World Wide Web (2023) 26:1501–1523
https://doi.org/10.1007/s11280-022-01089-0

Spatial-temporal fusion graph framework for trajectory similarity computation

Silin Zhou[1] · Peng Han[2] · Di Yao[3] · Lisi Chen[1] · Xiangliang Zhang[4]

Received: 27 June 2022 / Revised: 13 July 2022 / Accepted: 16 July 2022 /
Published online: 22 September 2022

Abstract
Trajectory similarity computation is an essential operation in many applications of spatial data analysis. In this paper, we study the problem of trajectory similarity computation over spatial network, where the real distances between objects are reflected by the network distance. Unlike previous studies which learn the representation of trajectories in Euclidean space, it requires to capture not only the sequence information of the trajectory but also the structure of spatial network. To this end, we propose GTS, a brand new framework that can jointly learn both factors so as to accurately compute the similarity. It first learns the representation of each point-of-interest (POI) in the road network along with the trajectory information. This is realized by incorporating the distances between POIs and trajectory in the random walk over the spatial network as well as the loss function. Then the trajectory representation is learned by a Graph Neural Network model to identify neighboring POIs within the same trajectory, together with an LSTM model to capture the sequence information in the trajectory. On the basis of it, we also develop the GTS^{+} extension to support similarity metrics that involve both spatial and temporal information. We conduct comprehensive evaluation on several real world datasets. The experimental results demonstrate that our model substantially outperforms all existing approaches.

✉ Lisi Chen
lchen012@e.ntu.edu.sg

Silin Zhou
zhousilinxy@gmail.com

Peng Han
pengh@cs.aau.dk

Di Yao
yaodi@ict.ac.cn

Xiangliang Zhang
xzhang33@nd.edu

1 University of Electronic Science and Technology of China, Chengdu, China

2 Aalborg University, Aalborg, Denmark

3 Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

4 University of Notre Dame, Notre Dame, USA

Keywords Trajectory · Similarity search · Spatial network · Deep learning · Spatio-temporal

1 Introduction

Trajectory similarity computation is a fundamental operation in a wide range of real world applications, such as route search [1–5], route planning [6–8], trajectory clustering [9–11], trajectory compression [12–14], and transportation optimizations [15–17]. A trajectory describes the path traced by bodies moving in space over time [18] and is usually represented as a sequence of discrete locations. To measure the similarity between two trajectories, many metrics are proposed in previous studies, such as Dynamic Time Warping [19] (DTW), longest common subsequence [20] (LCSS), edit distance with real penalty [21] (ERP) and edit distance on real sequences [22] (EDR). However, these metrics require quadratic computational complexity $\mathcal{O}(n^2)$, where n is the length of trajectories. As a result, the high computation cost of the above similarity metrics becomes a serious problem when dealing with massive trajectory data. To resolve the problem, some recent studies [6, 23, 24] utilized neural network based models to learn the representation of trajectories. The similarity between trajectories could be measured by that between the low-dimensional embedding vectors, which can be completed in linear time.

While the above approaches are effective for measuring the trajectory similarity in Euclidean space, they cannot be applied to the problem of trajectory similarity computation over the spatial network, such as road network. In many real application scenarios, objects are moving in spatial networks rather than in Euclidean space. In a spatial network, Euclidean distance may lead to errors when calculating the real distance between objects. Let us consider the example shown in Figure 1. The Euclidean distance between trajectories τ_1 and τ_2 is smaller than that between τ_1 and τ_3. But the distance on the road network between τ_1 and τ_3 is actually much smaller, as there is no passage between τ_1 and τ_2 in the road network.

There are several previous studies [25–28] focusing on the computation of trajectory similarity over spatial networks. They proposed hand-crafted heuristic approaches to align the trajectory to the spatial network to compute some user-defined similarity functions, which still suffer from high computational overhead. However, the difficulty in adopting deep learning-based techniques to this problem is two-fold. On the one hand, it is essential to consider the network structure when learning the trajectory embedding, while existing solutions for Euclidean space [6, 23, 24] only capture the sequence information. On the other hand, the learning process suffers from data sparsity: due to the large problem space which is exponential w.r.t. the number of POIs in the spatial network, the coverage of training data might be insufficient to include all possible combinations. As a result, once a trajectory pattern is infrequent or even missing in the training data, the trained model cannot learn a high-quality embedding for it.

To address above issues, in this paper we propose **G**raph-based approach for measuring **T**rajectory **S**imilarity (GTS), a novel framework of trajectory representation learning for similarity computation over spatial networks. GTS consists of three steps, namely measuring trajectory similarity, learning point-of-interest (POI) representation, and generating trajectory embedding. We start from the similarity measurement between trajectories, which is the first step towards a robust framework for learning trajectory embedding. To reflect the relationship between trajectories on the road network as well as the inherited properties

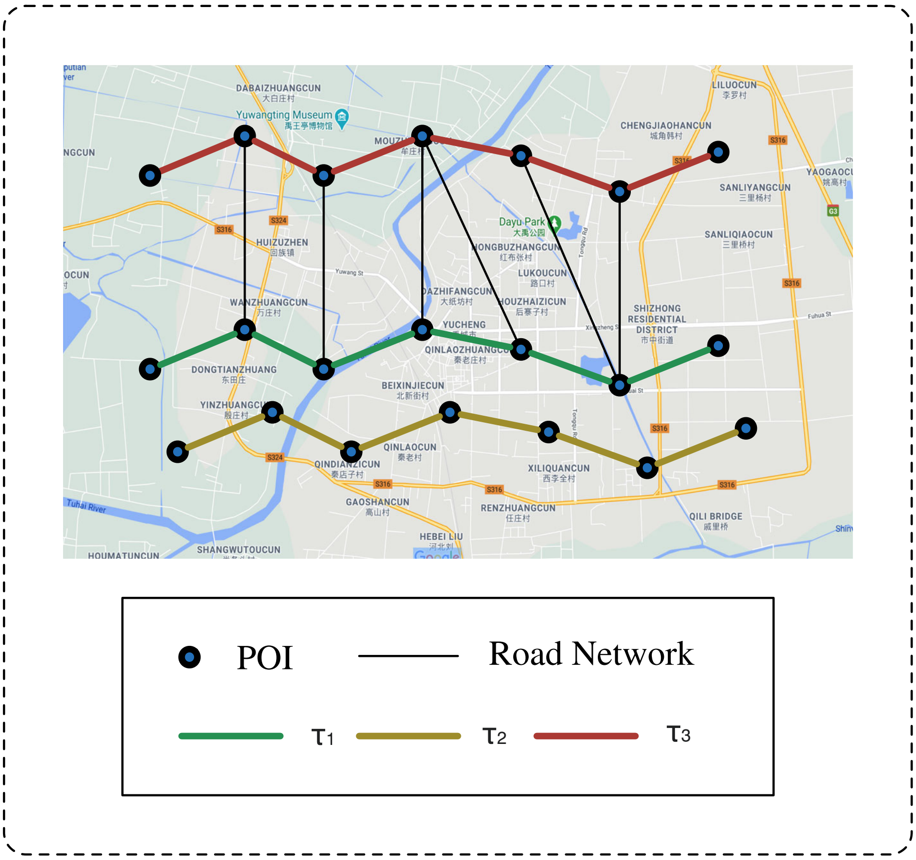


Figure 1 Example of Trajectory Similarity Measurement over Spatial Network

of every single trajectory, we define the trajectory similarity from three aspects: POI-wise distance, POI-Trajectory distance, and Trajectory-wise similarity.

Based on such definitions of trajectory similarity over the spatial network, we then learn the trajectory embedding in the following two steps. We first learn the embedding of each POI in the spatial network, which serves as a cornerstone for the embedding of trajectories. While previous works [29–31] learn the POI embedding mainly by learning its spatial information, here we need to consider the trajectories along with the topology of the spatial network. To this end, we propose a trajectory-aware random walk algorithm and a new loss function to train a skip-gram model such that POIs co-occurring in these random walks would produce similar embeddings. In the next step, we learn trajectory representation based on such POI embeddings. To overcome the data sparsity problem, we use Graph Neural Network (GNN) to encode the embedding of each POI with its neighbor information. Then a trajectory becomes a sequence of POIs and we can learn its representation with a Long Short-Term Memory (LSTM) network. In this way, the learned representation will contain richer information of the network structure and thus is capable to reflect various trajectory patterns even if they are not explicitly included in the training data.

Moreover, we further take the temporal information in trajectories into consideration and proposed an extended approach GTS$^+$ for trajectory similarity computation based on both spatial and temporal information. Specifically, we discretize the time intervals into smaller units and construct the temporal-aware graph of all POIs based on the co-occurrence of their time units. Then we propose a novel LSTM variant termed as ST-LSTM to jointly capture the spatial and temporal information to learn the trajectory embedding.

The main contributions of this paper are summarized as follows:

- We propose a graph-based framework GTS for the problem of trajectory similarity computation over the spatial network. To the best of our knowledge, it is the first work to solve this problem with deep learning techniques.
- We devise a trajectory-aware random walk algorithm with a new loss function to learn the embedding of each POI in the spatial network to integrate the trajectory information with the network structure.
- We further design a GNN-LSTM model which is robust to data sparsity and noisy in given trajectories to learn high-quality trajectory representations.
- We propose the GTS$^+$ framework to support trajectory similarity computation with both spatial and temporal information.
- We conduct an extensive set of experiments on popular real-world datasets. The results show that our proposed methods significantly outperform the existing approaches in terms of accuracy.

The rest of the paper is organized as follows: Section 2 surveys the related work. Section 3 introduces necessary background knowledge and problem settings. Sections 4 and 5 propose the techniques to learn the representation of a single location and the whole trajectory, respectively. Section 6 introduces how to extend the above techniques to include temporal information and proposes the GTS$^+$ framework accordingly. Section 7 reports the experimental results. Section 8 concludes the paper.

2 Related work

2.1 Non-learning trajectory similarity computation

There are two categories of traditional approaches for trajectory similarity computation. One is grid-based similarity, which uses distances in Euclidean space. Previous studies in this category rely on the distance aggregation over all points on the trajectory to compute the similarity, such as Dynamic Time Warping [19] (DTW), longest common subsequence [20] (LCSS), edit distance with real penalty [21] (ERP), edit distance on real sequences [22] (EDR) and Hausdorff [32]. They are expensive in efficiency even with some optimizations [33] and might suffer from noisy points in trajectories.

The other one is spatial network based similarity [26, 34], where trajectories are first mapped to the spatial network and then the similarity is computed by applying similarity functions on top of the transformed trajectories. Earlier approaches utilized metrics like shortest path and set-based similarity to describe the similarity between trajectories. Shang et al. [26] proposed a joint similarity function to consider both spatial and temporal similarity as well as several indexing and pruning techniques. Wang et al. [35] defined a new function called Longest Overlapping Road Segments to measure the similarity between two transformed trajectories. Unfortunately, existing road-constrained trajectory measures either suffer from the high computation cost or are not capable to satisfy the requirement of real-life applications.

There are many applications regarding trajectory similarity computation. Several previous studies [3, 8, 36] aimed at accelerating the similarity search and join over trajectory data by devising index and pruning techniques following the idea from string similarity search [37–40]. Specifically, tree-based index structures [22, 27] , such as K-D tree or R-tree are employed to organize the trajectories. Then, bounding-box-based pruning techniques are proposed to eliminate unnecessary computations. Zheng et al. [41] studied the problem of inference hidden route from known trajectories. Song et al. [15] focused on the problem of trajectory compression based on the road network. Han et al. [42] investigated this problem by considering only spatial information.

2.2 Deep learning based approaches

Deep learning techniques have been widely adopted to many problems related to spatial data analysis [29, 42–44]. A comprehensive survey is made in [18]. Some existing studies employ neural network models to learn the representation of trajectories and then compute the similarity by measuring that between the embedding vectors. Li et al. [23] adopted an encoder-decoder architecture to obtain trajectory vector representations. Yao et al. [24, 45] further improved the performance by devising new spatial attention mechanism and using pair-wise distance as guidance for learning. Zhang et al. [6] proposed several new loss functions to improve the quality of learned embedding. All the above methods are designed for similarity metrics in Euclidean space and cannot be directly adopted for our problem as they fail to learn the information from the spatial network. Deep learning techniques are also adopted for other related problems, such as clustering [9] and route prediction [25], which have different problem settings from our work.

2.3 Graph neural networks

Recent works on the Graph Neural Network (GNN), especially graph convolutional networks (GCNs) [46] have attracted considerable attention, motivating the remarkable success in various graph mining tasks in multiple domains [47, 48]. GNNs originated from the spectral graph networks [49]. Afterwards, Kipf and Welling [46] further extended it for semi-supervised node classification with concise form and achieved great success. Taking account of large-scale networks, Hamilton et al. [50] approximated GCN by an inductive representation learning framework. Later, the attention mechanism was also introduced to adaptively specify the weights during the training process [51]. Our work took advantage of GNN to obtain neighborhood information of each node in the spatial network so as to overcome the data sparsity problem.

3 Preliminary

3.1 Trajectory with spatial networks

We first describe our data model. The spatial network is represented as an undirected graph $G = (V, E)$. In this graph, each node $v \in V$ is a POI in the spatial network, where the representation of a road intersection or a road end with attributes latitude and longitude. Meanwhile, each edge $e = \langle v_i, v_j \rangle \in E$ represents the distance between two POIs v_i and

v_j. An original trajectory $\tau = \{p_1, p_2, \ldots, p_k\}$ is composed with sequential points with latitude and longitude. Then we mapped them into the POI set V with the nearest distance to generate the corresponding vertex trajectory $\tau = \{v_{n1}, v_{n2}, \ldots, v_{nk}\}$ and the length of a trajectory (denoted as $|\tau|$) is defined as the number of POIs in it.

3.2 Spatial Similarity between trajectories

Different from previous studies using Euclidean distance, the similarity measurement in our work should not only reflect the property of a trajectory, but also reflect the property of the spatial network. To achieve this goal, we define trajectory similarity by considering the distances from two aspects: POI-wise distance and POI-Trajectory distance.

The *POI-wise distance* is the distance between two POIs over the road network, which is defined as the length of the shortest path between them. Given two POIs $v_i, v_j \in V$, if v_i is reachable from v_j, we use $d(v_i, v_j)$ to denote the length of shortest path, i.e. the *POI-wise distance* between them.

Similarly, we define the *POI-Trajectory distance* as the shortest distance between the POI and the trajectory. However, the computation of the exact value is very expensive as we need to compute distances between the POI and all segments of this trajectory. To reduce the computation overhead, we define the POI-Trajectory distance as the shortest POI-wise distance between the given POI and all POIs in the trajectory. Although the computational complexity of our definition is the same as that of the original method, the cost is much less in practice since the distances between POIs in our definition could be reused for different trajectories and the amortized cost would be rather low. Then given one POI v and trajectory τ, the POI-Trajectory distance $d(v, \tau)$ from the POI to the trajectory is formulated as (1):

$$d(v, \tau) = \min_{v_i \in \tau} d(v, v_i). \tag{1}$$

Based on the above definitions, we propose the cornerstone of our learning framework: Trajectory-wise similarity. To define an effective similarity metric, the computation of Trajectory-wise similarity should have the property of commutativity. Moreover, it should also be negatively correlated to the actual distance between trajectories. Based on above consideration, given two trajectories τ_1 and τ_2, we formulate the definition of Trajectory-wise similarity between them (denoted as $\mathrm{Sim}(\tau_1, \tau_2)$) in (2):

$$\mathrm{Sim}(\tau_1, \tau_2) = \frac{\sum_{v_i \in \tau_1} e^{-d(v_i, \tau_2)}}{|\tau_1|} + \frac{\sum_{v_j \in \tau_2} e^{-d(v_j, \tau_2)}}{|\tau_2|}, \tag{2}$$

3.3 The GTS framework

With the definition in (2), we can then formally define the problem of trajectory similarity computation over the road network with the learning method formally as Definition 1.

Definition 1 Given a road network $G = (V, E)$ and a trajectory set $\mathbf{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, $\forall \tau_i \in \mathbf{T}$ it aims at finding a trajectory τ_j that minimizes $\mathrm{Sim}(\tau_i, \tau_j)$ or $\mathrm{Sim}_{ST}(\tau_1, \tau_2)$ and $i \neq j$.

To minimize $\mathrm{Sim}(\tau_i, \tau_j)$, we propose a two-step framework GTS shown in Figure 2. Compared with the end-to-end model architecture, the advantage of a two-step framework is that the training process is more stable and interpretable. The two steps will be detailed in Sections 4 and 5, respectively.

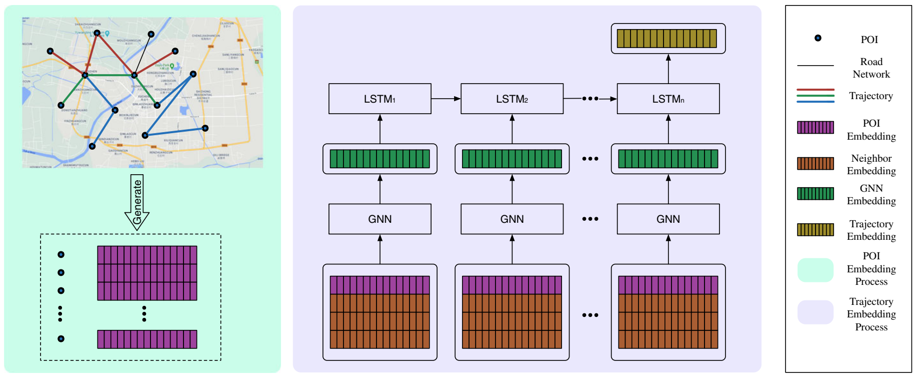


Figure 2 Overall Framework of GTS

There are mainly three challenges in the construction of the trajectory similarity measurement model.

1 How to utilize the information of spatial network in the perspective of trajectory, which could be different from grid-based methods.
2 How to represent the trajectory should be considered carefully, as the trajectory representations are closely related to the computation of trajectory similarity.
3 How to design the objective function to train the models, which should help learn the characteristics of collected trajectory datasets.

4 POI representation learning

In this section, we will introduce a new framework TraNode2Vec for learning the POI representation over the road network. We first give the big picture of the learning objective in Section 4.1 and then provide more technical details in Section 4.2.

4.1 Objective function

Since we targeted learning trajectory similarity over road network, the first step is to learn a high-quality presentation of POIs in the network. To this end, the learning objective should be with physical significance so as to include the information of trajectories into the POI representation. Unlike previous studies that utilized feature engineering methods based on expert knowledge, in our work, we aim at learning trajectory-aware POI embedding via the distance and similarity functions defined in Section 3. As a result, our approach can not only learn the topology of the road network but also fit the distribution of existing trajectories.

The first step towards this goal is to design a proper objective function that is consistent with the goal of learning trajectory similarity. According to our definition of Trajectory-wise similarity, it is essential to know the distance between POIs so as to estimate the similarity. Therefore, we aim at identifying a learning objective to help formulate a representation where the embedding vectors of nearby POIs or belonging to the same trajectories should also be closed to each other. In this target, there are two kinds of relationships between POIs. The first relationship is the topology relationship between POIs in the road network, which will influence the distances between them directly. The second one is whether two POIs

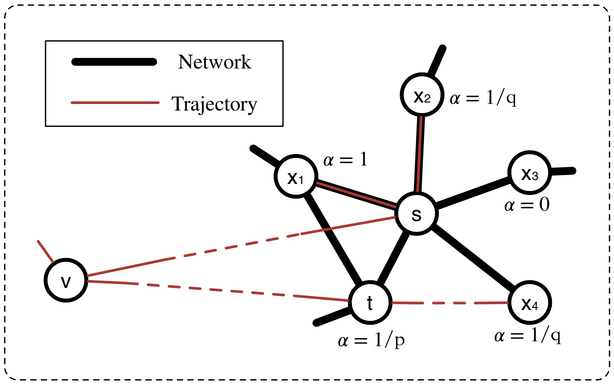


Figure 3 Illustration of the process of random walk with trajectory

belong to the same trajectory. These two relationships could be metaphysically described as the 'neighbors' of POIs, in which we can model them with existing embedding methods that are able to capture the property of neighbors.

To capture the property of 'neighbors', we could utilize the well-known Skip-gram [52] approach, which is originated in the field of natural language processing. It has been exploited in many applications to learn the representation of basic building blocks, such as the word embeddings in the article. Given the POI set $V = \{v_1, v_2, \ldots, v_m\}$, we could get the Skip-gram objective function for our task as (3)

$$\max_f \sum_{v \in V} \log P(N_s(v)|f(v)), \tag{3}$$

where $f : v \to \mathbb{R}^d$ is the encoder to map the POI into d dimension vectors, $P(\cdot)$ is probability function and $N_s(v) \subseteq V$ is the neighbors of POI v which is obtained via a random walk algorithm described later in Section 4.2. By optimizing this objective function, the learned embedding of given POI will have an explicit connection with those of its neighbors (Figure 3).

One limitation of the above objective function lies in the aspect of computational efficiency. To resolve this problem, we make a trade-off between accuracy and efficiency as follows: For the given POI v, we assume that all its neighbors in $N_s(v) \subseteq V$ are independent, which can reduce the computational complexity of the function $P(N_s(v)|f(v))$. Under this assumption, we have the objective function as (4):

$$P(N_s(v)|f(v)) = \prod_{v_i \in N_s(v)} P(v_i|f(v)). \tag{4}$$

Given POIs v_i and v, the value of $P(v_i|f(v))$ satisfies the following conditions: (i) the value of probability should be ranged in [0, 1]; and (ii) the sum of all probabilities for the given POI v should be 1. Thus we employ the *softmax* function that has been widely used to compute the probability in multiple classification problems. Then we have the explicit formulation of $P(v_i|f(v))$ as (5)

$$P(v_i|f(v)) = \frac{e^{f(v_i)\cdot f(v)}}{\sum_{v_j \in V} e^{f(v_j)\cdot f(v)}}. \tag{5}$$

However, the computation of $\sum_{v_j \in V} e^{f(v_j)\cdot f(v)}$ is time-consuming in the training process. The reason is that function $f(\cdot)$ will be updated after every epoch and the computation $\sum_{v_j \in V} e^{f(v_j)\cdot f(v)}$ cannot be reused. To solve this problem, the negative sampling method could be utilized to reduce the computation time by pair-wise loss.

4.2 Finding neighbors

Next, we discuss how to generate the set of neighbors $N_s(v)$ of given POI v in the objective function. Previous network embedding approaches, such as node2vec [53], find such neighbors by a random walk algorithm based on the topology structure of the graph. However, in our work, we need to not only consider the topology structure of the road network but also the given existing trajectories.

To address this issue, we employ a random walk algorithm to find $N_s(v)$ for given POI v the topology structure of the road network. Given a starting POI v and the length of walks n_w, the random walks method will generate a random path with starting POI v and n_w nodes. The generation process proceeds node-wise, and every node in the path is dependent on previous nodes. With the starting node $c_0 = v$, we generate the i-th node c_i for the random path as (6):

$$P(c_i = v_j | c_{i-1} = v_k) = \begin{cases} \frac{\pi_{v_j v_k}}{Z} & \text{if}(v_i, v_j) \in E, \\ 0 & \text{otherwise}, \end{cases} \tag{6}$$

where $\pi_{v_j v_k}$ is the transition probability from v_k to v_j and Z is a normalization constant.

As our goal is to learn a trajectory-aware POI representation, we need to reflect the influence of existing trajectories in the definition of transition probability in random walks. The probability of the next node in previous approaches such as node2vec is only decided by the node visited in the previous step. While this approach can capture the topology structure of the graph, it fails to take the trajectories into consideration in our problem setting. To ensure whether two POIs are in the same trajectory, we need to devise a random walk algorithm where the transition probability in each step is also influenced by the starting node of a trajectory.

To reach this goal, a straightforward solution for that is via a sampling-based method. Instead of only considering the last node, we choose the next node according to both the previous node and starting node. The relationship between the next and previous nodes could keep the topology structure of the graph. And the trajectory information will be maintained in the connection between the next and starting nodes.

Based on above discussion, we define the transition probability as $\pi_{v_j v_k} = \alpha(v_j, v_k) \cdot e^{-d(v_j, v_k)}$, where the distance in the road network between POIs in incurred in the term $d(v_j, v_k)$. And $\alpha(v_j, v_k)$ is probability of sampling defined in (7):

$$\alpha(v_j, v_k) = \begin{cases} \frac{1}{p} & \text{if } d_{v_j v_k} = 0 \text{ and } \{v_k, v\} \tilde{\in} \tau \\ 1 & \text{if } d_{v_j v_k} = 1 \text{ and } \{v_k, v\} \tilde{\in} \tau \\ \frac{1}{q} & \text{if } d_{v_j v_k} = 2 \text{ and } \{v_k, v\} \tilde{\in} \tau \\ 0 & \text{otherwise}, \end{cases} \tag{7}$$

where $d_{v_j v_k}$ is the path containing the least number of POIs between v_j and v_k, and the operation $\{v_k, v\} \tilde{\in} \tau$ means there is one trajectory $\tau \in \tau$ that $v_k \in \tau$ and $v \in \tau$.

In this way, we can ensure that all nodes in the path have a direct connection with the starting node v. Moreover, the topology structure of the road network could also be maintained in our sampling method.

5 Graph-based trajectory embedding

In this section, we introduce how to learn the trajectory embedding based on the POI representation learned previously.

5.1 GNN-based representation

Although the POI representation has captured certain information from the spatial network, we still need to incorporate it in the process of learning trajectory embedding. The reason is that we need the information of spatial network from different aspects: In the process of POI representation learning, the spatial network is exploited to make the embedding of connected POIs in the spatial network similar; while in the process of learning trajectory embedding, we need more information about node connections from the spatial network so as to make the trajectory representation more stable and robust.

The main challenge of the trajectory representation is that the search space is the enumeration of combination among all POIs. As a result, we cannot get sufficient training instances to cover all possible trajectory patterns and thus it results in the data sparsity problem. To overcome this problem, we utilize graph neural networks (GNN) to incorporate more information from the spatial network into each trajectory. The reason that we employ GNN here is that its Laplacian regularization term in the objective function of GNN could make the connected nodes keep the same labels and thus help alleviate the sparsity problem. Specifically, the computation of GNN incurs the information of neighbors for the given node.

In our task, we can deal with the sparsity by utilizing more POIs with the same set of trajectories. With the help of GNN, for each POI in one trajectory, we could impose all of its neighboring POIs to generate trajectory embedding. In this way, there will be a larger number of common POIs between similar trajectories. And for a given trajectory, it is easier to find the most similar trajectory in the training set to satisfy our goal in Definition 1.

To this end, we build the graph based on the POIs imposed for each POI in the trajectory as the input graph for GNN. Meanwhile, we could use the spatial network to construct the adjacent graph for GNN where only POIs in the spatial network will be connected in the adjacent graph. In this way, we can directly incorporate the spatial network to compute the trajectory representation.

Given the POIs $V = \{v_1, v_2, \ldots, v_m\}$ and weights set E, we construct the adjacent graph G as (8):

$$G_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

This adjacent graph G will be symmetric with diagonal element zeros.

Nevertheless, the adjacent graph G in this format still cannot accurately reflect the relationship between POIs. The reason is that values in G are not equal to the influence between the nodes. To address this issue, we construct the Laplacian matrix A in our work with a weight α to control the influence of neighbors as (9)

$$\mathbf{A} = \mathbf{I} + \alpha \text{Norm}\big(\mathbf{G}\big), \tag{9}$$

where $\mathbf{I}$ is the identity matrix and Norm($\mathbf{G}$) is the normalization function that every entry will be divided by the ℓ_1-norm of its corresponding row vector. We use $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ to denote the POI embeddings $V = \{v_1, v_2, \ldots, v_n\}$, where $\mathbf{p}_i = f(v_i)$ and $f(\cdot)$ is the POI embedding function learned by (3).

As every POI and its neighbors are a subset of V, we only need a subgraph from the graph G to generate the representation by GNN. Here we use the Graph Convolutional Network (GCN) [46] model as the encoder. Given a POI v_i and its neighbors $N(v_i) = \{v_{i_1}, v_{i_2}, \cdots, v_{i_k}\}$, the representation $\tilde{\mathbf{p}}_i$ is generated by a 1-layer GCN defined as (10)

$$\tilde{\mathbf{p}}_i = \mathbf{A}_i \mathbf{P}_i W, \tag{10}$$

where $\mathbf{A}_i$ is the row vector of the adjacent matrix $\mathbf{A}$ for the POI v_i, P_i is the stack of features $\{\mathbf{p}_i, \mathbf{p}_{i_1}; \mathbf{p}_{i_2}; \cdots ; \mathbf{p}_{i_k}\}$ for POI v_i and all its neighbors, and W is the learned parameter to project the combined POI features into a new space.

Once the GCN-based embeddings are obtained, we could use them to construct our trajectory embedding with the sequence model. Here we choose LSTM to fulfill this task. Specifically, we use the output of the last time step as the trajectory embedding. Given the network representation $\mathbf{P} = \{\tilde{\mathbf{p}}_1, \tilde{\mathbf{p}}_2, \ldots, \tilde{\mathbf{p}}_n\}$ of a trajectory τ, we generate its embedding E with LSTM as LSTM($\mathbf{P}$), where LSTM($\cdot$) is the operation of LSTM which will output the embedding vector in its last timestep.

5.2 Training

With the representation of all trajectories in the training set, we then specify the objective function. According to our definition of Trajectory-wise similarity, the goal of our task is to find the most similar trajectory for a given trajectory. To reach this goal, we use the dot product between the embedding vectors of trajectories to denote the similarity between them. Suppose the embedding vectors of trajectories τ_i and τ_j are E_i and E_j, the similarity score $\text{Sim}(\tau_i, \tau_j)$ can be computed as (11).

$$\text{Sim}(\tau_i \tau_j) = E_i^\top E_j \tag{11}$$

Since we aim at finding the most similar trajectory rather than calculating the exact similarity score, we do not need to perform the actual similarity computation in the process of testing. Therefore, we can decide the objective function in two ways. The first one is to apply the regression loss that uses the true similarity to optimize (11). The second one is using the pair-wise loss that maximizes the similarity between the most similar trajectory and the given one. In our framework, we use pair-wise loss as the objective function, which is also widely used in other ranking-based applications.

Then given the trajectory training set T^{tr}, we define the loss function as (12).

$$\max \sum_{\tau_i \in T^{tr}, \tau_j \in T^{tr} \setminus \{\tau_i', \tau_i\}} \mathbb{1}(\text{Sim}(\tau_i, \tau_i') > \text{Sim}(\tau_i, \tau_j)), \tag{12}$$

where τ_i' is the most similar trajectory for trajectory τ_i. And $\mathbb{1}$ is the indicator function that equals one if the condition satisfies, otherwise it will be zero.

For a given trajectory in (12), we need to compute similarities between all other trajectories and it. This process would be very time-consuming as the trajectory dataset usually varies largely. To reduce the computation time in the training process, we randomly sample one trajectory instead of traversing all trajectories for the given trajectory.

6 Spatio-temporal trajectory similarity

In this section, we introduce how to learn trajectory embedding based on both spatial and temporal information. We first define the spatio-temporal similarity metric for trajectories in Section 6.1. Next we introduce the way to utilize temporal information in Section 6.2. Finally we propose a spatio-temporal fusion technique to incorporate temporal information into LSTM cells when learning the trajectory representations from the sequence of POIs in Section 6.3.

6.1 Extended similarity with temporal information

For every POI $v_i \in \tau$, there is a timestamp $v_i.t$ associated with it which denotes the exact time when the object passed the POI v_i. In many applications, it is essential to also take the temporal information into consideration when We can come up with the temporal-aware POI-Trajectory distance following the idea of Section 3.2 as follows: Given a POI v and a trajectory τ, the definition POI-Trajectory distance between them can be calculated as (13):

$$d_T(v, \tau) = \min_{v_i \in \tau} \|v.t - v_i.t\|. \tag{13}$$

Based on the temporal POI-Trajectory distance, we then propose the temporal Trajectory-wise distance $\text{Sim}_T(\tau_1, \tau_2)$ between trajectories τ_1 and τ_2 as (14).

$$\text{Sim}_T(\tau_1, \tau_2) = \frac{\sum_{v_i \in \tau_1} e^{-d_T(v_i, \tau_2)}}{|\tau_1|} + \frac{\sum_{v_j \in \tau_2} e^{-d_T(v_j, \tau_2)}}{|\tau_2|}. \tag{14}$$

Finally, by combining the (2) and (14), we could get the spatial-temporal Trajectory-wise distance $\text{Sim}_{ST}(\tau_1, \tau_2)$ between trajectories τ_1 and τ_2 as (15):

$$\text{Sim}_{ST}(\tau_1, \tau_2) = \text{Sim}(\tau_1, \tau_2) + \text{Sim}_T(\tau_1, \tau_2). \tag{15}$$

6.2 Temporal-aware graph construction

We propose a new framework GTS$^+$ to minimize the value of $\text{Sim}_{ST}(\tau_1, \tau_2)$ by extending GTS with temporal information. To this end, it requires to first learn the temporal embedding of each POI, and then combine it with POI embedding to learn the representation of the whole trajectory. The first step towards this goal is to identify the temporal embedding of all POIs. To reach this goal, we first build a histogram vector for each POI based on the time it appeared in a trajectory. The closer two histogram vectors are, the more similar two POIs are in the temporal aspect. The first step to build such a histogram is to divide the whole time range into a fixed number of slots. Specifically, there are 24 slots where each slot is corresponding to 1-hour time range of a day. Then the temporal histogram of a POI becomes a 24-dimensional vector. Next we traverse the trajectory collection to build the histogram for all POIs: for each POI on a trajectory, we increase the value of corresponding dimension in the histogram according to its associated timestamp. Finally, we normalize the histogram vector by replacing the value of each dimension to the percentage of occurrence among all the time range, where the sum of values in all dimensions equal to 1. The diagram of this process could be found in Figure 4.

Next we construct the temporal aware graph based on above temporal histogram of all POIs. In the approach of learning trajectory representation from POI embeddings in Section 4, we construct the adjacent graph G by considering the structure of spatial network and learn the embedding with GNN to address the challenge of data sparsity problem

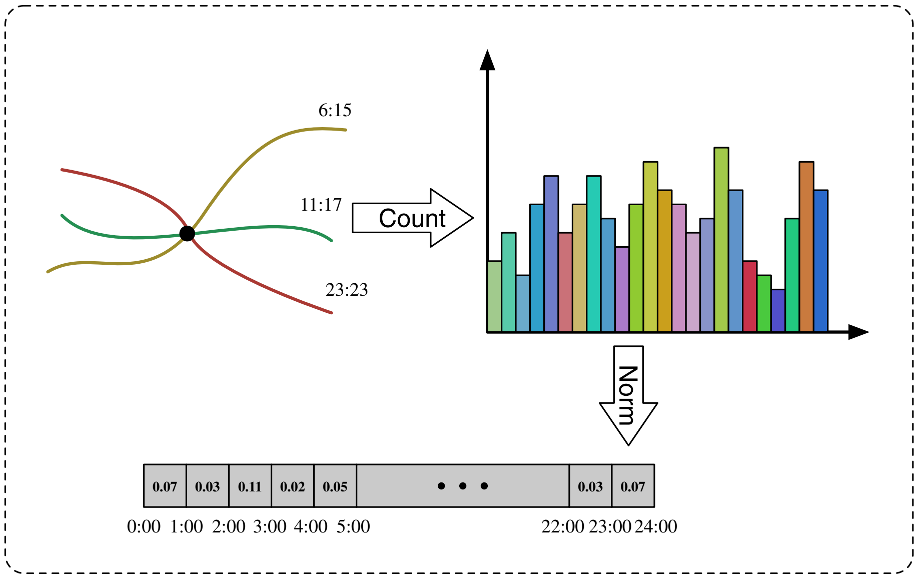


Figure 4 Example of Temporal Embedding

caused by large search space. Following this route, we then construct the temporal-aware adjacent graph so as to identify the neighbors for each POI from the temporal aspect. As the temporal histogram of POI is a fixed-length vector, the similarity between two POIs can be calculated as the cosine similarity between their histogram vectors. Then for each POI v, we add an edge between it and its top-K similarity POIs on the temporal aspect to construct the edge set E_T. The variable K can be regarded as a hyper-parameter and we set it as 20 empirically in this work. And the graph for temporal information $G_T = \langle V, E_T \rangle$ can be constructed in the similar way of that in Section 5.1: the set of vertices is the same with G_S and the set of edges is constructed in the way introduced above. Same with G_S, G_T is also symmetric with diagonal element zeros.

Now we have the spatial graph and temporal graph for POIs. Then the next problem is to make use of them to learn the POI embeddings. To solve this problem, we provide a joint training framework GTS$^+$ (Joint) that requires only one GNN model to learn the embedding of each POI. Recall that in order to build the graphs G_S and G_T, we need to construct the adjacent matrix M_S and M_T based on the edges in the two graphs, respectively. In the joint training framework, we construct only one graph G with the adjacent matrix $M = M_S \odot M_T$, where $\odot$ is the operation of element-wise product. Then we train one GNN model to learn the embedding of each POI based on G using the same way of that in Section 5.1. In this way, the POI embedding learned by the GNN model already carries both spatial and temporal information. Finally, we simple feed the POIs on a trajectory into an original LSTM model as shown in Figure 2 to learn the trajectory representation.

6.3 Fusing spatial and temporal features

Though the effectiveness of GTS$^+$ (Joint) framework introduced above, its performance might suffer from the lack of interaction between the spatial and temporal information. To simultaneously learn the characteristics of trajectory from these two kinds of information, we propose a new ST-LSTM model instead of using the original LSTM to generate the trajectory embedding. Specifically, we design a temporal gate to control the influence of temporal information on trajectory representation. In every time step, the temporal gate can select key information in the context according to the current POI embedding input and temporal representations and filter useless information. Meanwhile, the vectors of temporal information can influence the process of sequence modeling and keep the important information in the current hidden states.

To describe the structure of ST-LSTM, we start from reviewing the structure of original LSTM model. In each time step t, it uses the embedding vector $\boldsymbol{x}_t$ of the context POI embedding and temporal embedding vector $\boldsymbol{T}_t$ corresponding to current time step as input. The temporal embedding vector is the one-hot feature which denotes the current time slot of the input. It has three gates: the input gate $\boldsymbol{i}_t$, forget gate $\boldsymbol{f}_t$ and output gate $\boldsymbol{o}_t$ respectively. The basic update equations of LSTM are as follows:

$$\begin{pmatrix} \mathrm{i}_t \\ \mathrm{f}_t \\ \tilde{\mathrm{c}}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \tanh \end{pmatrix} (\mathrm{W}[\mathrm{h}_{t-1}, \mathrm{x}_t] + \mathrm{b}), \tag{16}$$

$$\mathrm{c}_t = \mathrm{f}_t \odot \mathrm{c}_{t-1} + \mathrm{i}_t \odot \tilde{\mathrm{c}}_t, \tag{17}$$

$$\mathrm{o}_t = \sigma(\mathrm{W}_o[\mathrm{h}_{t-1}, \mathrm{x}_t] + \mathrm{b}_o), \tag{18}$$

$$\mathrm{h}_t = \mathrm{o}_t \odot \tanh(\mathrm{c}_t) \tag{19}$$

where $\boldsymbol{h}_t$ and $\boldsymbol{c}_t$ are the cell states.

Based on above update operations of LSTM, we add a *temporal gate* to each LSTM cell and propose the ST-LSTM model as shown shown in Figure 5. The temporal gate is computed by a POI's spatial embedding $\boldsymbol{x}_t$ and its temporal embedding $\boldsymbol{T}_t$. It controls the influence of temporal specificity on trajectory representation for the current input $\boldsymbol{x}_t$. It can be formalized as follows:

$$\boldsymbol{g}_t = \sigma(\boldsymbol{W}_{xg}[\boldsymbol{x}_t \| \boldsymbol{T}_t] + \boldsymbol{b}_g) \tag{20}$$

where $[\cdot\|\cdot]$ is the operation of concatenation, $\boldsymbol{T}_t \in R^{dt}$ is temporal embedding; $\boldsymbol{W}_{xg} \in R^{d\times(dx+dt)}$ is the weighted matrix to be learned; $\boldsymbol{b}_g \in R^{dx+dt}$ is bias. Parameter dt stands

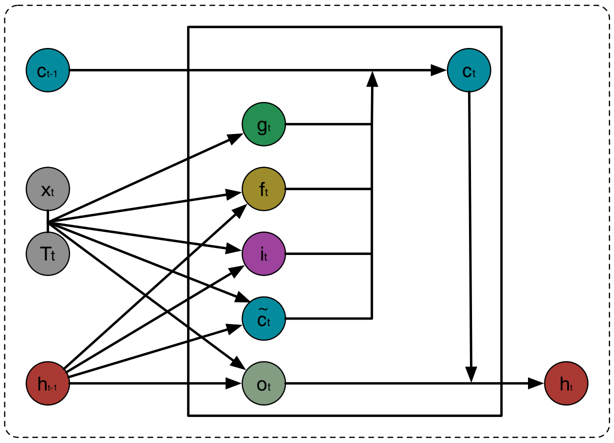


Figure 5 The Structure of ST-LSTM Cell

for the dimension of temporal vector. dx is the dimension of POI embedding and d is the dimension of hidden layer.

Correspondingly, we replace (17) and (18) with (21) and (22) in a cell of ST-LSTM.

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t \odot \tilde{\boldsymbol{c}}_t, \tag{21}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, [\boldsymbol{x}_t \| \boldsymbol{T}_t]] + \boldsymbol{b}_o). \tag{22}$$

To apply ST-LSTM in the framework, we just need to use two separate GNN model to learn the spatial and temporal embedding of each POI, respectively. Then we replace the LSTM in Figure 2 with ST-LSTM and use above spatial and temporal embedding of POIs on a trajectory as input.

7 Experiments

In this section, we will demonstrate the effectiveness of our proposed methods by conducting an extensive set of experiments. The experiment setup is introduced in Section 7.1. Results and the corresponding analysis are presented by comparing with 4 state-of-the-art baselines in Section 7.2. Moreover, we conduct an ablation study as well as parameter analysis in Sections 7.3 and 7.4. Finally, the results of spatio-temporal trajectory similarity computation are reported in Section 7.5.

7.1 Experiment setup

7.1.1 Dataset

For the road network, we use two spatial networks from different cities. One is from the city of Beijing, namely the Beijing Road Network (BRN). The other is from the city of New York, namely the New York Road Network (NRN)[1]. There are 28,342 POIs and 27,690 edges in the BRN dataset; and 95,581 POIs and 260,855 edges in the NRN dataset.

For trajectories in BRN, we use the taxi driving data [54] from the T-drive project[2]. The taxi trajectories in BRN are collected by taxi id, and the time range of one trajectory may last several days. So we split these trajectories by hour, then we could get 5,621,428 trajectories in total. The average length of these trajectories is 25 by filtering the abnormal ones. For trajectories in NRN, we use the taxi driving data from New York. There are 697,622,444 trips in the original dataset, and we randomly sample a subset of them to generate the trajectory dataset. After pre-processing, there are 10,541,288 trajectories in our experiments and the average length of them is 38. The details are summarized in Table 1. For both trajectory datasets, we randomly split them into training, evaluation, and testing set with the ratio 20%, 10%, and 70%.

To evaluate the spatio-temporal based similarity measurement, we still use the above two datasets but perform pre-processing to incorporate temporal information as following: we create the ground truth based on the similarity calculated from (2) and (14) that takes both spatial and temporal information into consideration.

[1] https://publish.illinois.edu/dbwork/open-data/

[2] https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/

Table 1 Statistics of datasets

	Beijing	New York
#POIs	28,342	95,581
#Edge	27,690	260,855
#Trajectory	5,621,428	10,541,288
Ave Length	25	38

7.1.2 Parameter setting

The details of the hyper-parameter setting are as follows. The dimension of POI embedding is set as 128. And the parameters p and q in the sampling strategy proposed in Section 4.2 are both set as 1. We conduct a grid search to decide on the following hyper-parameters: The dimension of GNN embedding is selected from the range {32, 64, 128, 256}; The dimension of trajectories is also from the range {32, 64, 128, 256} in a similar manner to select the dimension of GNN embedding. The parameter α to control the influence of neighbors in GNN is selected from the range of [0 : 0.1 : 1]. We use Adam [55] as the optimizer to train our proposed methods. The learning rate of Adam is set as 0.001.

7.1.3 Evaluation metric

Following previous studies, we use the hitting ratio in the top K list (HR@K) as the metric in our experiments to show the performance of different methods. The definition of HR@K is set as (23)

$$HR@K = \frac{1}{|T^{te}|} \sum_{\tau \in T^{te}} \frac{|L_\tau^T @K \cap L_\tau^R|}{|L_\tau^R|} \tag{23}$$

where T^{te} is the test set of trajectories, $|\cdot|$ is the set cardinality, $L_\tau^T @K$ is the list of predicted most similar trajectories for a given trajectory τ with length K, and L_τ^R is the set of most similar trajectory in the training set for the given trajectory τ where $L_\tau^R = \{\tau'\}$.

7.1.4 Baseline

As our work is the first deep learning-based method for trajectory similarity over the spatial network, we extend four state-of-the-art methods to similar research problems in our experiments as baselines to show the performance of our method. The details of these methods are summarized as follows:

- Traj2vec [9]: They use a sequence-to-sequence model to learn the representation of the trajectory. Mean square error is utilized as the loss function to optimize their method.
- Siamese [56]: This method is a time series learning approach based on the Siamese network. They use the cross-entropy as the objective function to train the framework. We set the backbone of their Siamese network with LSTM and use the similar setting as [24] to support trajectory similarity computation.
- NeuTraj [24]: This method revised the structure of LSTM to learn the embeddings of the grid in the process of training their framework. To support our task, we replace the grid with POIs in their framework.

- Traj2SimVec [6]: This method employs a new loss for learning the trajectory similarity by point matching. We apply their model to the road network in a similar way to learn the similarity between trajectories.

For the evaluation of spatio-temporal based similarity measurement, we extend the above approaches in a similar way: we initialize the temporal embedding with one-hot representation and feed it into an MLP layer. Then we concatenate the output of the MLP layer with that of the POI embedding to formulate the input of the above models.

7.2 Experimental results

The experiment results on the two datasets could be found in Tables 2 and 3. From these results, we present our observations and corresponding analysis as follows.

Firstly, our method outperforms all other methods on all metrics and this could verify the superiority of our method. The main reason is that our framework can utilize the information from the road network, where others only consider the information of the grid.

Specifically, our method significantly outperforms NeuTraj. The improvements come from two aspects: (i) The embedding generation of POIs is independent from the trajectory similarity learning in our method, where NeuTraj learns them simultaneously; and (ii) NeuTraj utilizes the regression loss to learn the actual similarity between two trajectories, while GTS can learn the partial ordering relationship between trajectories. These two factors both improve the performance of trajectory similarity computation.

Moreover, one additional reason why GTS is better than Traj2SimVec is that we use the dot product of embedding vectors to compute the similarity between two trajectories, where Traj2SimVec uses the L_2-norm of absolute the difference between embedding vectors. Using dot product to compute the similarity is inspired by collaborative filtering in the field of recommendation, which has been proved more effective than the linear operation as it could propagate the information between indirectly connected samples efficiently.

Finally, the advantage of GTS over Siamese lies in that the cross-entropy loss in Siamese cannot learn the partial ordering relationship between similar and dissimilar trajectories. And the objective of Siamese is to make the similarity between the similar trajectories as large as possible. Nevertheless, this optimization process will lead to overfitting. At the same time, the loss function of GTS can avoid this problem as its value will be zero if the predicted similarity between similar trajectories is larger than that between dissimilar ones.

7.3 Ablation study

The main components and contributions of our work are that we propose a new way to generate the POI embeddings and utilize GNN to learn the trajectory embeddings. To show

Table 2 Results on Beijing dataset (Bold numbers indicate the best performance.)

Method	HR@1	HR@5	HR@10	HR@20	HR@50
Traj2vec	5.82%	10.57%	18.64%	28.83%	40.07%
Siamese	6.33%	13.25%	20.17%	32.61%	45.58%
NeuTraj	7.72%	19.78%	27.54%	39.63%	53.57%
Traj2SimVec	7.81%	20.42%	29.17%	40.14%	56.75%
GTS	**9.21%**	**25.00%**	**35.48%**	**48.07%**	**66.12%**

Table 3 Results on New York dataset (Bold numbers indicate the best performance.)

Method	$HR@1$	$HR@5$	HR@10	HR@20	HR@50
Traj2vec	4.95%	9.33%	16.13%	24.57%	37.24%
Siamese	5.23%	11.12%	18.75%	27.74%	42.16%
NeuTraj	6.15%	15.57%	23.28%	30.18%	48.43%
Traj2SimVec	6.31%	17.03%	26.46%	32.52%	50.55%
GTS	**8.43%**	**21.64%**	**32.53%**	**41.69%**	**58.17%**

the effects of these two techniques in our framework, we give the ablation experiment in Table 4. The settings of these methods are summarized as follows:

- **GTS/POI**: In this method, we did not utilize our POI embedding as the input for the trajectory similarity model. The embedding matrix is randomly initialized and trained along with other components in the framework.
- **GTS/GNN**: Instead of applying GNN on POIs for further encoding, we just use our POI embedding as the input for the LSTM to get the trajectory embedding.

From the results in Table 4, we can obtain the following conclusions and analysis:

First, we find that utilizing our POI embedding could significantly improve the performance. As the objective function of the trajectory, similarity cannot directly constrain the POI embedding in GTS/POI, and the POI embedding learned in this process will be random without explainable physical significance. Then the relationship between POI embeddings will be uncertain, and the combinations of POIs cannot reflect the spatial topology of existing trajectories on the spatial network. The two-step strategy for the trajectory similarity learning in our framework could address this problem: The POI embedding learned in the first step would include the information of both the spatial network and existing trajectories in the training data. Then the combinations of them will lead to more reasonable trajectory patterns.

Moreover, we observe that the GNN can definitely improve the performance of our framework. By applying GNN on POI embeddings, it provides richer information of the spatial network. The reason is that the adjacent graph in GNN has the same topology structure as the spatial network. Moreover, the data sparsity problem in the trajectory dataset can also be alleviated with the help of GNN. For each node in the network, the GNN can help impose all of its connected POIs to generate trajectory embedding. In this way, the number of common POIs between similar trajectories will be larger. For a given trajectory, it is easier to find its most similar trajectory in the training set.

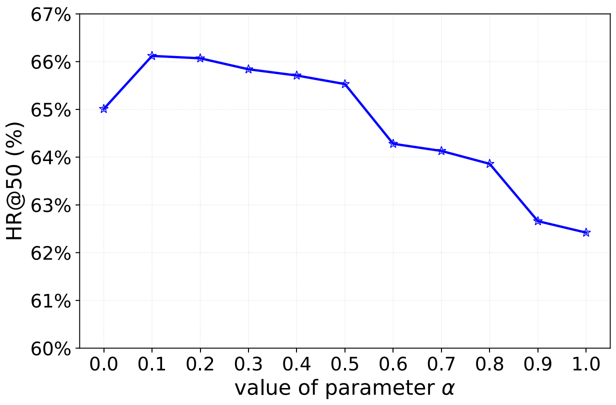
Table 4 Ablation Experiment (Bold numbers indicate the best performance.)

Method	$Pr@1$	$Pr@5$	Pr@10	Pr@20	Pr@50
GTS/POI	8.07%	21.38%	30.54%	41.01%	58.55%
GTS/GNN	8.80%	24.15%	33.28%	45.57%	63.31%
GTS	**9.21%**	**25.00%**	**35.48%**	**48.07%**	**66.12%**

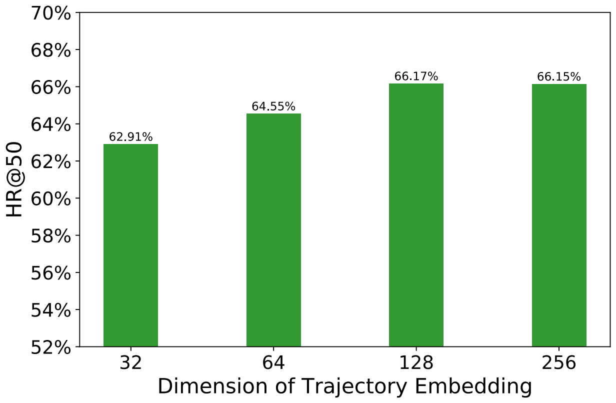
7.4 Parameter analysis

We also conduct the parameter analysis to provide more insights into some components in our framework. From the results in Figure 6, we have the following observations:

As shown in Figure 6(a), we could see that the results vary greatly with different values of parameter α. This serves as evidence that the usage of GNN has a significant influence on the performance of learning trajectory similarity. GNN would incur the information of neighbors for the given sample. The performance is the best when $\alpha = 0.1$, which means the relationship between a given POI and its neighbors achieves the best state for the trajectory similarity learning. When $\alpha = 0.0$, the GNN will be equivalent to MLP, where there are no neighbors for any given POI. By comparing the results between $\alpha = 0.1$ and $\alpha = 0.0$, we could conclude that for a given POI, gathering its neighborhood information in an appropriate way will help improve the performance. However, when the value of α is too large, the performance will become worse. The main reason is that in this case, the weights in the adjacent graph cannot reflect the actual relationship between POIs.



(a) Results of parameter α



(b) Results of trajectory dimension

Figure 6 Results of different parameters

The effect of the trajectory embedding dimension could be found in Figure 6(b). It is obvious that the dimension of trajectory embedding decides how much information they can contain in the training process. If the dimension is too small, it will lead to the underfitting problem, where the model cannot fit the training dataset well. Meanwhile, if the value is too large, it may cause the overfitting problem, where the model cannot achieve good performance on the test dataset. The overfitting problem could be resolved by many other technologies, such as our pair-wise loss and GNN component. And that's the reason why we could obtain a good performance when the dimension of trajectory embedding is large.

7.5 Evaluate the GTS$^+$ framework

Finally, we report the results of computing trajectory similarity-based both spatial and temporal information. Based on the techniques introduced in Section 6, we proposed 3 approaches:

- GTS$^+$ (Concat) is the method that first learns the temporal embedding of each POI with GAT based on the graph constructed in Section 6.2 and then composes the embedding of each POI by concatenating its spatial and temporal embeddings. Finally, the trajectory representation is learned with the original LSTM network.
- GTS$^+$ (Joint) is the method that jointly learns the spatial and temporal information of each POI with one GAT network introduced in the last paragraph of Section 6.2.
- GTS$^+$ (ST-LSTM) is the method that uses spatial and temporal embedding of each POI as the input and employs a ST-LSTM network introduced in Section 6.3 to learn the trajectory representation.

The experimental results are shown in Table 5. We have the following observations: Firstly, the three GTS$^+$ based approaches have a better overall performance than state-of-the-art methods. The reason is that previous approaches fail to capture both the structure of the spatial network and temporal representation. Meanwhile, with our graph-based method, GTS$^+$ can learn such information in the learned trajectory embedding. Thus it can achieve promising results in spatial-temporal based trajectory similarity computation.

Secondly, GTS$^+$ (Contact) performs the worst among the three approaches. The reason is that it cannot properly merge spatial and temporal information. Since the two embeddings are learned in two different vector spaces, directly concatenating them might lead to some noises. The GTS$^+$ (ST-LSTM) and GTS$^+$ (Joint) methods significantly outperform GTS$^+$

Table 5 Evaluation Results: Spatial and Temporal Information (Bold numbers indicate the best performance.)

	New York					Beijing				
	HR@1	HR@5	HR@10	HR@20	HR@50	HR@1	HR@5	HR@10	HR@20	HR@50
Traj2vec	3.11	9.83	11.95	18.63	28.93	2.76	8.63	12.81	17.37	26.58
Siamese	3.28	10.56	13.22	19.47	30.28	2.98	8.78	13.45	18.44	28.23
NeuTraj	3.75	11.21	15.52	21.13	33.89	3.43	9.54	14.21	20.51	31.90
Traj2SimVec	3.86	11.95	16.41	23.77	36.53	3.54	9.66	14.73	21.07	33.84
GTS$^+$ (Concat)	4.60	13.13	19.77	28.32	43.07	4.08	11.87	17.31	25.83	41.77
GTS$^+$ (ST-LSTM)	6.68	14.51	21.84	33.17	46.53	5.14	14.57	20.58	**26.54**	**42.54**
GTS$^+$ (Joint)	**6.74**	**14.94**	**22.11**	**33.54**	**46.88**	**5.15**	**14.63**	**20.64**	26.48	42.10

(Contact) as they employ reasonable approaches for information fusion. For example, on the New York dataset for the results of HR@50, the results of Concat, ST-LSTM and Joint is 43.07, 46.53, 46.88, respectively.

Thirdly, the performance of GTS^+ (ST-LSTM) is comparable with that of GTS^+ (Joint). But the overall performance of GTS^+ (Joint) is slightly better. The reason might be that although ST-LSTM can effectively make use of the spatial and temporal embedding of POIs, such embeddings are still learned from two different GAT models. Due to the potential issues in data sparsity and vector space alignment, the quality of spatial and temporal embedding themselves is somewhat limited. At the same time, GTS^+ (Joint) can combine the spatial and temporal information with only one GAT in an earlier stage when learning the POI embedding. It is proved to be a simple but effective approach for this kind of task.

8 Conclusion

In this paper, we proposed the first deep learning-based framework for trajectory similarity computation over the spatial network. Compared with existing approaches, our framework is able to capture underlying route information of the trajectories by considering the structure of the spatial network, thus being robust to the number of available training instances and noisy points introduced by system errors. To this end, our GTS framework first employs a trajectory-aware random walk scheme to learn the representation of each POI in the spatial network. Then it utilizes a GNN-based model combined with LSTM to learn the trajectory representation for similarity computation. We further extend the framework to incorporate temporal information into the similarity computation and design a joint training approach. Experimental results on several popular real-life datasets show the superiority of our framework in terms of effectiveness.

Acknowledgements This work was supported by the NSFC (No. U21B2046). We acknowledge the editorial committee's support and all anonymous reviewers for their insightful comments and suggestions, which improved the content and presentation of this manuscript.

Author Contributions All authors contributed to the study conception and model design. Silin Zhou and Pen Han worked on the full manuscript. The first draft of the manuscript was written by Silin Zhou and Pen Han. Silin Zhou, Peng Han, and Lisi Chen wrote Sections 1–2. Silin Zhou and Pen Han prepared Sections 3–5. The experimental study was conducted by Di Yao and Xiangliang Zhang. All authors commented on previous versions of the manuscript. All authors have proofread and approved the final manuscript.

Funding This work was supported by the NSFC (No. U21B2046).

Data Availability All datasets used in this paper are open datasets.

Declarations

Competing interests We declare that authors have no known competing interests or personal relationships that might be perceived to determine the discussion report in this paper.

References

1. Chen, L., Shang, S., Guo, T.: Real-time route search by locations. In: AAAI, pp. 574–581 (2020)

2. Chen, L., Shang, S., Jensen, C.S., Yao, B., Zhang, Z., Shao, L.: Effective and efficient reuse of past travel behavior for route recommendation. In: SIGKDD, pp. 488–498 (2019)
3. Shang, S., Chen, L., Jensen, C.S., Wen, J., Kalnis, P.: Searching trajectories by regions of interest. IEEE Trans. Knowl. Data Eng. **29**(7), 1549–1562 (2017)
4. Zheng, K., Shang, S., Yuan, N.J., Yang, Y.: Towards efficient search for activity trajectories. In: ICDE, pp. 230–241 (2013)
5. Shang, S., Ding, R., Yuan, B., Xie, K., Zheng, K., Kalnis, P.: User oriented trajectory search for trip recommendation. In: EDBT, pp. 156–167 (2012)
6. Zhang, H., Zhang, X., Jiang, Q., Zheng, B., Sun, Z., Sun, W., Wang, C.: Trajectory similarity learning with auxiliary supervision and optimal matching. In: IJCAI, pp. 3209–3215 (2020)
7. Chen, L., Shang, S., Feng, S., Kalnis, P.: Parallel subtrajectory alignment over massive-scale trajectory data. In: IJCAI, pp. 3613–3619 (2021)
8. Shang, S., Chen, L., Zheng, K., Jensen, C.S., Wei, Z., Kalnis, P.: Parallel trajectory-to-location join. IEEE Trans. Knowl. Data Eng. **31**(6), 1194–1207 (2019)
9. Yao, D., Zhang, C., Zhu, Z., Hu, Q., Wang, Z., Huang, J., Bi, J.: Learning deep representation for trajectory clustering. Expert Syst. J. Knowl. Eng 35(2) (2018)
10. Zheng, K., Zheng, Y., Yuan, N.J., Shang, S., Zhou, X.: Online discovery of gathering patterns over trajectories. IEEE Trans. Knowl. Data Eng., 1974–1988 (2014)
11. Zheng, K., Zheng, Y., Yuan, N.J., Shang, S.: On discovery of gathering patterns from trajectories. In: ICDE, pp. 242–253 (2013)
12. Zhao, Y., Shang, S., Wang, Y., Zheng, B., Nguyen, Q.V.H., Zheng, K.: REST: a reference-based framework for spatio-temporal trajectory compression. In: SIGKDD, pp. 2797–2806 (2018)
13. Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Lee, J., Jurdak, R.: A novel framework for online amnesic trajectory compression in resource-constrained environments. IEEE Trans. Knowl. Data Eng., 2827–2841 (2016)
14. Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Jurdak, R.: Bounded quadrant system: error-bounded trajectory compression on the go. In: ICDE, pp. 987–998 (2015)
15. Song, R., Sun, W., Zheng, B., Zheng, Y.: PRESS: A novel framework of trajectory compression in road networks. PVLDB **7**(9), 661–672 (2014)
16. Yang, C., Chen, L., Wang, H., Shang, S.: Towards efficient selection of activity trajectories based on diversity and coverage. In: AAAI, pp. 689–696 (2021)
17. Liu, Y., Ao, X., Dong, L., Zhang, C., Wang, J., He, Q.: Spatiotemporal activity modeling via hierarchical cross-modal embedding. IEEE Trans. Knowl. Data Eng. **34**(1), 462–474 (2022)
18. Atluri, G., Karpatne, A., Kumar, V.: Spatio-temporal data mining: A survey of problems and methods. ACM Comput. Surv. **51**(4), 83–18341 (2018)
19. Yi, B., Jagadish, H.V., Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In: ICDE, pp. 201–208 (1998)
20. Vlachos, M., Gunopulos, D., Kollios, G.: Discovering similar multidimensional trajectories. In: ICDE, pp. 673–684 (2002)
21. Chen, L., Ng, R.T.: On the marriage of lp-norms and edit distance. In: VLDB, pp. 792–803 (2004)
22. Chen, L., Özsu, M.T., Oria, V.: Robust and Fast Similarity Search for Moving Object Trajectories. In: SIGMOD, pp. 491–502 (2005)
23. Li, X., Zhao, K., Cong, G., Jensen, C.S., Wei, W.: Deep representation learning for trajectory similarity computation. In: ICDE, pp. 617–628 (2018)
24. Yao, D., Cong, G., Zhang, C., Bi, J.: Computing trajectory similarity in linear time: a generic seed-guided neural metric learning approach. In: ICDE, pp. 1358–1369 (2019)
25. Li, X., Cong, G., Cheng, Y.: Spatial transition learning on road networks with deep probabilistic models. In: ICDE, pp. 349–360 (2020)
26. Shang, S., Chen, L., Wei, Z., Jensen, C.S., Zheng, K., Kalnis, P.: Trajectory similarity join in spatial networks. PVLDB **10**(11), 1178–1189 (2017)
27. Chen, Z., Shen, H.T., Zhou, X., Zheng, Y., Xie, X.: Searching trajectories by locations: an efficiency study. In: SIGMOD, pp. 255–266 (2010)
28. Shang, S., Ding, R., Zheng, K., Jensen, C.S., Kalnis, P., Zhou, X.: Personalized trajectory matching in spatial networks. VLDB J., 449–468 (2014)
29. Han, P., Li, Z., Liu, Y., Zhao, P., Li, J., Wang, H., Shang, S.: Contextualized point-of-interest recommendation. In: IJCAI, pp. 2484–2490 (2020)
30. Tang, J., Wang, K.: Personalized Top-N sequential recommendation via convolutional sequence embedding. In: WSDM, pp. 565–573 (2018)
31. Feng, S., Cong, G., An, B., Chee, Y.M.: Poi2vec: Geographical latent representation for predicting future visitors. In: AAAI, pp. 102–108 (2017)

32. Atev, S., Miller, G., Papanikolopoulos, N.P.: Clustering of vehicle trajectories. IEEE Trans. Intell. Trans. Syst. **11**(3), 647–657 (2010)
33. Rakthanmanon, T., Campana, B.J.L., Mueen, A., Batista, G.E.A.P.A., Westover, M.B., Zhu, Q., Zakaria, J., Keogh, E.J.: Searching and mining trillions of time series subsequences under dynamic time warping. ACM SIGKDD, pp. 262–270 (2012)
34. Shang, S., Chen, L., Wei, Z., Jensen, C.S., Zheng, K., Kalnis, P.: Parallel trajectory similarity joins in spatial networks. VLDB J., 395–420 (2018)
35. Wang, S., Bao, Z., Culpepper, J.S., Xie, Z., Liu, Q., Qin, X.: Torch: a search engine for trajectory data. In: SIGIR, pp. 535–544 (2018)
36. Chen, L., Shang, S., Jensen, C.S., Yao, B., Kalnis, P.: Parallel semantic trajectory similarity join. In: ICDE, pp. 997–1008 (2020)
37. Yang, J., Zhang, Y., Zhou, X., Wang, J., Hu, H., Xing, C.: A hierarchical framework for top-k location-aware error-tolerant keyword search. In: ICDE, pp. 986–997 (2019)
38. Wu, J., Zhang, Y., Wang, J., Lin, C., Fu, Y., Xing, C.: Scalable metric similarity join using mapreduce. In: ICDE, pp. 1662–1665 (2019)
39. Zhang, Y., Wu, J., Wang, J., Xing, C.: A transformation-based framework for KNN set similarity search. IEEE Trans. Knowl. Data Eng. **32**(3), 409–423 (2020)
40. Wang, J., Lin, C., Li, M., Zaniolo, C.: Boosting approximate dictionary-based entity extraction with synonyms. Inf. Sci. **530**, 1–21 (2020)
41. Zheng, K., Zheng, Y., Xie, X., Zhou, X.: Reducing uncertainty of low-sampling-rate trajectories. In: ICDE, pp. 1144–1155 (2012)
42. Han, P., Wang, J., Yao, D., Shang, S., Zhang, X.: A graph-based approach for trajectory similarity computation in spatial networks. In: SIGKDD, pp. 556–564 (2021)
43. Han, P., Shang, S., Sun, A., Zhao, P., Zheng, K., Kalnis, P.: AUC-MF: Point of interest recommendation with AUC maximization. In: ICDE, pp. 1558–1561 (2019)
44. Zhao, K., Zhang, Y., Yin, H., Wang, J., Zheng, K., Zhou, X., Xing, C.: Discovering subsequence patterns for next POI recommendation. In: IJCAI, pp. 3216–3222 (2020)
45. Yao, D., Cong, G., Zhang, C., Meng, X., Duan, R., Bi, J.: A linear time approach to computing time series similarity based on deep metric learning. IEEE Transactions on Knowledge and Data Engineering (2020)
46. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
47. Li, J., Rong, Y., Cheng, H., Meng, H., Huang, W., Huang, J.: Semi-supervised graph classification: a hierarchical graph perspective. In: WWW, pp. 972–982 (2019)
48. Chen, Y., Wu, L., Zaki, M.J.: Reinforcement learning based graph-to-sequence model for natural question generation. In: ICLR (2020)
49. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. In: ICLR (2014)
50. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS, pp. 1024–1034 (2017)
51. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. In: ICLR (2018)
52. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed Representations of Words and Phrases and Their Compositionality. In: NIPS, pp. 3111–3119 (2013)
53. Grover, A., Leskovec, J.: Node2vec: Scalable Feature Learning for Networks. In: ACM SIGKDD, pp. 855–864 (2016)
54. Zheng, Y., Xie, X., Ma, W.: Geolife: a collaborative social networking service among user, location and trajectory. IEEE Data Eng. Bull. **33**(2), 32–39 (2010)
55. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. ICLR (2015)
56. Pei, W., Tax, D.M.J., van der Maaten, L.: Modeling time series similarity with siamese recurrent networks. CoRR 1603.04713 (2016)