Deep Representation Learning for Trajectory Similarity Computation

Xiucheng Li†, Kaiqi Zhao†, Gao Cong†, Christian S. Jensen‡, Wei Wei°

† *Nanyang Technological Univeristy,* ‡ *Aalborg University,* ° *Huazhong University of Science and Technology*

{xli055@e., kzhao002@e., gaocong@}ntu.edu.sg, csj@cs.aau.dk, weiw@hust.edu.cn

***Abstract*—Trajectory similarity computation is fundamental functionality with many applications such as animal migration pattern studies and vehicle trajectory mining to identify popular routes and similar drivers. While a trajectory is a continuous curve in some spatial domain, e.g., 2D Euclidean space, trajectories are often represented by point sequences. Existing approaches that compute similarity based on point matching suffer from the problem that they treat two different point sequences differently even when the sequences represent the same trajectory. This is particularly a problem when the point sequences are non-uniform, have low sampling rates, and have noisy points. We propose the first deep learning approach to learning representations of trajectories that is robust to low data quality, thus supporting accurate and efficient trajectory similarity computation and search. Experiments show that our method is capable of higher accuracy and is at least one order of magnitude faster than the state-of-the-art methods for k-nearest trajectory search.**

I. Introduction

With the proliferation of GPS-enabled devices, trajectory data is being generated at an unprecedented speed. A trajectory is typically represented as a sequence of discrete locations, or sample points, which describes the underlying route of a moving object over time. Quantifying the similarity between two trajectories is a fundamental research problem and is a foundation for many trajectory based applications, such as tracking migration patterns of animals [1], mining hot routes in cities [2], trajectory clustering [3] and moving group discovery [4], [5]. The importance of measuring trajectory similarity has also been recognized by researchers, and a number of classic methods have been proposed, such as dynamic time wrapping (DTW) [6], longest common subsequence (LCSS) [7], edit distance with real penalty (ERP) [8], and edit distance on real sequences (EDR) [9].

These existing methods usually try to form a pairwise matching the sample points of two trajectories and identify the best alignment using dynamic programming. More specifically, these pairwise point-matching methods implicitly partition the space into cells according to a threshold ϵ and match two points if they fall into the same cell. Dynamic programming is then used to find an alignment that minimizes a matching cost.

The pairwise point-matching methods for computing trajectory similarity often suffer in three scenarios: the first is when the sampling rates of trajectories are non-uniform. Sampling rates often vary across devices due to different device settings, battery constraints, and communication failures. Even for the same device, the sampling rates may vary. For example, a taxi driver may alter the default device sampling rate to reduce the power consumption [10]. This may result in sampling points alternating between sparse and dense episodes. This poses challenges to the existing methods—if two trajectories represent the same underlying route, but are generated at different sampling rates, it is difficult for these methods to identify them as similar trajectories. The second scenario where it is challenging to align the sample points of similar trajectories is when the sampling rate is low. For example, the sampling rates for trajectories generated from online "check-ins" (e.g., Foursquare, Facebook), geo-tagged tweets, geo-tagged photo albums, and call detail records are low and inherently non-uniform [11]. Third, the performance of these methods may be degraded when the sample points are noisy. Such noise may occur in GPS points when moving in urban canyons.

Moreover, the existing methods rely on local point matching and identify an optimal alignment using dynamic programming, which leads to quadratic computational complexity $O(n^2)$, where n is the mean length of the trajectories. We argue that a good trajectory similarity measure should be both accurate and scalable to large datasets.

We illustrate the problems caused by non-uniform and low sampling rates with two examples. First, consider the two trajectories $T_a = [a_1, a_2, a_3]$ and $T_b = [b_1, \ldots, b_6]$ in Figure 1a that are sampled from the same underlying route with different sampling rate. Assuming that the cell threshold ϵ is 1, a pairwise point-matching method such as EDR will match (a_1, b_1) and (a_3, b_6) while the remaining points will be unmatched, which yields a cost (edit distance) of 5. Although the two trajectories represent the same underlying route, they differ when compared using the pairwise point-matching methods. Second, to see the challenge of low sampling rates, consider the trajectory in Figure 1b consists of sample points $[a_1, \ldots, a_5]$. Due to the low sampling rate in the middle part of the trajectory, the pairwise point-matching methods may falsely match the two trajectories because four point pairs are matched.

The essential problem underlying the two examples is how to infer the true route from the observed trajectory points in the presence of non-uniform sampling rates, low sampling rates, and noise. As shown in Figure 1b, it is hard to tell which route, e.g., R_A, R_B is represented by the set of sample points $[a_1, \ldots, a_5]$. Recent studies [10], [12] reveal that the transiting

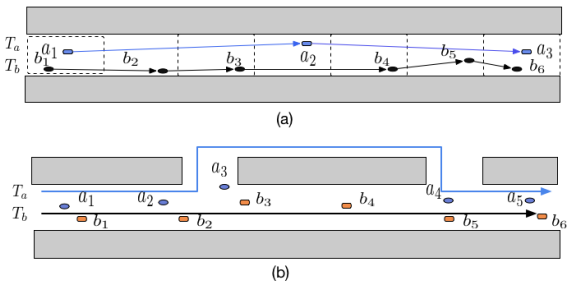

Fig. 1: Examples of the challenge in quantifying trajectory similarity.

patterns between certain locations are often highly skewed, i.e., some routes are more likely to be traveled than others, and these transition patterns, which are accumulated in the spatiotemporal databases [10], [12], [13], hold the potential to help quantify trajectory similarity. To harness such transition patterns, Su et al. [13] proposed an Anchor Points based Method (APM). APM first learns the transfer relationships among a fixed set of spatial objects (such as Points of Interest), called anchor points, from dense (i.e., with high sampling rate) historical trajectories by using Hidden Markov Models (HMM). Then sparse trajectories are calibrated to the anchor points, such that the existing pairwise point-matching methods, e.g., DTW, EDR, LCSS, can be employed to compute similarities of trajectories more accurately. APM requires the availability of a large amount of POIs and suffers from the limitations inherent in HMM like requiring explicit dependency assumptions to make inferences tractable [14]. Moreover, even after being trained on a historical dataset, APM still cannot reduce the $O(n^2)$ time complexity of the pairwise point-matching methods.

In this paper, we propose a novel approach, called t2vec (**t**rajectory to **vec**tor), to inferring and representing the underlying route information of a trajectory based on deep representation learning. The learned representation is designed to be robust to non-uniform and low sampling rates, and noisy sample points for trajectory similarity computation. This is achieved by taking advantage of the archived historical trajectory data and a new deep learning framework. With the learned representation, it only takes a linear time $O(n + |\mathrm{v}|)$ ($|\mathrm{v}|$ is the length of vector v) to compute the similarity between two trajectories, while all the existing approaches take $O(n^2)$ time. To the best of our knowledge, this is the first deep learning based solution for computing the similarity of trajectories.

To learn trajectory representations, it is natural to consider the use of Recurrent Neural Networks (RNNs), which are able to embed a sequence into a vector. However, simply applying RNNs to embed trajectories is impractical. First, the representation obtained using RNNs is unable to reveal the most likely true route of a trajectory when uncertainty arises due to low sampling rates or noise. Second, the existing loss functions used to train RNNs fail to consider spatial proximity, which is inherent in spatial data. Thus, they cannot guide the model to learn consistent representations for trajectories generated by the same route. To overcome the first challenge, we propose a sequence-to-sequence (seq2seq) based model to maximize the probability of recovering the true route of trajectory. To contend with the second challenge, we design a spatial proximity aware loss function and a cell pretraining algorithm that encourage the model to learn consistent representations for trajectories generated from the same route. We also propose an approximate loss function using Noise Contrastive Estimation [15] to boost the training speed. Overall, the paper makes the following contributions:

- We propose a seq2seq-based model to learn trajectory representations, for the fundamental research problem of trajectory similarity computation. The trajectory similarity based on the learned representations is robust to non-uniform, low sampling rates and noisy sample points. Our solution computes the similarity of two trajectories in linear time.
- For the purpose of learning consistent representations, we develop a new spatial proximity aware loss function and a cell representation learning approach that incorporate the spatial proximity into the deep learning model. To further speed up training, we propose an approximate loss function based on Noise Contrastive Estimation.
- We conduct extensive experiments on two real-world trajectory datasets that offer evidence that the proposed method is capable of outperforming the existing trajectory similarity measure techniques in terms of both accuracy and efficiency.

The rest of the paper is organized as follows. In Section II, we discuss the related work. The problem definition and preliminaries are given in Section III. Section IV presents the details of our method. The experimental results are presented in Section V. Finally, we summarize the paper and discuss future research directions in Section VI.

II. Related Work

We briefly review the related work on trajectory similarity computation and deep representation learning.

A. Trajectory similarity computation

Computing the similarity between two trajectories is fundamental functionality in many spatiotemporal data analysis tasks. Not surprisingly, the problem of accurately and efficiently measuring the similarity of trajectories has been studied extensively [6]–[9]. DTW [6] was a first attempt at tackling the local time shift issue for computing trajectory similarity. ERP [8], EDR [9], DISSIM [16], and the model-driven approach MA [17] were developed to further improve the ability of capturing the spatial semantics in trajectories. Wang et al. [18] studied the effectiveness of these similarity methods according to their robustness to noise, varying sampling rates, and shifting. All of these methods focus on identifying the optimal alignment based on sample point matching, and thus they are inherently sensitive to variation in the sampling rates. To solve this issue, APM [13] and EDwP [11] are proposed.

As discussed in the introduction, APM solves this issue by learning transition patterns of anchor points from historical trajectories. To compute the similarity of two trajectories, EDwP computes the cheapest set of replacement and insertion operations using linear interpolation to make them identical. Our solution is very different from APM and EDwP in that we aim to learn a vector that represents a trajectory and to then compute similarity using the new representation. In experiments, we compare with EDwP and not APM for two reasons: i) The implementation of APM partly requires an abundance of POIs which are not available in our datasets; ii) the more recent EDwP has been reported to perform better in similarity analysis of trajectories with non-uniform and low sampling rates. Our work is also related to the inference of hidden routes from partial observations. Zheng et al. [10] first studied the problem, and Banerjee et al. [12] further explored it using Bayesian posterior inference to estimate the top-k most likely routes. Our work differs from these two in that our ultimate goal is to learn representations of the trajectories rather than solely inferring the most possible routes.

Moreover, all the aforementioned existing measures for trajectory similarity are based on the dynamic programming technique to identify the optimal alignment which leads to $O(n^2)$ computation complexity. Given the complexity, it will be computationally expensive if we want to apply these similarity measures to cluster a large trajectory database. In contrast, our method has a linear time complexity $O(n + |\mathrm{v}|)$ to measure the similarity of two trajectories, which is able to support analysis on big trajectory data, such as clustering trajectories. We can also offer near-instantaneous response times that support interactive use, while the competition cannot.

B. Representation learning

Learning representations for specific tasks has been a longstanding open problem in machine learning. Recently, inspired by the success of word2vec [19], the idea of learning general representation has been extended to paragraphs [20], networks [21], [22], etc. To capture the sequential order information emerging in the sequence processing tasks, Recurrent Neural Networks (RNNs) based encoder-decoder models have been developed, such as sequence to sequence learning [23]–[25], and skip-thought vectors [26]. Our model is based on the general sequence encoder-decoder framework. However, these existing sequence encoder-decoder models were initially proposed for natural language processing to deal with discrete tokens (i.e., words, punctuations). Our scenarios is different in that the tokens inherently share the spatial proximity relation. Our model therefore differs from the above sequence encoder-decoders in two ways: i) we design a spatial proximity aware loss function and a cell representation pretraining approach to incorporate the spatial proximity into the deep representation model, and ii) we also propose an approximate loss function based on Noise Contrastive Estimation to accelerate the training.

III. Definitions and Preliminaries

In this section, we present definitions and preliminaries essential to understand the problem addressed and the sequence encoder-decoder model used in our solution. For ease of reference, frequently used notation is given in Table I.

TABLE I: Frequently used notation.

Symbol	Definition
T (or T_a)	Trajectory
R	Underlying route
x	A sequence of tokens
x_t	Token at position t
$x_{1:t}$	A sequence of tokens from position 1 to t
$\|T\|$ (or $\|x\|$)	Length of T (or x)
v	Embedded vector
h_t	Hidden state (vector)
V	Vocabulary
r_1 (r_2)	Dropping rate (distorting rate)

A. Definitions

We next define the notions of *underlying route* and *trajectory*.

Definition 1. *(UNDERLYING ROUTE) An underlying route of a moving object is a continuous spatial curve (e.g., in the longitude-latitude domain), indicating the exact path taken by the object.*

The underlying route is only a theoretical concept as location acquisition techniques do not record moving locations continuously.

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

In practice, an underlying route can be represented by enormous trajectories, depending on the specifics of the moving objects and the sampling strategies used. Each generated trajectory can be considered as a representative of an underlying route. In the rest of this paper, a trajectory is also referred to as trip, depending on the context.

Problem statement. Given a collection of historical trajectories, we aim to learn a representation $\mathrm{v} \in \mathbb{R}^{\mathsf{n}}$ (n is the dimension of a Euclidean space) for each trajectory T such that the representation can reflect the underlying route of the trajectory for computing trajectory similarity. The similarity of two trajectories based on the learned representations must be robust to non-uniform, low sampling rates and noisy sample points.

Our proposed method for solving the problem is based on deep representation learning techniques. Specifically, we adapt the sequence encoder-decoder framework for the first time to compute trajectory similarity (the motivation for adapting that particular framework is explained in Section IV-A).

B. Preliminaries of sequence encoder-decoders

We briefly present the sequence encoder-decoder framework. Consider two sequences $x = \langle x_t \rangle_{t=1}^{|x|}$ and $y = \langle y_t \rangle_{t=1}^{|y|}$

where each x_t and y_t denotes token (e.g., a word or punctuation mark in a natural language sentence) and $|x|$ and $|y|$ represent lengths. We next illustrate how to build the conditional probability $\mathbb{P}(y|x)$ in the framework.

The sequence encoder-decoder model has two main components—an encoder and a decoder, as depicted in Figure 2. The encoder is used to encode sequence x into a fixed-dimensional vector $\mathtt{v}$ that preserves the sequential information in x, and then the decoder decodes out sequence y conditioned on the encoded representation $\mathtt{v}$. Since Recurrent Neural Networks (RNNs) [27], [28] accept input in the form of real-valued vectors, a token embedding layer is added to embed the discrete token in a vector. The token embedding layer is form of a feed forward neural network [29].

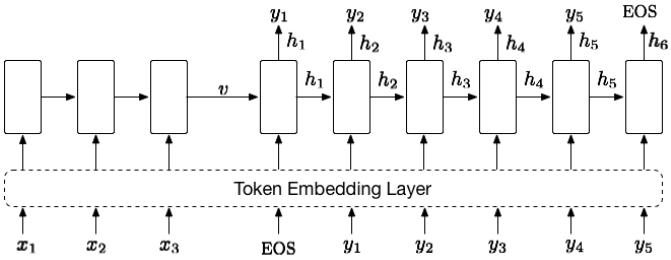

Fig. 2: Sequence encoder-decoder model. The model reads the sequence x and outputs the sequence y, where EOS is a special token indicating the end of a sequence and $\mathtt{v}$ is the representation of x. The hidden state h_t captures the sequential information in $[x, y_1, y_2, \ldots, y_{t-1}]$.

To see how the model builds the probability $\mathbb{P}(y|x)$, we first rewrite it by the chain rule, i.e.,

$$\mathbb{P}(y_1, \ldots, y_{|y|}|x_1, \ldots, x_{|x|}) = \mathbb{P}(y_1|x) \prod_{t=2}^{|y|} \mathbb{P}(y_t|y_{1:t-1}, x),$$

where $y_{1:t-1}$ represents $y_1, y_2, \ldots, y_{t-1}$. The encoder reads and encodes sequence x into the fixed-dimensional vector $\mathtt{v}$. Since $\mathtt{v}$ encodes the sequential information in x, we have

$$\mathbb{P}(y_t|y_{1:t-1}, x) = \mathbb{P}(y_t|y_{1:t-1}, \mathtt{v})$$

The decoder builds the probability $\mathbb{P}(y_t|y_{1:t-1}, \mathtt{v})$ at every position t by squashing $\mathtt{v}$ and $y_{1:t-1}$ into the hidden state h_t, which is simply a forward computation. More specifically, h_t is computed from the output of the previous position h_{t-1} and token y_{t-1}, as follows:

$$h_t = \begin{cases} f(\mathtt{v}, \text{EOS}) & t = 1 \\ f(h_{t-1}, y_{t-1}) & t \geq 2, \end{cases} \tag{1}$$

where $f(\cdot, \cdot)$ indicates the RNNs forward computation and EOS is the special token that signals the end of a sequence, which is necessary in order to support variable-length sequences [24].

Note that we recursively compute $h_t = f(h_{t-1}, y_{t-1})$: h_{t-1} encodes the information of $\mathtt{v}$ along with $y_1, y_2, \ldots, y_{t-2}$, and y_{t-1} is further encoded into h_t along with $y_1, y_2, \ldots, y_{t-2}$. Finally, $\mathbb{P}(y_t|y_{1:t-1}, \mathtt{v})$ can be modeled as follows.

$$\mathbb{P}(y_t = u|y_{1:t-1}, \mathtt{v}) = \mathbb{P}(y_t = u|h_t) = \frac{\exp(W_u^\top h_t)}{\sum_{v \in V} \exp(W_v^\top h_t)}$$

Here, $W^\top$ is the projection matrix that projects h_t from the hidden state space into the vocabulary space, $W_u^\top$ denotes its u-th row, and V is the vocabulary.

IV. Proposed Method

We first present the underlying motivation for the proposed method and the challenges to be addressed when developing it. Then we describe how to handle non-uniform and low sampling rates, as well as noisy sample points in our model. In Section IV-C, we discuss the details of the proposed spatial proximity aware loss function and cell representation learning approach in order to encourage the sequence encoder-decoder to learn consistent representations for trajectories. We discuss the time complexity of using our model to compute trajectory similarity in Section IV-D.

A. Motivations and challenges

Recent work [10], [12] reveals that the transition patterns between road network locations are often highly skewed. This implies that some routes have higher probability of being traveled than others. Rich movement patterns have been archived in spatiotemporal databases (we have 20 different data sources at Daisy[1] and different data sources have different sampling rates), which then afford us the opportunity to learn representations for trajectories that overcome the shortcomings of the point-matching methods.

Recall that we intend to learn a representation $\mathtt{v} \in \mathbb{R}^{\mathtt{n}}$ for a trajectory T that is robust to non-uniform, low sampling rates and noisy sample points when using it to compute trajectory similarity. To learn a representation for sequential data, it is very natural to consider RNNs [27], [28] since these have been shown successful at handling sequences in natural language processing, including generating sequences [30], neural machine translation [24], [25] and paraphrases detection [26]. However, by simply applying RNNs we will not be able to accomplish our goal, which is due to two difficulties: First, the trajectory representation obtained using RNNs is unable to represent the most likely underlying route that generates the trajectory when uncertainty arises due to low sampling rates and noise. Second the loss functions employed in natural language processing do not consider the spatial proximity information inherent in the spatial data, so simply adapting existing loss functions to our scenario will fail to guide RNNs to learn consistent representations for trajectories that share the same underlying route.

To tackle the first difficulty, the desired model must be able to maximize the conditional probability $\mathbb{P}(R|T)$, i.e., finding the most likely underlying route R for the given trajectory T. However, the underlying route R is often not available in

[1] http://www.daisy.aau.dk/

practice. To circumvent this, we exploit two observations: i) both a non-uniform, relatively low sampling rate trajectory, denoted by T_a, and a relatively high sampling rate trajectory, denoted by T_b, are paraphrases of their underlying route, and ii) a relatively high sampling rate trajectory T_b is closer to their true underlying route R than is T_a, and it has lower uncertainty. These observations cause us to replace the objective of maximizing $\mathbb{P}(R|T_a)$ into the objective of maximizing $\mathbb{P}(T_b|T_a)$ and to build the model using a sequence encoder-decoder framework. The encoder embeds T_a into its representation $\mathbf{v}$, and the decoder will try to recover its counterpart T_b with relatively high sampling rate conditioned on $\mathbf{v}$ by optimizing its parameters. When the model is trained using the real-world trajectories, the transition patterns hidden in historical data will be learned by the model. To overcome the second difficulty, we propose a new spatial proximity aware loss function and cell (token) representation pre-training method to incorporate spatial proximity into the model. To accelerate the training, we develop an approximate spatial proximity aware loss function based on Noise Contrastive Estimation [15].

B. Handling varying sampling rates and noise

Based on the above analysis, given a collection of sampling rate trajectories $\{T_b^{(i)}\}_{i=1}^N$ (where N is the cardinality of the collection), we create a collection of pairs (T_a, T_b), where T_b is an original trajectory and T_a is obtained by randomly dropping sample points from T_b with dropping rate r_1. By doing so, each down-sampled T_a is also non-uniformly sampled and thus represents a real-life trajectory with non-uniform and low sampling rate. The start and end points of T_b are preserved in T_a to avoid changing the underlying route of the down-sampled trajectory. To illustrate, consider generating the sub-trajectories for T_b in Figure 1b, we randomly drop points in $b_{2:5}$, i.e., all generated sub-trajectories will start with b_1 and end with b_6. After the generating procedure, we maximize the joint probability of all (T_a, T_b) pairs with the sequence encoder-decoder model:

$$\text{maximize} \quad \prod_{i=1}^{N} \mathbb{P}(T_b^{(i)}|T_a^{(i)}) \tag{2}$$

In the sequence encoder-decoder model, the inputs should be sequences of discrete tokens. Therefore, we need to find a way to map the continuous coordinates (i.e., longitude, latitude) into discrete tokens (analogous to words in natural language). We adopt a simple strategy that is used commonly in spatial data analytics, i.e., we partition the space into cells of equal size [31] and treat each cell as a token. All sample points falling into the same cell are then mapped to the same token.

The above helps mainly to overcome the problems of non-uniform and low sampling rates. However, the realistic trajectories also may have noisy sample points. For example, when a GPS receiver is in an urban canyon and satellite visibility is poor, inaccurate locations may result. To eliminate the influence of noisy sample points, we only keep the cells which are hit by more than δ sample points. These cells are referred to as hot cells and form the final vocabulary V (in the rest of paper, we will interchangeably use token and cell to refer to an element V). Sample points are represented by their nearest hot cell. To make the learned representations more robust to the noisy data, we further distort each downsampled T_a based on a distorting rate r_2 to create the distorted variants, i.e., we randomly sample a fraction of the points (size indicated by r_2) that are then distorted. Point (p_x, p_y) is distorted by adding a Gaussian noise with a radius 30 (meters) as follows,

$$\begin{aligned} p_x &= p_x + 30 \cdot d_x, \quad d_x \sim \text{Gaussian}(0,1) \\ p_y &= p_y + 30 \cdot d_y, \quad d_y \sim \text{Gaussian}(0,1) \end{aligned} \tag{3}$$

We can optimize the same objective as shown in Equation 2 where T_a is both downsampled and distorted.

C. Learning consistent representations

The original sequence encoder-decoder does not model the spatial correlation between cells, which is important in order to learn consistent representations for trajectories drawn from the same route. To address this, we propose a novel spatial proximity aware loss function (in Section IV-C1) and a new cell representation pretraining approach that takes into account spatial proximity (in Section IV-C2) To further improve the training, an approximate loss function based on Noise Contrastive Estimation [15] is also proposed (in Section IV-C1).

1) Spatial proximity aware loss function: To train a sequence encoder-decoder, we need a loss function to characterize the optimization objective. This is important, as differences in the loss function would encourage the model to learn different representations [32]. When the sequence encoder-decoder is employed in natural language processing, e.g., in neural machine translation [23], [24], [33], Negative Log Likelihood (NLL) loss is chosen to minimize the negative log likelihood function for tokens in the target sentence as follows,

$$\mathcal{L}_1 = -\log \prod_t \mathbb{P}(y_t|y_{1:t-1}, x) \tag{4}$$

However, simply adopting this loss function is problematic for the spatiotemporal data. Recall that our original purpose is to maximize $\mathbb{P}(R|T_a)$. Due to the unavailability of the underlying route R, we use the original trajectory T_b to represent R and instead maximize $\mathbb{P}(T_b|T_a)$. In practice, even original trajectories may not cover their underlying route R well. For example, Figure 3 has two trajectories T_b and $T_{b'}$ generated from the same route R (after transforming the coordinates to cells, their corresponding sequences are y and y', respectively). The sample points of the two trajectories interleave the cells in our space partitioning. Let T_a and $T_{a'}$ denote the sub-trajectories of T_b and $T_{b'}$, respectively. Ideally, the representations learned for T_a and $T_{a'}$ should be similar, as they are both generated from route R. The NLL loss function in Equation 4 differentiates T_b and T_b' as two identical target trajectories, so that it cannot discover the similarity between T_a and T_a'.

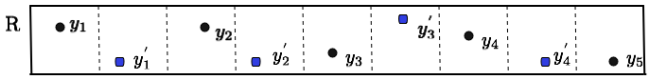

Fig. 3: T_b and $T_{b'}$ are two trajectories generated from an underlying route R. After transforming the coordinates to cells, their corresponding sequences are y, y' respectively. The sample points in the two trajectories interleave on the route.

The reason is that the loss function in Equation 4 penalizes the output cells with equal weight. Intuitively, the output cells that are closer to the target should be more acceptable than those that are far way. For example, if the decoded target cell is y_3 (in Figure 3), the loss function penalizes the outputs y_3' and y_1 equally. This is not a good penalty strategy. Since y_3' is spatially closer to y_3, it is more acceptable for the decoder to output y_3' rather than to output y_1.

The intuition behind the proposed spatial proximity aware loss function is that we assign a weight for each cell when we try to decode a target cell y_t from the decoder. The weight of cell $u \in V$ is inversely proportional to its spatial distance to the target cell y_t, so the closer the cell is to y_t the larger weight we will assign to it. The spatial proximity aware loss is given as follows.

$$\mathcal{L}_2 = -\sum_{t=1}^{|y|} \sum_{u \in V} w_{uy_t} \log \frac{\exp(W_u^\top h_t)}{\sum_{v \in V} \exp(W_v^\top h_t)}, \tag{5}$$

where

$$w_{uy_t} = \frac{\exp\left(-||u - y_t||_2/\theta\right)}{\sum_{v \in V} \exp(-||v - y_t||_2/\theta)}$$

is the spatial proximity weight for cell u when decoding target y_t, and $||u - y_t||_2$ denotes the Euclidean distance between the centroid coordinates of the cells. Here $\theta > 0$ is a spatial distance scale parameter. A small θ penalizes far away cells heavily, and when $\theta \to 0$, the loss function will be reduced to the NLL loss function in Equation 4. The exponential kernel function is chosen as it decays fast at the tail, which would encourage the model to learn to output cells near the target cell y_t.

Although the spatial proximity aware loss function in Equation 5 helps us learn consistent representations for trajectories generated from the same routes, it requires us to sum over the entire vocabulary twice every time we decode a target y_t:

$$\sum_{u \in V} w_{uy_t} \underbrace{\left(W_u^\top h_t - \sum_{v \in V} \exp(W_v^\top h_t)\right)}_{\text{log probability}} \tag{6}$$

Thus, the cost of decoding a trajectory y is $O(|y| \times |V|)$. When the vocabulary size $|V|$ is large, it will be expensive to train the model.

Approximate spatial proximity aware loss function. To reduce the training cost, we design an approximate spatial proximity aware loss function based on the following two observations: i) most of w_{uy_t} are very small except cells that are close to target cell y_t; ii) it is not necessary to calculate the exact value of the log probability in Equation 6, as long as we can encourage the decoder to assign the probability to the cells that are close to the target cell. Based on the first observation, we can use just the K nearest cells of y_t, denoted as $\mathcal{N}_K(y_t)$, instead of using the whole vocabulary in the first sum in Equation 6. Based on the second observation, we can use Noise Contrastive Estimation (NCE) [15] to compute the log probability in Equation 6. NCE was developed by Gutmann et al. [15] to differentiate data from noise by training a logistic regression. We can use it to approximately maximize the log probability of cells in $\mathcal{N}_K(y_t)$ by randomly sampling a small set of cells from $V - \mathcal{N}_K(y_t)$ as noise data, denoted as $\mathcal{O}(y_t)$. In our experiments, we find that 500 randomly sampled noise cells can give a very good approximation, and thus the time complexity is reduced from $O(|y| \times |V|)$ to $O(|y|)$. In summary, our approximate spatial proximity aware loss is given as follows.

$$\mathcal{L}_3 = -\sum_{t=1}^{|y|} \sum_{u \in \mathcal{N}_K(y_t)} w_{uy_t} \left(W_u^\top h_t - \sum_{v \in \mathcal{NO}} \exp(W_v^\top h_t)\right), \tag{7}$$

where

$$w_{uy_t} = \frac{\exp\left(-||u - y_t||_2/\theta\right)}{\sum_{v \in \mathcal{N}_K(y_t)} \exp(-||v - y_t||_2/\theta)}$$
$$\mathcal{NO} = \mathcal{N}_K(y_t) \cup \mathcal{O}(y_t)$$

2) Pre-training cell representations: To further guarantee that the trajectories generated by the same route have close representations in the latent space, we propose a cell representation learning algorithm to pre-train the cells in the embedding layer of the model. The intuition is that the encoder squeezes a sequence of cells covered by the trajectory to get the trajectory representation $\mathbf{v}$, and thus the representations of two trajectories along the same route will be close in their latent space if we can learn similar representations for cells that are spatially close. For example, if each y_i has a similar cell representation as that of y_i' in Figure 3, the trajectory representations of T_a and T_a' will be close in the latent space since they are encoded by the same encoder.

Two straightforward representations exist for the cells, the one-hot representation [32] and the centroid coordinates of the cells (GPS coordinates). However, both representations have limitations. The one-hot representation loses the spatial distance relation of the cells as all the cells are treated independently. As a result, the proposed model may take more training time to discover spatial relations in the cell embedding layer. It would help accelerate the training if the input cell representations provide the prior knowledge of spatial proximity. Next, the centroid coordinates of the cells naturally encode the spatial proximity for the cells but restrict the representations in a two-dimensional space, which make it difficult for the loss function to further optimize the representations in their parameter space.

Based on the above analysis, we propose to feed the distributed cell representations, which capture the cell spatial

proximity relation, to the embedding layer of the model. We achieve this by borrowing the key idea from skip-grams [19]. The intuition behind skip-grams is that words with similar meanings tend to appear together in the same contexts, and if we use the representation of a word to predict its surrounding words then we can embed the words into a Euclidean space in a way that captures the semantic distances between the words. Towards this end, we create the context for a given cell $u \in V$ by randomly sampling its neighbor $u' \in \mathcal{N}_K(u)$ (we also only consider its K-nearest neighbors) according to the following cell sampling distribution:

$$\mathbb{P}(u') = \frac{\exp(-||u' - u||_2/\theta)}{\sum_{v \in \mathcal{N}_K(u)} \exp(-||v - u||_2/\theta)} \tag{8}$$

The cell sampling distribution is similar to the spatial proximity weight in Equation 5. Note that their θ values do not have to be equal. For each cell $u \in V$, the cell sampling distribution tends to sample the cells that are spatially close to it as its context. In this fashion, we are able to create the context for each cell and to learn the cell representation efficiently with the negative sampling algorithm [34] by maximizing the log probability of observing the neighboring cells in its context, $\mathcal{C}(u)$, given cell u:

$$\text{maximize} \quad \sum_{u \in V} \log \mathbb{P}(\mathcal{C}(u)|g(u)) \tag{9}$$

Here, $g(u)$ denotes the representation of cell u. The learned cell representations will be used to initialize the embedding layer in the model, but we do not fix their values. Thus they can still be further optimized by the loss function in Equation 7. The learning algorithm is shown in Algorithm 1.

Algorithm 1: `CellLearning (CL)`

Input: The dimension of the learned representations d, context window size l
Output: The learned cell representations $g(u)$

```
for u ∈ V do
    C(u) ← ∅ ;
    while |C(u)| < l do
        u' ~ P(u') according to Equation 8;
        C(u) ← C(u) ∪ u';
Optimizing the loss function in Equation 9;
return g(u) for u ∈ V;
```

D. Complexity of similarity computation

Our model can be trained completely unsupervised with the SGD (Stochastic Gradient Descent) algorithm. Given a trained model, it only requires $O(n)$ time (as shown in Equation 1, where $f(\cdot,\cdot)$ indicates the encoder-RNN and h_0 is a zero vector) to embed a trajectory into a vector which is fast and can be done using GPUs. Then we can use the Euclidean distance of the two vectors to measure the similarity of two trajectories, with a time complexity of $O(|\mathtt{v}|)$. Therefore the time complexity of measuring the similarity between two trajectories is $O(n + |\mathtt{v}|)$.

TABLE II: Dataset statistics.

Dataset	#Points	#Trips	Mean length
Porto	74,269,739	1,233,766	60
Harbin	184,809,109	1,527,348	121

V. Experiments

We study the effectiveness and scalability of our proposed method on two real-world taxi datasets. The experimental setup and parameter settings are presented in Sections V-A and V-B respectively. Then we evaluate the accuracy of different methods using most similar search, cross-similarity, and k-nn queries in Sections V-C1 to V-C3, respectively. Scalability is covered in Section V-D. The proposed loss functions and cell learning approach are evaluated in Section V-E. We end by evaluating the impact of the cell size, the hidden state size of the encoder, and the training data size on the learned trajectory representations in Sections V-F and V-G.

A. Experimental setup

Dataset. The experiments are conducted on two real-world taxi datasets. The first dataset[2] is collected in the city of Porto, Portugal over 19 months and contains 1.7 million trajectories. Each taxi reports its location at 15 second intervals. We remove trajectories with length less than 30, which yields 1.2 million trajectories. The second dataset contains trajectories collected from 13,000 taxis over 8 months in Harbin, China. We select trajectories with length at least 30 and time gaps between consecutive sample points being less than 20 second. This yields 1.5 million trajectories. We partition both sets into training data and testing data based on the starting timestamp of the trajectories. For both sets, the first 0.8 million trajectories are used for training, and the remaining trajectories are used for testing. Statistics of the two sets are shown in Table II.

To create the training trajectory pairs as described in Section IV-A, we perform two kinds of transformations, down-sampling and distortion. For each trajectory T_b we first down-sample it with a dropping rate r_1 varied in $[0, 0.2, 0.4, 0.6]$ to create its 4 sub-trajectories T_a. And we further distort each down-sampled T_a based on a distorting rate r_2 (as described in Equation 3) varied in $[0, 0.2, 0.4, 0.6]$. As a result, 16 training pairs (T_a, T_b) are created for each original trajectory T_b.

Benchmarking Methods: We compare t2vec with three other methods for measuring the trajectory similarity, namely EDR [9], LCSS [7], and EDwP [11]. LCSS and EDR are two of the most widely adopted trajectory similarity measures in spatiotemporal data analyses. EDwP is the state-of-the-art method for measuring similarity of non-uniform and low sampling rate trajectories. We do not include DTW as it has been demonstrated to be consistently inferior to EDR in trajectory similarity computation [11]. Moreover, we compare with the vanilla RNN (vRNN) [35] and the common set representation (CMS). The vanilla RNN serves as an embedding baseline method, and the common set representation is used to measure

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

the similarity of two trajectories based on their common set after they have been mapped to cells. We discuss the reasons for comparing with the two baselines in Section V-C1.

Evaluation Platform: Our method[3] is implemented in Julia [36] and PyTorch, and trained using a Tesla K40 GPU. The baseline methods are written in Java[4]. The platform runs the Ubuntu 14.04 operating system with an Intel Xeon E5-1620 CPU.

B. Parameter settings and training details

Cell size: The default cell size in the experiments is 100 meters. After removing the cells hit by less than 50 points (i.e., $\delta = 50$), we get 18,866 hot cells for the Porto dataset and 22,171 hot cells for the Harbin dataset.

RNN units: In our model, GRU [35] with 3 layers is chosen as the computational unit because it has been shown to be as good as LSTM [37] in sequence modeling tasks, while it is much more efficient to compute [35].

Gradient clipping: Although RNNs tend not suffer from gradient vanishing problem, they may have exploding gradients [38]. Hence, we clip the gradients by enforcing a maximum gradient norm constraint [30], which is set to 5 in our experiments.

Terminating condition: We randomly select 10,000 trajectories as a validation dataset from the test dataset (the selected trajectories are removed from the test dataset). The training is terminated if the loss in the validation dataset does not decrease in 20,000 successive iterations.

In addition, both the hidden layer size in GRU and the dimension of the learned cell representation d are set to 256, the context window size l in the cell learning algorithm is set to 10. For simplicity, θ in Equations 5 and 8 is fixed at 100 (meters). Parameter K and the size of $\mathcal{O}(y_t)$ in Section IV-C are set to 20 and 500, respectively. We adopt Adam stochastic gradient descent [39] with an initial learning rate of 0.001 to train the model. We evaluate the training time in Sections V-E and V-F.

To set the parameter ϵ of the baseline methods EDR and LCSS, we adopt the strategies described in the studies [7], [9] proposing the two methods; the parameters of vRNN are set to be the same as our encoder-RNN except that it is trained by predicting the next cell based on the cells that it has already seen.

C. Performance evaluation

The lack of ground-truth dataset makes it a challenging problem to evaluate the accuracy of trajectory similarity. Two recent studies [11], [13] propose to evaluate the accuracy of methods for computing trajectory similarity using the self-similarity and cross-similarity comparisons and assessments of the precision of finding the k-nearest neighbors. Currently, this is the best evaluation methodology, and we adopt this methodology in the experiments. In addition, we also design a new experiment, called most similar search (which can be considered as a sort of self-similarity measure) to evaluate the effectiveness of different methods, in Section V-C1.

[3]https://github.com/boathit/research-papers/tree/master/t2vec

[4]The authors of EDwP give us access to their compiled jar file.

TABLE III: Mean rank versus the database size using the Porto and Harbin datasets.

Porto					
DB size	20k	40k	60k	80k	100k
EDR	25.73	50.70	76.07	104.01	130.98
LCSS	31.95	59.20	95.85	130.40	150.67
CMS	62.18	112.84	173.34	231.55	291.26
vRNN	32.73	61.24	100.20	135.22	163.10
EDwP	6.78	11.48	16.08	23.02	28.90
t2vec	**2.30**	**3.45**	**4.73**	**6.35**	**7.67**

Harbin					
DB size	20k	40k	60k	80k	100k
EDR	30.37	57.90	85.72	118.02	149.01
LCSS	35.49	63.20	105.46	137.20	160.67
CMS	97.41	141.04	209.37	271.45	316.81
vRNN	34.30	65.24	103.05	140.25	162.10
EDwP	12.80	20.64	29.10	35.20	45.30
t2vec	**5.10**	**7.50**	**9.62**	**12.51**	**15.70**

One of the most important tasks in trajectory analysis is similar trajectory search. To overcome the lack of ground-truth, we design three experiments to evaluate the performance using different methods for this task.

We randomly choose 10,000 trajectories from the test dataset, denoted as Q, and then we choose another m (a parameter to be evaluated) trajectories, denoted by P. For each trajectory $T_b \in Q$, we create two sub-trajectories from it by alternately taking points from it, denoted as T_a and $T_{a'}$ (see Figure 4), and we use them to construct two datasets $D_Q = \{T_a\}$ and $D'_Q = \{T_{a'}\}$. We perform the same transformation for the trajectories in P to get D_P and D'_P. Then for each query $T_a \in D_Q$, we retrieve its top-k most similar trajectories from database $D'_Q \cup D'_P$ and calculate the rank of $T_{a'}$. Ideally $T_{a'}$ is ranked at the top since it is generated from the same original trajectory as T_a. The reason for using $D'_Q \cup D'_P$ as the database instead of $D'_Q \cup P$ is that the query trajectory will have similar mean length as the trajectories in the database[5]. Moreover, to evaluate whether RNN encodes two sequences of cells into two similar vectors simply because the two sequences have the same starting or ending cells, or just because they have sufficient numbers of common cells, we include another two baselines, vRNN and CMS. If the aforementioned reason is true, vRNN and CMS should also give good performance in the task.

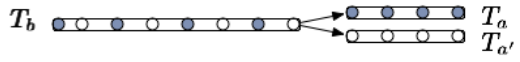

Fig. 4: Creating two sub-trajectories T_a and $T_{a'}$ from trajectory T_b by alternately taking points from it.

1) Most similar trajectory search: **Experiment 1** We first study the performance of the different methods when we increase m, the size of P, from 20,000 to 100,000. Table III

[5]Similar results were found using database $D'_Q \cup P$.

TABLE IV: Mean rank versus the down-sampling rate r_1 using the Porto and Harbin datasets.

Porto					
r_1	0.2	0.3	0.4	0.5	0.6
EDR	160.03	208.01	235.60	285.10	340.68
LCSS	168.02	173.45	187.60	188.40	192.20
CMS	296.56	317.70	430.00	387.90	446.50
vRNN	173.45	179.58	190.24	200.13	210.20
EDwP	29.10	30.50	31.64	39.67	61.72
t2vec	**7.88**	**8.00**	**9.48**	**12.70**	**15.99**

Harbin					
r_1	0.2	0.3	0.4	0.5	0.6
EDR	183.02	231.01	265.50	316.30	380.86
LCSS	178.80	193.50	195.30	208.40	210.30
CMS	330.70	376.20	450.04	460.90	476.50
vRNN	176.45	184.83	191.42	203.13	250.20
EDwP	47.32	49.80	51.39	57.91	81.81
t2vec	**15.92**	**17.21**	**19.87**	**21.74**	**30.95**

TABLE V: Mean rank versus the distorting rate r_2 using the Porto and Harbin datasets.

Porto					
r_2	0.2	0.3	0.4	0.5	0.6
EDR	132.40	133.10	135.60	134.90	139.10
LCSS	210.30	215.70	214.60	215.05	228.03
CMS	296.16	317.27	337.31	327.90	346.05
vRNN	212.16	220	217.30	220.61	235.70
EDwP	30.10	30.16	32.63	31.23	33.53
t2vec	**9.10**	**9.20**	**9.52**	**9.49**	**10.80**

Harbin					
r_2	0.2	0.3	0.4	0.5	0.6
EDR	142.44	143.69	145.67	146.90	152.11
LCSS	230.32	235.93	244.65	245.15	251.60
CMS	306.62	327.87	329.31	339.34	349.51
vRNN	222.41	227.20	236.37	250.62	245.75
EDwP	46.41	47.97	49.32	50.68	51.10
t2vec	**16.43**	**17.28**	**17.52**	**18.51**	**21.08**

shows the mean rank of the 10,000 queries in D_Q using different methods when using the Porto and Harbin datasets. As the size of P grows, the performance of all methods degrades. CMS performs the worst, as it ignores the sequential information in the trajectories. vRNN and LCSS demonstrate similar performance, and this is reasonable because both methods can be considered as enhanced version of CMS that preserve the order of the points in the sequence. EDwP performs the best among all baseline methods. t2vec outperforms the other methods significantly, and even when the size of database P reaches 100,000, it gives a low mean rank for the queries.

Experiment 2 Next, we study the impact of down-sampling on the methods with a fixed database size $|D'_Q \cup D'_P| = 100,000$. We vary the dropping rate r_1 from 0.2 to 0.6 and down-sample the trajectories in both D_Q and $D'_Q \cup D'_P$ based on the dropping rate. Table IV depicts the mean rank for the queries in D_Q of the different methods using Porto and Harbin datasets. EDR degrades quickly when we increase the dropping rate r_1, while LCSS and vRNN are not very sensitive to variations in r_1. EDwP shows relatively consistent performance when the down-sampling rate r_1 varies between 0.2 and 0.5, but when raising r_1 to 0.6, its mean rank increases markedly. This implies that the linear interpolation may not be able to handle a low sampling rate effectively if the dropping rate is large. The reason is that the assumption made by the linear interpolation that the object would move along a straight line between two consecutive sample points is no longer true. t2vec consistently outperforms the other methods by a large margin.

TABLE VI: Mean cross-distance deviation for varying dropping rate r_1 and distorting rate r_2.

r_1	0.1	0.2	0.4	0.6
t2vec	**0.057**	**0.010**	**0.016**	**0.025**
EDwP	0.059	**0.010**	0.024	0.039
EDR	0.130	0.190	0.380	0.580

r_2	0.1	0.2	0.4	0.6
t2vec	**0.010**	**0.013**	**0.018**	**0.021**
EDwP	**0.010**	0.018	0.031	0.038
EDR	0.012	0.019	0.033	0.039

Experiment 3 Finally, we evaluate the effect of noise on results. We still fix $|D'_Q \cup D'_P| = 100,000$ and distort the points of trajectories in both query D_Q and database $D'_Q \cup D'_P$ with distorting rate r_2, as described in Equation 3. The results are shown in Table V using the Porto and Harbin datasets. Unlike with down-sampling, we observe that all methods are not very sensitive to point distortion. Even if the distorting rate is set to 0.6, no method shows obvious performance degradation. We observe that t2vec achieves the best performance.

In subsequent experiments, we observes similar results on the two datasets, and we only report the results when using the Porto dataset due to the space limitation.

2) Cross-similarity comparison: A good similarity measure should be able to not only recognize the trajectory variants of the same underlying route (self-similarity), but should also preserve the distance between two different trajectories, regardless of the sampling strategy. We adopt an evaluation criterion from the literature [13], [18], namely cross distance deviation, which is calculated as follows:

$$\frac{|d(T_a(r), T_{a'}(r)) - d(T_b, T_{b'})|}{d(T_b, T_{b'})},$$

where T_b and $T_{b'}$ represent two distinct original rate trajectories, and $T_a(r)$ and $T_{a'}(r)$ are their variants obtained by randomly dropping (or distorting) sample points with the dropping (or distorting) rate r. We randomly select 10,000 trajectory pairs $(T_b, T_{b'})$ from the test dataset to calculate their mean cross distance deviation. A small cross distance deviation indicates that the evaluated distance is much close to the ground truth. The mean cross distance deviation of the different methods is described in Table VI, where we vary the dropping rate r_1 and the distorting rate r_2. We notice that t2vec outperforms the other two methods in terms of cross distance deviation for different dropping and distorting rates.

3) k-nn queries: The similarity measure is a fundamental operation that can be used in many applications, e.g., similarity search, clustering, and classification. In this experiment,

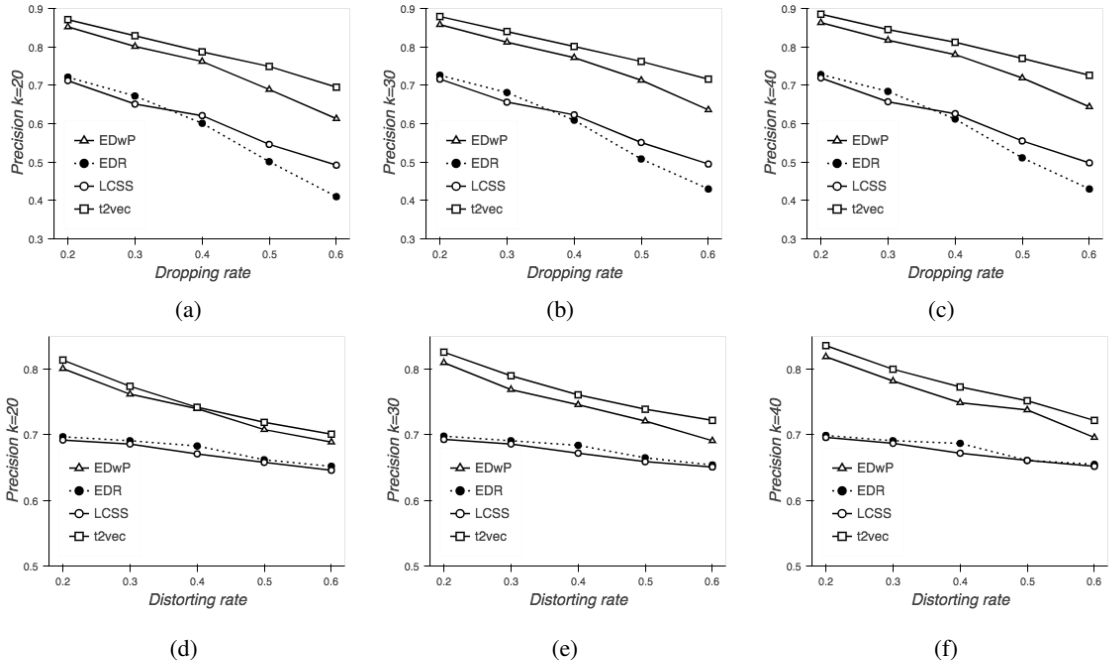

Fig. 5: (a)-(c) k-nn results when varying the dropping rate for $k = 20, 30, 40$. (d)-(f) k-nn results when varying the distorting rate for $k = 20, 30, 40$.

we evaluate the performance of different similarity search methods. To contend with the lack of ground-truth, we follow the methodology used in previous work [11], [13]. We first randomly choose 1000 trajectories as query and 10,000 trajectories as the target database from the test dataset. We apply each method to find the k-nearest-neighbors (k-nn) of each query trajectory from the target database as its ground-truth. Next, we transform queries and database trajectories by randomly dropping (resp. distorting) certain sample points according to the dropping (resp. distorting) rate r_1 (resp. r_2). Finally, for each transformed query, we find its k-nn from the target database using each method and then compare the result with the corresponding ground-truth. The rationale behind this methodology is that a robust distance measure should adapt to non-uniform and relatively low sampling rates (resp. distortion) and yield results close to those for relatively high sampling rate (resp. non-distorted) counterparts.

Figure 5 shows the precision (the proportion of true k-nn trajectories) of the different similarity methods when the dropping rate (as shown in Figure 5a-5c) and the distorting rate (as shown in Figure 5d-5f) is varied for $k = 20, 30, 40$. The precision of all methods decreases when the dropping rate or distorting rate increases. EDR and LCSS show similar performance, but the precision of EDR drops rapidly when the dropping rate reaches 0.6. EDwP surpasses them by a fair magnitude, and t2vec consistently performs the best.

D. Scalability

The time complexity of LCSS and EDR for determining the similarity of two trajectories T_a, T_b is $O(|T_a| \times |T_b|)$. EDwP has complexity $O((|T_a| + |T_b|)^2)$. These methods rely on intricate pruning techniques [9], [11] to answer k-nn queries on large datasets. As discussed in Section IV-D, once our model has been trained offline, it takes linear time to encode a trajectory into a vector $\mathtt{v}$, and the encoding process can also be done offline. As an example, we can encode the 1.7 million trajectories in the Porto dataset into vectors within 30 minutes using one Tesla K40 GPU. After encoding the trajectories into vectors offline, its online complexity is $O(|\mathtt{v}|)$. The linear complexity makes t2vec scale well on large datasets. Note that model training is done offline; we will study the training time in Section V-E.

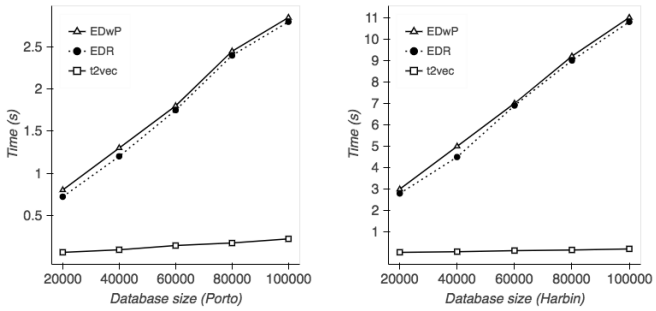

Fig. 6: (a) k-nn query efficiency versus database size using the Porto dataset. (b) k-nn query efficiency versus database size using the Harbin dataset.

Although it is obvious that our method is much more efficient in terms of the complexity analysis, we conduct an experiment to compare the efficiency of t2vec with those of EDR and EDwP empirically. Figures 6a and 6b show that the query time grows with the size of the target database for the k-nn query ($k = 50$) using the Porto and Harbin dataset, respectively. Although both EDR and EDwP employ

TABLE VII: Mean rank and training time (hours) for the model equipped with loss functions $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$, $\mathcal{L}_3 + \texttt{CL}$ using the Porto dataset.

Loss	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_3 + \texttt{CL}$
$\texttt{MR@}r_1 = 0.4$	46.56	21.34	9.70	**9.48**
$\texttt{MR@}r_1 = 0.5$	55.72	27.30	13.50	**12.70**
$\texttt{MR@}r_1 = 0.6$	68.49	32.01	16.52	**15.99**
Time	26	120	22	**14**

TABLE VIII: The impact of the cell size on the model using the Porto dataset.

Cell size	25	50	100	150
#Cells	60,004	35,335	18,866	12,425
$\texttt{MR@}r_1 = 0.5$	216.23	15.21	**12.70**	**12.70**
$\texttt{MR@}r_1 = 0.6$	234.18	19.21	**15.99**	16.03
$\texttt{MR@}r_2 = 0.5$	291.57	**9.49**	**9.49**	9.51
$\texttt{MR@}r_2 = 0.6$	302.91	10.87	**10.80**	11.03
Time	37	25	14	8

carefully designed pruning and indexing techniques, t2vec is at least one order of magnitude faster than both methods. t2vec offers near-instantaneous response times that support interactive use and analysis on big trajectory data, such as trajectory clustering, while the competition cannot. A response in less that 200 ms is perceived as instantaneous.

E. Evaluation on the loss function

In this experiment we evaluate the effectiveness of the proposed loss function and the cell representation learning (CL in Algorithm 1) approach on most similar trajectory search by using the same setting in Section V-C1. The database size is fixed at $|D'_Q \cup D'_P| = 100,000$. Table VII shows the mean rank (MR) w.r.t dropping rates $r_1 = 0.4, 0.5, 0.6$ and the training time of different loss functions using the Porto dataset. The $\mathcal{L}_2$ loss, is very expensive to compute, and since the model does not converge even after training for over 5 days (120 hours), we terminate the training process before it converges. $\mathcal{L}_3$ loss is capable of improving the mean rank significantly when compared to $\mathcal{L}_1$. The cell representation learning approach further improves the mean rank and reduces the training time by 1/3.

F. Effect of the cell size and the hidden layer size

Intuitively, a small cell size provides a higher resolution of the underlying space, but it also generates more cells (tokens), which leads to higher training complexity since the model complexity is linear in the number of tokens [40]. We evaluate the influence of the cell granularity on the method performance in answering most similar search with $r_1 = 0.5, r_2 = 0.5$. As shown in Table VIII, a cell size of 100 gives the best mean rank. The smallest cell size (25) performs the worst, which is probably because it has the highest model complexity and thus is much more difficult to train.

TABLE IX: The impact of the hidden layer size on the model using the Porto dataset.

$\lvert\texttt{v}\rvert$	64	128	256	484	512
$\texttt{MR@}r_1 = 0.5$	400.01	50.21	12.70	**10.24**	11.26
$\texttt{MR@}r_1 = 0.6$	431.11	63.71	15.99	**16.70**	17.42
$\texttt{MR@}r_2 = 0.5$	390.27	48.36	9.49	**8.01**	9.09
$\texttt{MR@}r_2 = 0.6$	397.22	50.26	10.80	**9.27**	10.05

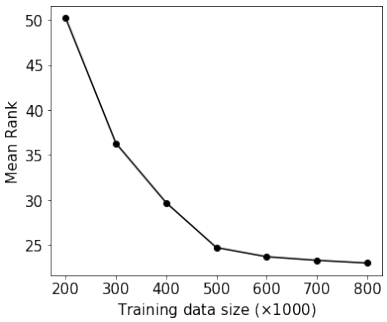

Fig. 7: The impact of training dataset size on the model using the Porto dataset.

Another important parameter that determines the quality of the learned representation v is the dimension of the hidden layer in the encoder. A high dimension of the hidden layer is typically much more expressive, but requires more training data to avoid overfitting. Table IX summarizes the impact of the hidden layer size on the most similar trajectory search with the same settings as in the above experiment. It is obvious that increasing $|\texttt{v}|$ from 64 to 256 significantly enhances the quality of the learned representations, while the performance drops when we increase it further.

G. Effect of the training data size

In this experiment, we evaluate the effect of training data size on the accuracy of the trajectory similarity search. The setting is similar to the one in Section V-E, with exception that we vary the training data size and fix the dropping rate r_1 at 0.6.

Figure 7 shows the effect of the training data size on the most similar search on the Porto dataset. The mean rank drops rapidly as we increase the training data size from 200,000 to 600,000, and the decrease slows down when we continue to enlarge the training data size. When we increase the training data size from 200,000 to 600,000, more transition patterns can be learned for representing the trajectories, and thus the model quality is enhanced significantly. However, when we further increase the training data size, the marginal benefit of using larger training data is less pronounced.

VI. Conclusions and Future Work

To the best of our knowledge, we present the first solution for using learning for creating representations of trajectories that are well suited for use in trajectory similarity computation. Most existing methods for trajectory similarity computation measure similarity by means of pairwise point-matching, which may not capture the underlying route information of the trajectories, thus being sensitive to non-uniform and low sampling rates, and noise sample points. Moreover, all of them have a quadratic time complexity. Motivated by these

observations, we propose a seq2seq-based method to learn representations for trajectories that enable accurate and efficient trajectory similarity search that is robust to sampling rate variations and noisy sample points. The method is evaluated empirically with favorable results in terms of both accuracy and efficiency. It consistently outperforms the baseline methods by a large margin in the most similarity tasks. The method is at least one order of magnitude faster than the other methods, thus it can support big trajectory analysis and interactive use while the competition cannot.

This work sheds light on several new research directions: 1) Employing the learned representations to explore more downstream tasks, e.g., trajectory clustering and popular-routes search. 2) Extending the proposed method to more general time series data beyond trajectories. 3) Developing indexing techniques like Locality-Sensitive Hashing [41] to further speed up the proposed method.

Acknowledgments. This work is supported in part by the Rapid-Rich Object Search (ROSE) Lab at Nanyang Technological University. The ROSE Lab is supported by the National Research Foundation, Prime Minister's Office, Singapore, under its IDM Futures Funding Initiative, administered by the Interactive and Digital Media Programme Office. This work was also supported by the MOE Tier-2 grant MOE2016-T2-1-137, MOE Tier-1 grant RG31/17, NSFC under the grant 61772537, and a grant from Microsoft. C. S. Jensen was supported by the DiCyPS project and by a grant from the Obel Family Foundation. W. Wei was supported by NSFC under the grant 61602197, NSF under the grant 2016CFB192.

REFERENCES

[1] Z. Li, J. Han, M. Ji, L. A. Tang, Y. Yu, B. Ding, J. Lee, and R. Kays, "Movemine: Mining moving object data for discovery of animal movement patterns," *ACM TIST*, vol. 2, no. 4, pp. 37:1–37:32, 2011.

[2] Z. Chen, H. T. Shen, and X. Zhou, "Discovering popular routes from trajectories," in *ICDE*, 2011, pp. 900–911.

[3] C.-C. Hung, W.-C. Peng, and W.-C. Lee, "Clustering and aggregating clues of trajectories for mining trajectory patterns and routes," *VLDBJ*, vol. 24, no. 2, pp. 169–192, 2015.

[4] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *PVLDB*, vol. 1, no. 1, pp. 1068–1080, 2008.

[5] X. Li, V. Ceikute, C. S. Jensen, and K.-L. Tan, "Effective online group discovery in trajectory databases," *IEEE TKDE*, vol. 25, no. 12, pp. 2752–2766, 2013.

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

[7] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *ICDE*, 2002, pp. 673–684.

[8] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *PVLDB*, 2004, pp. 792–803.

[9] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD*, 2005, pp. 491–502.

[10] K. Zheng, Y. Zheng, X. Xie, and X. Zhou, "Reducing uncertainty of low-sampling-rate trajectories," in *ICDE*, 2012, pp. 1144–1155.

[11] S. Ranu, P. Deepak, A. D. Telang, P. Deshpande, and S. Raghavan, "Indexing and matching trajectories under inconsistent sampling rates," in *ICDE*, 2015, pp. 999–1010.

[12] P. Banerjee, S. Ranu, and S. Raghavan, "Inferring uncertain trajectories from partial observations," in *ICDM*, 2014, pp. 30–39.

[13] H. Su, K. Zheng, H. Wang, J. Huang, and X. Zhou, "Calibrating trajectory data for similarity-based analysis," in *SIGMOD*, 2013, pp. 833–844.

[14] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *ICML*, 2006, pp. 369–376.

[15] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *AISTATS*, 2010, pp. 297–304.

[16] E. Frentzos, K. Gratsias, and Y. Theodoridis, "Index-based most similar trajectory search," in *ICDE*, 2007, pp. 816–825.

[17] S. Sankararaman, P. K. Agarwal, T. Mølhave, J. Pan, and A. P. Boedihardjo, "Model-driven matching and segmentation of trajectories," in *SIGSPATIAL*, 2013, pp. 234–243.

[18] H. Wang, H. Su, K. Zheng, S. Sadiq, and X. Zhou, "An effectiveness study on trajectory similarity measures," in *Australian Database Conference*, 2013, pp. 13–22.

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[20] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents." in *ICML*, 2014, pp. 1188–1196.

[21] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *SIGKDD*, 2014, pp. 701–710.

[22] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[23] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[24] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.

[25] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *NIPS*, 2015, pp. 2773–2781.

[26] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," in *NIPS*, 2015, pp. 3294–3302.

[27] D. Williams and G. Hinton, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–538, 1986.

[28] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[29] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *JMLR*, vol. 3, pp. 1137–1155, 2003.

[30] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[31] R. H. Güting and M. Schneider, "Realm-based spatial data types: the rose algebra," *VLDBJ*, vol. 4, no. 2, pp. 243–286, 1995.

[32] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *TPAMI*, vol. 35, no. 8, pp. 1798–1828, 2013.

[33] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[34] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.

[35] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[36] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[37] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[38] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *ICML (3)*, 2013, pp. 1310–1318.

[39] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[40] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," *arXiv preprint arXiv:1412.2007*, 2014.

[41] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *FOCS*, 2006, pp. 459–468.