; and (3) the spatial gate $\mathbf{s_t}$. The SAM-GRU unit updates as follows:

$$(\mathbf{r_t}, \mathbf{u_t}, \mathbf{s_t})^T = \mathbf{W_I} \cdot X_t^c + \mathbf{U_h} \cdot \mathbf{h_{t-1}} + \mathbf{b} \quad (6)$$

$$\hat{\mathbf{c_t}} = \tanh(\mathbf{W_c} \cdot \mathrm{X_t^c} + \sigma(\mathbf{r_t}) \cdot \mathbf{U_c} \cdot \mathbf{h_{t-1}}) \quad (7)$$

$$\mathbf{c_t} = \hat{\mathbf{c_t}} + \sigma(\mathbf{s_t}) \cdot read(\hat{\mathbf{c_t}}, X_t^g, \mathbf{M}) \quad (8)$$

$$\mathbf{h_t} = (1 - \sigma(\mathbf{u_t})) \cdot \mathbf{c_t} + \sigma(\mathbf{u_t})) \cdot \mathbf{h_{t-1}} \quad (9)$$

$$write(\mathbf{c_t}, \mathbf{s_t}, X_t^g, \mathbf{M}) \quad (10)$$

where $\mathbf{W_I} \in 3d \times 2$, $\mathbf{U_h} \in 3d \times d$.

C. Attention-Based Reads and Writes

With the memory tensor $\mathbf{M}$, NEUTRAJ uses the attention mechanism to capture the information in processed trajectories: (1) the *read* operation retrieves relevant grid embeddings from the memory to augment encoding the current trajectory; and (2) the *write* operation attends to relevant grids and updates grid embeddings with the information of the current trajectory.

1) Spatial Memory Reader: The reader retrieves information from the memory and using the information to augment RNN-based trajectory encoding. At each step t, the reader takes two inputs: (1) the input grid X_t^g; and (2) the intermediate cell state $\hat{\mathbf{c_t}}$ of the RNN encoder. With these two inputs, the reader outputs a cell state-like vector $\mathbf{c_{his}}$, which augments $\hat{\mathbf{c_t}}$ with the influence of previously processed trajectories close to X_t^g in the current trajectory.

In order to compute $\mathbf{c_{his}}$, we design an attention network shown in Figure 2. The computation of $\mathbf{c_{his}}$ proceeds as follows. As $X_t^g = (x_t^g, y_t^g)$ is input, the attentional reader first looks up the grids that are spatially close to (x_t^g, y_t^g) with a scanning bandwidth w which controls how many adjacent girds can be read. Specifically, the reader uses the bandwidth w to perform a memory scan and identify the grids around (x_t^g, y_t^g): $scan(x_t^g) = [x_t^g - w, x_t^g + w]$; $scan(y_t^g) = [y_t^g - w, y_t^g + w]$. The read grid embeddings are stored in a matrix $\mathbf{G_t}$ with shape $(2w+1)^2 \times d$. After memory scan, the reader employs the attention network to transform the matrix $\mathbf{G_t}$ to an d-dimensional vector. The attention mechanism is performed as follows:

$$\mathbf{A} = \mathrm{softmax}(\mathbf{G_t} \cdot \hat{\mathbf{c_t}}); \quad \mathbf{mix} = \mathbf{G_t}^{\mathrm{T}} \cdot \mathbf{A};$$

$$\mathbf{c_t^{cat}} = [\hat{\mathbf{c_t}}, \mathbf{mix}]; \quad \mathbf{c_t^{his}} = \tanh(\mathbf{W_{his}} \cdot \mathbf{c_t^{cat}} + \mathbf{b_{his}})$$

where the matrix $\mathbf{W_{his}}$ and $\mathbf{b_{his}}$ are the parameters of the attention network. $\mathbf{A} \in (2w+1)^2 \times 1$ is the attention weight that reflects the similarity of the intermediate state $\hat{\mathbf{c_t}}$ over the historical grid embedding matrix $\mathbf{G_t}$. $\mathbf{mix} \in d \times 1$ is the weight sum of the $\mathbf{G_t}$ to concatenate to $\hat{\mathbf{c_t}}$. Finally, a fully connected layer transforms $\mathbf{c_t^{cat}}$ to fit the current state and generate the spatial attention cell state $\mathbf{c_t^{his}}$. Once $\mathbf{c_t^{his}}$ is generated, we can combine it with the intermediate cell state

$\hat{\mathbf{c_t}}$ to get the final cell state $\mathbf{c_t}$ by equation 3 and 8. Just like the gates in standard LSTM or GRU, the spatial gate $\mathbf{s_g}$ is decided by the current coordinate input $X_t^c = (x_t^c, y_t^c)$ and the previous hidden state $\mathbf{h_{t-1}}$. It controls what part of the spatial attention cell state is useful for the current trajectory.

2) Spatial Memory Writer: The grid embeddings in $\mathbf{M}$ should be updated during the training process. At each step, the writer directly performs sparse updating of all the corresponding entries in the memory $\mathbf{M}$ based on the $\mathbf{s_g}$:

$$\mathbf{M}(X_g)_{new} = \sigma(\mathbf{s_g}) \cdot \mathbf{c_t} + (1 - \sigma(\mathbf{s_g})) \cdot \mathbf{M}(X_g)_{old}.$$

Note that we are using the same spatial gate $\mathbf{s_g}$ for both the reader and writer. It is because $\mathbf{s_g}$ reflects not only the confidence level that $\mathbf{c_{his}}$ is useful for the current input, but also how much information in the current input is suitable for updating $\mathbf{M}(X_g)$. Another benefit of sharing the same gate is that it limits the number of parameters of our model.

V. Seed-Guided Neural Metric Learning

In this section, We first describe the metric learning procedures of NeuTraj and then present a weighted sampling and optimization method that learns the model from seed trajectories.

A. Metric Learning Procedures of NeuTraj

Figure 1 illustrates our NeuTraj model that uses neural networks to embed trajectories into d-dimensional space and approximates the similarity function. As shown, the core of NeuTraj is the spatial memory-augmented RNN encoder, which generates latent vector representation for any trajectory. For an input trajectory, the final hidden state of our RNN encoder is used as the trajectory representation.

Given a pool of seed trajectories $\mathfrak{S}$ and their distance matrix $\mathbf{D}$, NeuTraj normalizes $\mathbf{D}$ to a similarity matrix $\mathbf{S}$ and uses the $\mathbf{S}$ as guidance. For any two input trajectories T_i and T_j ($i, j \in [1, ..., N]$), the RNN encoder is able to project them to two d-dimensional vectors $\mathbf{E_i}$ and $\mathbf{E_j}$. Our goal is to learn the network parameters such that similarity between $\mathbf{E_i}$ and $\mathbf{E_j}$ is close to the original trajectory similarity $f(T_i, T_j)$. To this end, we design a similarity-preserving ranking objective in NeuTraj. Specifically, for any anchor trajectory T_a, we will sample a set $\mathscr{T}_a^s$ of similar neighbors for T_a, as well as a set $\mathscr{T}_a^d$ of dissimilar neighbors for T_a from the seed trajectories to form the similar and dissimilar pairs. With the sampled pairs, we design a weighted ranking objective, which encourages the network to learn a regression function for fitting the similarities to $\mathbf{S}$, as well as preserving the ranking order in both similar and dissimilar pairs. In what follows, we introduce our weighted sampling and optimization procedure.

B. Weighted Sampling and Optimization

Our NeuTraj model for metric learning is related to the classic Siamese network [5]. The Siamese network uses random sampling method to generate training pairs, and learns a regression function to fit the target measure. However, such a random sampling strategy implies all pairs have the same weight to the total loss. This assumption does not hold for trajectory metric learning as it ignores the spatial proximity between trajectories. Given one anchor trajectory, we need to focus on the most similar trajectories and the most dissimilar ones, because they are more discriminative than middle ones. Directly using random sampling and treating all pairs equally can lead to slow convergence and suboptimal accuracies.

To remedy the above problem, we propose a distance-weighted sampling and optimization strategy. We first transform the original distance matrix $\mathbf{D}$ into a normalized similarity matrix $\mathbf{S}$ as follows:

$$\mathbf{S}_{i,j} = \exp(-k \cdot \mathbf{D}_{i,j}) / \sum_{n=1}^{N} \exp(-k \cdot \mathbf{D}_{i,n})$$

where k is a tunable parameter controlling the similarity value distribution. The reason behind the transformation is that the distribution of raw distances often obey to power-law distributions and the magnitude of the distance can span a large range. The transformation is essentially a smoothing operation which brings the similarity values into the range $[0, 1]$ and smooths the distribution.

Inspired by [20], our distance-weighted sampling procedure works as follows. We take trajectories in the pool of N seeds as anchor trajectories sequentially. For one anchor trajectory T_a, we take the corresponding row from the similarity matrix $\mathbf{S}$ as the importance vector $\mathbf{I_a}$. With the entries in $\mathbf{I_a}$ as importance weights, we sample n distinct trajectories as similar samples: $\mathscr{T}_a^s = \{T_1^s, ..., T_n^s\}$. Conversely, we sample another n dissimilar samples $\mathscr{T}_a^d = \{T_1^d, ..., T_n^d\}$ using the entries in $\mathbf{1} - \mathbf{I_a}$ as importance weights. Then we rank the similar samples with the decrease of its similarity to T_a and rank dissimilar samples with the increase order. Finally, we obtain $2n$ pairs for T_a.

After sampling, we generate the trajectory embeddings and define the pair-wise similarities for the anchor trajectory over the similar and dissimilar pairs as follows:

$$\begin{aligned} \hat{\mathbf{S}}_a^s &= \hat{\mathbf{S}}(T_a, \mathscr{T}_a^s) = [g(T_a, T_1^s), ...g(T_a, T_n^s)] \\ \hat{\mathbf{S}}_a^d &= \hat{\mathbf{S}}(T_a, \mathscr{T}_a^d) = [g(T_a, T_1^d), ...g(T_a, T_n^d)] \end{aligned} \tag{11}$$

where $g(T_i, T_j) = \exp(-\text{Euclidean}(\mathbf{E_i}, \mathbf{E_j}))$ computes the similarity between two trajectory embeddings, and $\mathbf{E}$ is the embedding of the corresponding trajectory.

Coupled with weighted sampling, we propose a weighted ranking loss which is motivated by list-wise ranking [7] and Mean Reciprocal Rank [21]. Given a ranked list of n sampled trajectories, we set their ranking weights as $\mathbf{r} = (1, 1/2, ..., 1/l, ..., 1/n)$ and normalize the weights by $\sum_{l=1}^{n} r_l$. For the n similar pairs, their weights decrease with the ranking order, namely the most similar trajectory in $\mathscr{T}_a^s$ is regarded as the most important. Thus we define the loss for similar pairs of T_a as:

$$L_a^s = \sum_{l=1}^{n} r_l \cdot (g(T_a, T_l^s) - f(T_a, T_l^s))^2 \tag{12}$$

where $f(T_i, T_j)$ is the ground truth similarity of (T_i, T_j).

For dissimilar pairs, it is not reasonable to focus more on fitting the similarity value. Instead, we design a margin loss

to separate dissimilar trajectories from the anchor trajectory:

$$L_a^d = \sum_{l=1}^{n} r_l \cdot [\mathbf{ReLU}(g(T_a, T_l^d) - f(T_a, T_l^d))]^2 \quad (13)$$

The $\mathbf{ReLU}(x) = max(0, x)$ function defines the margin loss as follows: when $g(T_a, T_l^d) - f(T_a, T_l^d) < 0$, $L_a^d = 0$, meaning that dissimilar sample is faraway enough from the anchor trajectory in the embedding space; when $g(T_a, T_l^d) - f(T_a, T_l^d) > 0$, $L_a^d > 0$, the embeddings should be adjusted to enlarge the embedding-based distance of the dissimilar sample of the anchor. Finally, the loss for the given $\mathfrak{S}$ is the sum of the similar and dissimilar samples over all N seeds.

$$L_{\mathfrak{S}} = \sum_{a \in [1,\ldots,N]} (L_a^s + L_a^d)$$

Since all the modules and the loss functions are differentiable, all the parameters in NEUTRAJ can be learned in an end-to-end manner. In the training process, we update the parameters with back-propagation through time (BPTT) algorithm and employ Adam optimizer for stochastic optimization.

VI. COMPLEXITY ANALYSIS

The time complexity of NEUTRAJ for computing the similarity of a trajectories pair includes two parts: the embedding part and the distance computation part. For embedding, the computation is linear to the number of recurrent operations. In one time step, the higher-order term of complexity in classic recurrent units is $(m+1)*d^2$, where m is the number of gates, *e.g.*, $m = 3$ for LSTM. In SAM units, extra computation cost is involved by a new gate and spatial attention reader which complexity is also quadratic of d. For a pair of trajectories, d is a constant and the complexity of distance computation in the embedding space is a constant. The overall time complexity of NEUTRAJ is thus linear in the length of the trajectories.

For a trajectory database, the trajectories embeddings only need to be computed once. When new trajectory similarity search task is conducted, we generate the embedding of new trajectory and preform search based on the distance of embeddings. So the computation is linear with the size of search space and it can be further reduced if index is available for the trajectory database. This makes NEUTRAJ suitable for large dataset.

VII. EXPERIMENT

A. Experimental Settings

1) Datasets.: Our experiments are based on two public real trajectory datasets in two cities: Beijing and Porto. The first dataset [32], referred Geolife, consists of 17,621 trajectories of human mobility from 2007 to 2010. The second dataset [22] consists of over 1.7 millions of taxi trajectories from 2013 to 2014. To moderate the dimensions of $\mathbf{M}$, we choose trajectories in the center area of the city and discretize the area into 50m $\times$ 50m grids. Then, we remove the trajectories whose lengths are smaller than 10. After such prepossessing, we obtain 8203 trajectories in Geolife and 601,071 trajectories in Porto[1].

[1]Code and data available at https://github.com/yaodi833/NeuTraj

2) Experimental Protocol: To evaluate the performance of NEUTRAJ, we study top-k similarity search problem on both Geolife and Porto datasets and evaluate NEUTRAJ under four distance measures: the Fréchet distance, the Hausdorff distance, Edit distance with Real Plenty(ERP) and Dynamic Time Warping (DTW). The first three are metric, namely the distance is symmetric and satisfies the triangle inequality. We thus learn the models to approximate the metrics directly. However, the DTW distance is not a metric mainly because it not satisfies the triangle. Experiments on DTW explore the performance of NEUTRAJ on non-metric similarity measure.

The ground-truth of the problem is the exact top-k results based on the accurate similarity. For Geolife, we compute the accurate similarity of all trajectory pairs and random choose 20% trajectories as the seeds to train NEUTRAJ. Additionally, 10% trajectories are used for tuning parameters and 70% are used for testing. For Proto, due to the enormous trajectories, directly compute the exact similarity of all trajectory pairs is impractical. We random choose 10k trajectories to compute the similarity and follow the experimental protocol as the same as Geolife. The performance comparison, efficiency study and parameter sensitivity study of top-k similarity search are shown in VII-B VII-C and VII-D. In addition, based on the well-trained model on 10k Proto trajectories, we conduct a case study of entire Proto dataset(reported in VII-E) to show the efficiency and accuracy of NEUTRAJ on large dataset.

We also conduct a zero shot learning task to test whether NEUTRAJ works well on a city which has no available trajectories but just the road networks. Based on the road networks in Beijing [31], we simulate 6000 synthetic trajectories as the seeds for training and test NEUTRAJ with the real trajectories in Geolife. Result of zero short learning is presented in VII-F.

3) Compared Methods: For the studied four measures, we compare NEUTRAJ with seven baselines, which can be roughly divided into three categories:

- *Approximate algorithms*: Except ERP which has no approximate algorithm, each of the three measures has several approximate algorithms to fast compute them. We compare with the state-of-the-art approximate algorithms from [12], [4], and [1], respectively. We call these algorithms as **AP** in general for all the distance measures.
- *Siamese Network*: This category is a metric learning approach based on the Siamese network. We instantiate the Siamese network with both LSTM and GRU backbones, and denote them as **Siamese (LSTM)** and **Siamese (GRU)**, respectively.
- *Ablations:* Finally, we include two kinds of ablations of NEUTRAJ. (1) the weight sampling in NEUTRAJ is replaced by random sampling to test the effectiveness of distance-weighted ranking loss. We denote these two variants as **NT-NO-WS (LSTM)** and **NT-NO-WS (GRU)**, respectively. (2) we replace the SAM units with GRU and LSTM, denoted as **NT-NO-SAM (GRU)** and **NT-NO-SAM (LSTM)**, to test the effects of the proposed spatial attention memory mechanism.

TABLE II
PERFORMANCE COMPARISON FOR DIFFERENT METHODS ON FRÉCHET AND HAUSDORFF DISTANCES.

Note: HR is the hitting ratio; R10@50 is the top-50 recall for the top-10 ground truth; δ_{H10} is the distortion of average distance on the top 10 results; δ_{R10} is the distortion of average distance on the top 10 recall in top 50 result. The ground truth of top-10 average distance of Fréchet and Hausdorff distances are: $1044m$ and $730m$(Geolife); $935m$ and $679m$(Porto).

Data	Method	Fréchet				Hausdorff			
		HR@10	HR@50	R10@50	$\delta_{H10}/\delta_{R10}$	HR@10	HR@50	R10@50	$\delta_{H10}/\delta_{R10}$
Geolife	AP	0.2374	0.2542	0.5290	213m/ 87m	0.2967	0.3180	0.5363	217m/113m
	Siamese(LSTM)	0.4631	0.6032	0.8121	162m/ 34m	0.3820	0.5136	0.7640	179m/ 69m
	Siamese(GRU)	0.4922	0.6400	0.8368	145m/ 30m	0.3964	0.5458	0.7728	173m/ 64m
	NEUTRAJ (LSTM)	0.5609	0.7123	0.8781	74m/ 16m	0.4610	0.6172	0.8238	152m/ 42m
	NEUTRAJ (GRU)	**0.5872**	**0.7553**	**0.9033**	**65m/ 13m**	**0.4911**	**0.6507**	**0.8533**	**141m/ 21m**
	Ablation Experiments								
	NT-NO-WS (LSTM)	0.5136	0.6453	0.8296	139m/ 27m	0.4038	0.5893	0.7973	169m/ 55m
	NT-NO-WS (GRU)	0.5212	0.6823	0.8581	110m/ 25m	0.4283	0.6023	0.8173	163m/ 47m
	NT-NO-SAM (LSTM)	0.5342	0.6983	0.8698	117m/ 17m	0.4374	0.6107	0.8219	157m/ 46m
	NT-NO-SAM (GRU)	0.5518	0.7049	0.8726	87m/ 16m	0.4714	0.6229	0.8384	159m/ 34m
Porto	AP	0.2542	0.2851	0.5520	208m/ 79m	0.2832	0.2966	0.5620	201m/ 86m
	Siamese(LSTM)	0.5140	0.6302	0.8370	128m/ 27m	0.3834	0.4999	0.7760	165m/ 48m
	Siamese(GRU)	0.5329	0.6337	0.8522	116m/ 21m	0.4105	0.5309	0.7931	149m/ 35m
	NEUTRAJ (LSTM)	0.5930	0.7151	0.9081	61m/ 8m	0.4920	0.6213	0.8703	101m/ 15m
	NEUTRAJ (GRU)	**0.6021**	**0.7622**	**0.9140**	**58m/ 7m**	**0.5032**	**0.6451**	**0.8810**	**97m/ 13m**
	Ablation Experiments								
	NT-NO-WS (LSTM)	0.5690	0.6883	0.8981	92m/ 10m	0.4290	0.5628	0.8109	140m/ 33m
	NT-NO-WS (GRU)	0.5762	0.6989	0.9001	84m/ 9m	0.4319	0.5765	0.8217	136m/ 29m
	NT-NO-SAM (LSTM)	0.5754	0.7021	0.8971	82m/ 10m	0.4438	0.5831	0.8233	126m/ 27m
	NT-NO-SAM (GRU)	0.5832	0.7129	0.9026	75m/ 8m	0.4860	0.6152	0.8624	114m/ 18m

TABLE III
PERFORMANCE COMPARISON FOR DIFFERENT METHODS ON ERP AND DTW DISTANCES.

Data	Method	ERP			DTW		
		HR@10	HR@50	R10@50	HR@10	HR@50	R10@50
Geolife	AP	——	——	——	0.3870	0.4268	0.7139
	Siamese(LSTM)	0.6287	0.7363	0.8964	0.4080	0.5582	0.7872
	Siamese(GRU)	0.6340	0.7480	0.9082	0.4328	0.5896	0.8102
	NEUTRAJ (LSTM)	0.6932	0.8103	0.9521	0.4970	0.5961	**0.8521**
	NEUTRAJ (GRU)	**0.7113**	**0.8157**	**0.9630**	**0.5004**	0.6047	0.8481
	Ablation Experiments						
	NT-NO-WS (LSTM)	0.6680	0.7870	0.9386	0.4591	0.5600	0.7860
	NT-NO-WS (GRU)	0.6783	0.8023	0.9450	0.4763	0.5731	0.8101
	NT-NO-SAM (LSTM)	0.6710	0.7937	0.9491	0.4881	0.5825	0.8082
	NT-NO-SAM (GRU)	0.6893	0.8052	0.9510	0.4950	**0.6106**	0.8422
Porto	AP	——	——	——	0.3798	0.4160	0.7010
	Siamese(LSTM)	0.6782	0.7593	0.9243	0.4132	0.4804	0.7602
	Siamese(GRU)	0.6883	0.7629	0.9320	0.4379	0.5021	0.7801
	NEUTRAJ (LSTM)	0.7343	0.8267	0.9801	**0.4891**	**0.5610**	0.8198
	NEUTRAJ (GRU)	**0.7493**	**0.8399**	**0.9983**	0.4740	0.5527	**0.8249**
	Ablation Experiments						
	NT-NO-WS (LSTM)	0.7192	0.7920	0.9617	0.4330	0.5013	0.7919
	NT-NO-WS (GRU)	0.7138	0.8180	0.9782	0.4536	0.5198	0.8086
	NT-NO-SAM (LSTM)	0.7182	0.8111	0.9707	0.4738	0.5425	0.8148
	NT-NO-SAM (GRU)	0.7231	0.8193	0.9832	0.4804	0.5532	0.8240

4) Evaluation Metrics: We use three different metrics for performance evaluation. The first is the top-k hitting ratio, which examines the overlap percentage of the top-k results and the ground truth. We report the hitting ratio for both top-10 (HR@10) and top-50 searches (HR@50). The second is the top-50 recall for the top-10 ground truth (R10@50). This one evaluates how many of top 10 ground-truth trajectories are recovered by the top 50 lists produced by different methods. The third metric is the distortion of average distance for the top-10 results, denoted as δ_{H10} and δ_{R10}. δ_{H10} directly gets the top-10 trajectories from the results and δ_{R10} get the top-10 most similar trajectories from top-50 results. They measure the distortion of average exact distances between the query trajectory between the top-10 search results and the average exact distance. The smaller the distance is, the stronger a method performs.

5) Parameter Settings: The key parameters in NEUTRAJ include: (1) the embedding dimension d; (2) the scan width w of the attention memory reader. We have tuned d by the grid search in range $\{16, 32, 64, 128, 256\}$. In general, the performance increases with the d and gradually stabilizes when

it is large enough. For w, we tuned it in $\{0, 1, 2, 3, 4\}$ and found that it had an optimal value as $w = 2$ on both datasets. Finally, we set $d = 128$ and $w = 2$. In addition, we set the batch sizes as 20 and the sampling size n as 10. For the compared methods, we tuned their parameters to obtain the best performance in our datasets. We will also report the parameter study results shortly in Section VII-D.

B. Performance Comparison

Table II& III show the performance of different methods for the top-k similarity search task. As shown, on both datasets, NEUTRAJ significantly outperforms all the baseline methods in different metrics. Take the Fréchet distance as an example. Compared with state-of-the-art approximate algorithms (AP), the two variants of NEUTRAJ, *i.e.*, NEUTRAJ (LSTM) and NEUTRAJ (GRU) achieve more than two time higher hitting ratios, about 70% gain in R10@50, and about 69% reductions in average distance. Such huge improvements is impressive given the fact that NEUTRAJ does not rely on any hand-crafted heuristics but learn the similarity function automatically from seed trajectories. The superiority of NEUTRAJ over the Siamese network is also obvious in all the four metrics. While both methods employ neural metric learning for approximating the similarity function, NEUTRAJ has two advantages over the Siamese network. First, the weighted sampling and ranking loss can yield more distinguishing trajectories and training loss compared to Siamese network. Second, the spatial attention memory module models the correlation between spatially close trajectories, which is very beneficial for generating quality trajectory embedding.

Comparing NEUTRAJ with its ablations, one can further see the effectiveness of the two major modules in NEUTRAJ. Continue using the Fréchet distance on the Geolife dataset as an example. We observe: (1) by including the SAM module, NEUTRAJ improves HR@10 of NT-NO-WS from 0.49 to 0.52; and (2) by including the weighted sampling and optimization module, NEUTRAJ improves the HR@10 of NT-NO-WS from 0.52 to 0.58. The trends are similar for the Porto dataset and other three similarity measures.

The absolute values of HR@10 and HR@50 seem not very high on both datasets. The reason is that the trajectories in both datasets have a lot of near-duplicate instances. This can be observed from the value of δ_{H10}, which measures the spatial closeness for both the top-10 ground truth and the generated top-10 list.

C. Efficiency Study

In this subsection, we study the efficiency of NEUTRAJ. We first report its time cost for online similarity search, and then report the offline time cost for training the NEUTRAJ model. The experiments are conducted on a machine with Inter Xeon E5 @2.20GHz CPU and one Nvidia P100 GPU.

1) Online Similarity Computation: Table IV shows the time cost of NEUTRAJ for performing top-k similarity search with different sizes. Specifically, from the test set of Porto, we randomly sample four sub-corpora with sizes 1K, 5K, 10K, and 200K respectively. Then we use NEUTRAJ to perform top-50 similarity for each trajectory and re-rank the 50 trajectories by calculating their accurate distance. Table IV reports the average time cost for processing one query. We compare it with the BruteForce method that directly computes the exact distances following the definition, state-of-the-art approximate algorithms (AP), as well as the neural network based methods(NT-NO-SAM). We omit the results of NT-NO-WS and Siamese methods because the time cost of NT-NO-WS and Siamese methods are analogous to NEUTRAJ and NT-NO-SAM in online search procedure. As shown in Table IV, for all the four measures, NEUTRAJ provides 50x-1500x speedup over BruteForce and 2x -500x speedup over existing approximate algorithms. The speedup ratios are especially significant for large datasets. The time cost of ablation methods NT-NO-SAM are very close to NEUTRAJ because the embedding time difference of one search trajectory is small. Between the two variants of NEUTRAJ, the GRU version is faster than the LSTM one by involving fewer gates.

TABLE IV
TIME COST FOR ONLINE TOP-k SIMILARITY SEARCH.

Method	1k	5k	10k	200k
	Fréchet			
BruteForce	8.712s	41.876s	84.480s	1639.834s
AP [12]	1.840s	11.319s	23.107s	532.652s
NT-NO-SAM (LSTM)	0.461s	0.471s	0.489s	1.576s
NT-NO-SAM (GRU)	0.432s	**0.452s**	0.477s	1.554s
NEUTRAJ (LSTM)	0.461s	0.470s	0.490s	1.574s
NEUTRAJ (GRU)	**0.432s**	0.453s	**0.476s**	**1.553s**
	Hausdorff			
BruteForce	0.238s	1.416s	2.981s	51.642s
AP [4]	0.127s	0.154s	0.179s	3.426s
NT-NO-SAM (LSTM)	0.026s	0.046	0.072s	1.133s
NT-NO-SAM (GRU)	0.021s	0.041	0.065s	1.128s
NEUTRAJ (LSTM)	0.024s	0.047	0.073s	1.131s
NEUTRAJ (GRU)	**0.019s**	**0.041**	**0.064s**	**1.125s**
	ERP			
BruteForce	0.409s	1.982s	3.807s	73.054s
NT-NO-SAM (LSTM)	0.027s	0.046s	0.081s	1.154s
NT-NO-SAM (GRU)	**0.025s**	**0.039s**	0.072s	1.148s
NEUTRAJ (LSTM)	0.026s	0.047s	0.081s	1.152s
NEUTRAJ (GRU)	0.026s	0.040s	**0.071s**	**1.147s**
	DTW			
BruteForce	0.305s	1.482s	3.070s	59.054s
AP [1]	0.119s	0.142s	0.185s	4.021s
NT-NO-SAM (LSTM)	0.023s	0.044s	0.066s	1.028s
NT-NO-SAM (GRU)	0.019s	0.040s	**0.059s**	1.024s
NEUTRAJ (LSTM)	0.021s	0.043s	0.067s	1.027s
NEUTRAJ (GRU)	**0.019s**	**0.038s**	0.060s	**1.023s**

2) Time Cost of Offline Training and Embedding: **Offline Training Time**. Table V only shows the comparison of offline training time on the Porto dataset under Fréchet distance. This result is similar for other similarity measures. For 2,000 training trajectories, NEUTRAJ converges within 20 epochs. The training time of one epoch is around 5 min for NEUTRAJ, thus the entire training time of NEUTRAJ is less then 2 hours. As comparison, the Siamese network takes more than 60 epochs to converge, which is about 3X slower than NEUTRAJ. We also compare the convergence rate of NEUTRAJ and NT-NO-

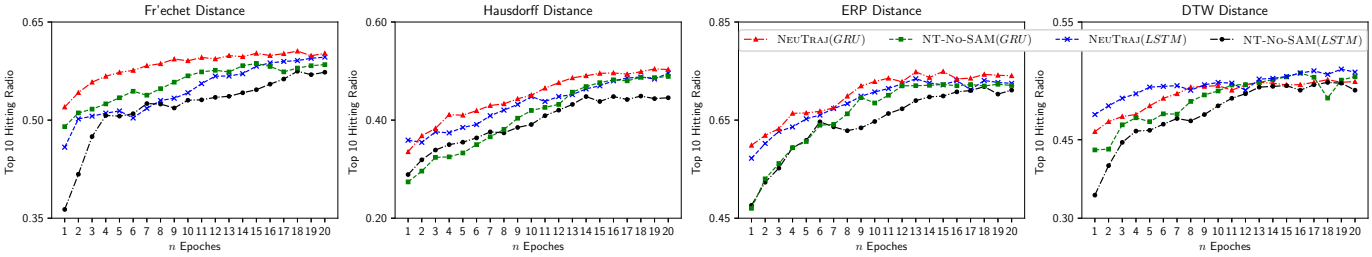


Fig. 4. The convergence curve of NEUTRAJ and NT-NO-SAM on Fréchet, Hausdorff, ERP and DTW with respect to 20 epochs.

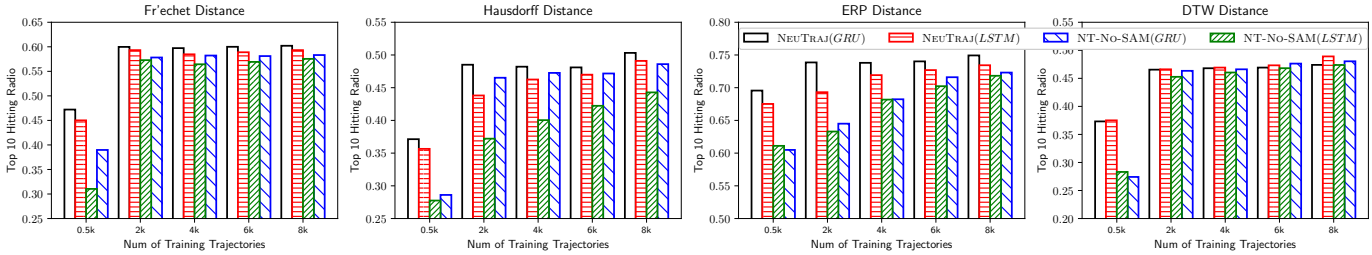


Fig. 5. HR@10 of NEUTRAJ and NT-NO-SAM on Fréchet, Hausdorff and DTW with varying training data size.

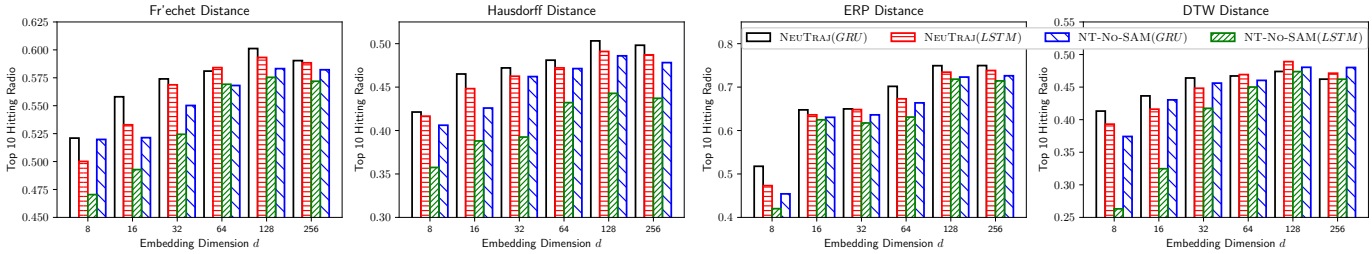


Fig. 6. HR@10 of NEUTRAJ and NT-NO-SAM on Fréchet, Hausdorff and DTW with varying embedding size d.

TABLE V
TIME COST FOR OFFLINE MODEL TRAINING.

Methods	Model Train			Embed
	t_{epoch}	$\#_{epoch}$	t_{total}	200k
Siamese(LSTM)	164s	71	11644s	411s
Siamese(GRU)	153s	65	9945s	386s
NEUTRAJ (LSTM)	285s	15	5130s	639s
NEUTRAJ (GRU)	276s	14	3864s	483s
Ablations Experiments				
NT-NO-SAM (LSTM)	168s	15	2520s	412s
NT-NO-SAM (GRU)	159s	13	**2067s**	**384**s
NT-NO-WS (LSTM)	283s	20	5660s	636s
NT-NO-WS (GRU)	271s	18	4878s	496s

SAM in Figure 4 and observe that: (1)NEUTRAJ has higher convergence rate than NT-NO-SAM because SAM module captures useful information from processed trajectories; (2)the GRU version of NEUTRAJ is stronger than the LSTM version. The reason is GRU-based NEUTRAJ has less parameters than LSTM, which tends to achieve better convergence.

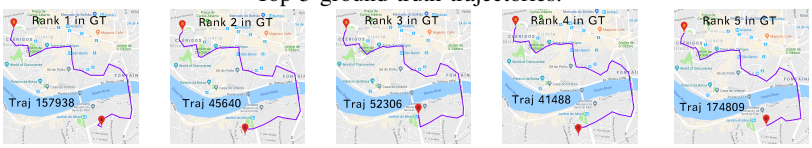
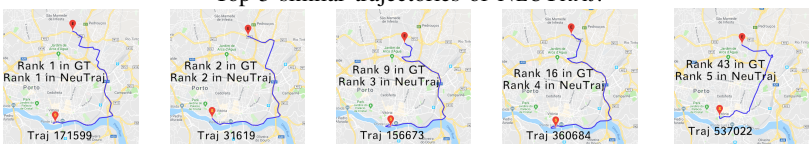
Offline Embedding Time. Another part of offline training is the embedding procedure which generates the embeddings of trajectories by using the well-trained model. In this experiment, we set the embedding batch size as 2000 and report the time cost of embedding 200k trajectories. The results of all neutral network based methods are shown in Table V. We can easily find that SAM units based methods(NEUTRAJ, NT-NO-WS) are little slower than standard units based method(NT-NO-SAM, Siamese). The reason is SAM units need more calculation for finding useful information in memory tensor. For methods using the same kind of units, GRU based methods are faster than LSTM methods because GRU has fewer gates than LSTM and needs fewer matrix multiply operations.

D. Parameter Sensitivity Study

In this subsection, we evaluate the model sensitivity on three parameters: the training data size, the scan width w, and the embedding dimension d. Due to the space limit, we only shows the experimental result on the Porto datasets.

1) The sensitivity of training data size: The first question we are interested in is how does the number of seed trajectories affect the performance of NEUTRAJ? To answer this question, we vary the number of seed trajectories and evaluate the performance of NEUTRAJ along with the ablations NT-NO-SAM. Figure 5 shows the results for the four measures as the training data size varies from 500 to 8000 on Porto. As shown, the performance of NEUTRAJ becomes relatively stable when there are more than 2000 training trajectories. Comparing the performance across different methods, SAM-based models outperform the classic LSTM and GRU models. Another interesting fact is that NEUTRAJ is more robust than

TABLE VI
CASE STUDIES ON ENTIRE PORTO DATASET.

Searching Trajs	Comparison of top-5 similar trajectories between ground truth(GT) and NEUTRAJ
Result of T_{91} HR@10: 0.7 HR@50: 0.78 H10@R50: 0.9  δ_{H5} = 4m δ_{H10} = 4m δ_{R10} = 2m	Top-5 ground truth trajectories.  Top-5 similar trajectories of NEUTRAJ. 
Result of T_{65} HR@10: 0.4 HR@50: 0.52 H10@R50: 0.8  δ_{H5} = 296m δ_{H10} = 236m δ_{R10} = 62m	Top-5 ground truth trajectories.  Top-5 similar trajectories of NEUTRAJ. 

NT-NO-SAM with sparse training data. As shown, when the number of training trajectories is only 500, the performance gaps between NEUTRAJ and NT-NO-SAM are particularly large. This is because SAM employs a memory tensor to memorize useful information from processed trajectories.

2) The sensitivity of embedding dimension d: We proceed to study the effect of the embedding dimension d on the performance of NEUTRAJ. Figure 6 illustrates HR@10 as d varies from 8 to 256. As shown, the performance of NEUTRAJ and its variants first increases and then drops slightly. The reason is the parameter d controls the complexity of NEUTRAJ. When d increases, the model enjoys more expressive power to capture the intrinsic structures of trajectories. However, when d is too large, the model can suffer from overfitting due to the limited size of training data.

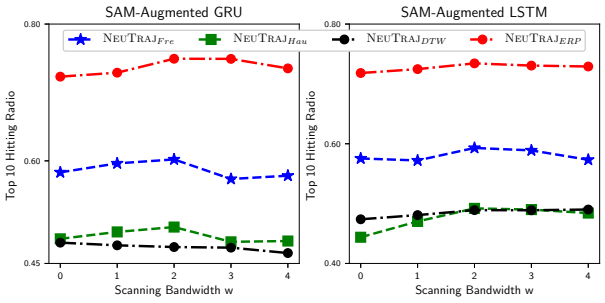


Fig. 7. HR@10 of NEUTRAJ with varying w.

3) The sensitivity of scan width w: Finally, the scan width w in the SAM module is a key parameter that controls the exploration spread for historical trajectories. As shown in Figure 7, with the increase of w, the HR@10 for all methods first increase and then slight drop. This phenomenon is attributed to two reasons: (1) as w increases, more information tends to be accessed by the SAM reader, which is useful for encoding the current trajectory in the initial stage; (2) when w is too large, the information of some non-relevant trajectories will be inevitably incorporated. Even if the attention mechanism in NEUTRAJ can help reduce this effect, the performance of the model can still be harmed.

E. Case Studies

In this case, we use the entire Porto dataset to perform several case studies to intuitively examine the top-k search results of our model. For this purpose, we randomly choose several trajectories and retrieve their top-5 neighbors under the Fréchet distance. Due to the space limit, Table VI only shows the results of two representative trajectories: T_{91} and T_{65} . For each query trajectory, we plot both the top-5 ground truth trajectories as well as the top-5 trajectories retrieved by NEUTRAJ.

Examining the result in Table VI, we can find that NEUTRAJ is quite effective on both short(T_{91}) and long(T_{65}) trajectory: the results returned by NEUTRAJ match the ground truth very well, and the distortions of top-5 average distance δ_{H5} are very small. Moreover, by training with the weighted ranking loss, NEUTRAJ preserves the ranking order of trajectories.

F. Evaluation on Zero-Shot Learning

In the final set of experiments, we are interested in applying NEUTRAJ for zero-shot learning scenarios. Specifically, the NEUTRAJ model relies on a real trajectory database and sampling seeds from the database as guidance. It is interesting

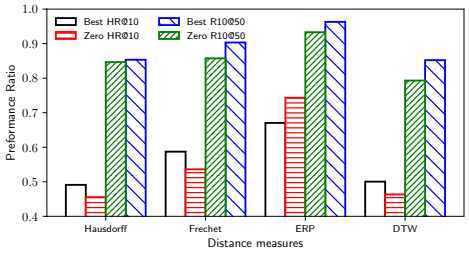


Fig. 8. Illustration of zero-shot learning results on Geolife dataset. $Best$ is the best performance NEUTRAJ can achieve on real dataset and $Zero$ is the performance trained with synthetic data.

to examine the question: how does NEUTRAJ perform if there are no real seed trajectories, yet we still want to compute the similarity for an ad-hoc pair of trajectories? For this purpose, we extend NEUTRAJ for the zero-shot learning scenario. We assume no trajectory databases are given but only a road network of the target area. Then we generate a bunch of simulated trajectories as our seeds and train the NEUTRAJ model for computing the similarity for a pair of real trajectories. Based on the road network in Beijing [31], we generate 6,000 synthetic trajectories by employing random walk on road node graph and interpolating coordinates between the nodes. Then we take the synthetic trajectories as the seeds to train NEUTRAJ, and test it with the real trajectories from Geolife. Figure 8 shows the HR@10 and R10@50 of NEUTRAJ.

Impressively, even using synthetic trajectories and their distances as guidance, NEUTRAJ can still achieve around 0.8 recall for all the four metrics. Such a phenomenon indicates that NEUTRAJ can be applied to scenarios even when no real trajectory databases are available.

VIII. CONCLUSION

We proposed a seed-guided neural metric learning approach NEUTRAJ that is fast, accurate, generic and elastic for trajectory similarity computation. Its novelty lies on two aspects: (1) a spatial attention memory (SAM) module that can augment existing RNN architectures to capture the correlations between trajectories; and (2) a distance-weighted ranking loss that effectively leverages seed information to learn quality trajectory embeddings. Experiments on two real-life datasets have shown that NEUTRAJ can effectively accelerate on various distance measures, while producing more accurate function approximations over state-of-the-art baselines.

REFERENCES

[1] P. K. Agarwal, K. Fox, J. Pan, and R. Ying, "Approximating dynamic time warping and edit distance for a pair of point sequences," in *SoCG'16*, 2016, pp. 6:1–6:16.

[2] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *Int. J. Comput. Geometry Appl.*, vol. 5, pp. 75–91, 1995.

[3] S. Atev, G. Miller, and N. P. Papanikolopoulos, "Clustering of vehicle trajectories," *IEEE Trans. Intelligent Transportation Systems*, vol. 11, no. 3, pp. 647–657, 2010.

[4] A. Backurs and A. Sidiropoulos, "Constant-distortion embeddings of hausdorff metrics into constant-dimensional l_p spaces," in *APPROX/RANDOM'16*, 2016, pp. 1:1–1:15.

[5] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a siamese time delay neural network," in *NIPS'93*, 1993, pp. 737–744.

[6] K. Buchin, M. Buchin, D. Duran, B. T. Fasy, R. Jacobs, V. Sacristán, R. I. Silveira, F. Staals, and C. Wenk, "Clustering trajectories for map construction," in *SIGSPATIAL'17*, 2017, pp. 14:1–14:10.

[7] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *ICML'07*, 2007, pp. 129–136.

[8] S. Chandar, S. Ahn, H. Larochelle, P. Vincent, G. Tesauro, and Y. Bengio, "Hierarchical memory networks," *arXiv:1605.07427*, 2016.

[9] L. Chen and R. T. Ng, "On the marriage of lp-norms and edit distance," in *VLDB'04*, 2004, pp. 792–803.

[10] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD'05*, 2005, pp. 491–502.

[11] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie, "Searching trajectories by locations: an efficiency study," in *SIGMOD'10*, 2010, pp. 255–266.

[12] A. Driemel and F. Silvestri, "Locality-sensitive hashing of curves," in *SoCG'17*, 2017, pp. 37:1–37:16.

[13] M. Farach-Colton and P. Indyk, "Approximate nearest neighbor algorithms for hausdorff metrics via embeddings," in *FOCS'99*, 1999, pp. 171–180.

[14] M. G. Gowanlock and H. Casanova, "Distance threshold similarity searches: Efficient trajectory indexing on the GPU," *IEEE TPDS.*, vol. 27, no. 9, pp. 2533–2545, 2016.

[15] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv:1410.5401*, 2014.

[16] M. Kiermeier and M. Werner, "Similarity search for spatial trajectories using online lower bounding DTW and presorting strategies," in *TIME'17*, 2017, pp. 18:1–18:15.

[17] R. Laxhammar and G. Falkman, "Online learning and sequential anomaly detection in trajectories," *IEEE TPAMI.*, vol. 36, no. 6, pp. 1158–1173, 2014.

[18] T. Lee and S. Lee, "OMT: overlap minimizing top-down bulk loading algorithm for r-tree," in *CAiSE'03*, 2003.

[19] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *ICDE'18*, 2018.

[20] R. Manmatha, C. Wu, A. J. Smola, and P. Krähenbühl, "Sampling matters in deep embedding learning," in *ICCV'17*, 2017, pp. 2859–2867.

[21] B. McFee and G. R. G. Lanckriet, "Metric learning to rank," in *ICML'10*, 2010, pp. 775–782.

[22] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Time-evolving O-D matrix estimation using high-speed GPS data streams," *Expert Syst. Appl.*, vol. 44, pp. 275–288, 2016.

[23] W. Pei, D. M. J. Tax, and L. van der Maaten, "Modeling time series similarity with siamese recurrent networks," *arXiv*, vol. abs/1603.04713, 2016. [Online]. Available: http://arxiv.org/abs/1603.04713

[24] Q. Qian, R. Jin, S. Zhu, and Y. Lin, "Fine-grained visual categorization via multi-stage metric learning," in *CVPR'15*, 2015, pp. 3716–3724.

[25] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *KDD '12*, 2012, pp. 262–270.

[26] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis, "Trajectory similarity join in spatial networks," *PVLDB*, vol. 10, no. 11, pp. 1178–1189, 2017.

[27] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *NIPS'15*, 2015, pp. 2440–2448.

[28] J. Weston, S. Chopra, and A. Bordes, "Memory networks," *arXiv*, vol. abs/1410.3916, 2014.

[29] D. Xie, F. Li, and J. M. Phillips, "Distributed trajectory similarity search," *PVLDB'17*, vol. 10, no. 11, pp. 1478–1489, 2017.

[30] B. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *ICDE'98*, 1998, pp. 201–208.

[31] X. Zhan, S. V. Ukkusuri, and P. S. C. Rao, "Dynamics of functional failures and recovery in complex road networks," *Physical Review E*, vol. 96, no. 5, p. 052301, 2017.

[32] Y. Zheng, X. Xie, and W. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.