TrajGAT: A Graph-based Long-term Dependency Modeling Approach for Trajectory Similarity Computation

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

Haonan Hu
Institute of Computing Technology, Chinese Academy of Sciences
huhaonan1007@outlook.com

Lun Du
Microsoft Research Asia
lun.du@microsoft.com

Gao Cong
Nanyang Technological University
gaocong@ntu.edu.sg

Shi Han
Microsoft Research Asia
shihan@microsoft.com

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

ABSTRACT

Computing trajectory similarities is a critical and fundamental task for various spatial-temporal applications, such as clustering, prediction, and anomaly detection. Traditional similarity metrics, *i.e.* DTW and Hausdorff, suffer from quadratic computation complexity, leading to their inability on large-scale data. To solve this problem, many trajectory representation learning techniques are proposed to approximate the metric space while reducing the complexity of similarity computation. Nevertheless, these works are designed based on RNN backend, resulting in a serious performance decline on long trajectories. In this paper, we propose a novel graph-based method, namely TrajGAT, to explicitly model the hierarchical spatial structure and improve the performance of long trajectory similarity computation. TrajGAT consists of two main modules, *i.e.*, graph construction and trajectory encoding. For graph construction, TrajGAT first employs PR quadtree to build the hierarchical structure of the whole spatial area, and then constructs a graph for each trajectory based on the original records and the leaf nodes of the quadtree. For trajectory encoding, we replace the self-attention in Transformer with graph attention and design an encoder to represent the generated graph trajectory. With these two modules, TrajGAT can capture the long-term dependencies of trajectories while reducing the GPU memory usage of Transformer. Our experiments on two real-life datasets show that TrajGAT not only improves the performance on long trajectories but also outperforms the state-of-the-art methods on mixture trajectories significantly.

CCS CONCEPTS

• **Information systems** → **Geographic information systems**; *Query representation.*

KEYWORDS

trajectory similarity computation, long-term dependency, graph attention network, transformer

ACM Reference Format:
Di Yao, Haonan Hu, Lun Du, Gao Cong, Shi Han, and Jingping Bi. 2022. TrajGAT: A Graph-based Long-term Dependency Modeling Approach for Trajectory Similarity Computation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539358

1 INTRODUCTION

Trajectory similarity computation is an essential task in spatial-temporal data analysis. Classic similarity measures, such as DTW[33], Hausdorff[2] and ERP[6], have been presented to quantify the intrinsic similarities of trajectories, which can be applied in trajectory clustering[31], location prediction[17, 29], anomaly detection[15, 18, 40], *etc.* The quadratic complexity of these measures, on the other hand, restricts their application to large-scale trajectory analysis and is the *de facto* bottleneck for computing trajectory similarities.

To tackle this issue, various strategies for approximate similarity measures have been proposed, including locality sensitive hashing (LSH) for Hausdorff [8] and constraining the wrapping window for DTW [20]. Unfortunately, these techniques are designed for one specific measure and not applicable to other measures. Deep representation learning(DRL) methods[12, 14, 16, 25, 30, 35, 36] have been successfully applied for trajectory similarity computation in recent years. To approximate the similarity, they represent trajectories with vectors and learn a metric space of vectors. Comparing with traditional approximation strategies, DRL methods are fast yet general for a variety of similarity metrics. Nevertheless, we evaluate existing DRL methods on top-K similarity search and find their performances dramatically decrease on long trajectories. As shown in Figure 1 (a), the top 10 hitting-ratios of the state-of-the-art methods, *i.e.* **NeuTraj** and **Traj2SimVec**, suffer from at least a 40% drop on long trajectories. Similar results can also be observed on other distance metrics. Solving this problem is vital for DRL-based similarity computation.

Current DRL methods are incapable of modeling the long-term dependencies, which leads to the performance drop on long trajectory. On the one hand, according to the definitions of similarity measures, the similarity of two trajectories is usually dominated by

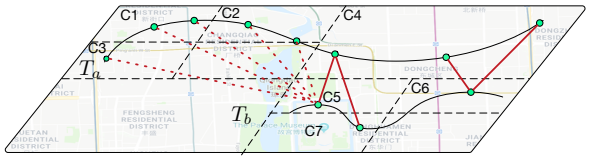
*Corresponding authors.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/3534678.3539358

DTW	Mix			Long		
Method	**HR@10**	**HR@50**	**R10@50**	**HR@10**	**HR@50**	**R10@50**
NeuTraj	0.4635	0.5519	0.8494	0.0585	0.1213	0.2188
Traj2SimVec	0.2628	0.3045	0.5702	0.1476	0.1577	0.2812
NT-No-SAM	0.4490	0.5272	0.8322	0.0578	0.1170	0.2157

(a). Performance comparison of DRL methods on Xian trajectory dataset



(b). illustration of long and short trajectories

Figure 1: The motivation of TrajGAT. The reasons of performance decline on long trajectories are illustrated in (b). T_a is the long trajectory and T_b is the short trajectory. The solid red lines indicate the alignments captured by RNNs.

some record alignments. The alignments between long and short trajectories span across different regions, as illustrated in Figure 1 (b). Current methods employ recurrent neural networks(RNNs) to encode trajectories into embeddings while maintaining the similarity relationship. These RNN models use backpropagation through time (BPTT) for optimization, which can only capture the short-term dependence of the latest observed records and is difficult to scale to long sequence[37]. As a result, existing models fail to capture the alignment information of long trajectories. On the other hand, DRL methods model the spatial information by learning the shared representations of equal-sized grids[16, 28], resulting in weak connections for records located far apart in sequence modeling. In addition, the records are unevenly distributed in spatial. Some grid cells lack sufficient data to train their representations, which further deteriorates the performance on long trajectory. Therefore, it is critical and essential to capture the long-term dependency in long trajectories for computing the similarity.

Nevertheless, modeling the long-term dependency for trajectory similarity computation is not trivial. Existing works for modeling long sequence can be categorized into three groups, *i.e.* RNN-based methods[4, 22], memory network-based methods[5, 11], and Transformer-based methods [7, 26, 39]. But none of them can be used in our task. For RNN-based methods, auxiliary losses are employed in optimization, which not only make the model hard to train but also lead to suboptimal performance on metric approximation. Memory network-based methods rely on heuristics designs of memory structure and are usually incapable of capturing the sequential relationship. By stacking self-attention operations, Transformer has demonstrated its superiority in capturing long-term dependence. However, with the increase of sequence length, the GPU memory demand of Transformer increases quadratically. Although several algorithms [32, 39] are proposed for improving the efficiency of self-attention, they cannot utilize the spatial information and thus cannot be used directly for trajectory similarity computation.

In this paper, we propose a novel method, namely TrajGAT, to capture the long-term dependency for trajectory similarity computation. TrajGAT integrates the hierarchical spatial structures into trajectory encoding, which not only models the cross-region relationships in long trajectories explicitly but also constrains the GPU memory demand of self-attention in Transformer. Specifically, TrajGAT first employs PR quadtree[21] to build the hierarchical structure. Location records in all quadtree leaf grids are balanced, which ensures the equivalent training of grids representations. Based on the quadtree, we construct graphs for all trajectories by involving extra edges between original records and their related grids. Then, a graph attention(GAT)-based Transformer is designed to encode trajectory graphs into embedding vectors. Instead of computing attention of all pair-wise records, GAT-based Transformer only aggregates information along the edges in trajectory graph to reduce GPU memory cost. Finally, the embedding vectors are fed to a metric learning framework to approximate the similarity metrics. The main contributions of this paper are summarized as follows:

- We propose a novel method to solve the performance decline problem on long trajectory similarity computation. To the best of our knowledge, TrajGAT is the first deep learning method for GPS trajectory modeling that captures long-term dependence.
- We design GAT-based Transformer which explicitly integrates the hierarchical spatial structure and converts trajectories into graphs for trajectory encoding. It can model the cross-region relationship while reducing the GPU memory usage.
- Extensive experiments on two public trajectory datasets show that TrajGAT can not only improve the similarity computation performance on long trajectories, but also outperform the state-of-the-art methods on mixture trajectories.

2 RELATED WORK

In this section, we provide an overview of existing studies related to TrajGAT from three perspectives: (1) trajectory similarity computation; (2) long sequence modeling; and (3) graph transformers.

Trajectory similarity computation. Existing trajectory similarity computation methods can be broadly divided into two categories, non-learning-based methods and learning-based methods. Most non-learning-based methods [1, 3, 8, 20] view each trajectory as a spatial curve and employ computational geometry to simplify calculation. These techniques mainly rely on hand-crafted heuristic rules, which may result in poor performance. Furthermore, they are designed for specific distances, and hence it is hard to adapt these techniques for other similarity metrics. Recently, with the development of AI[27], many learning-based methods [12, 16, 25, 30, 35, 36] are proposed to embed the spatio-temporal characteristics of trajectories into vectors and compute the similarity. However, these methods rely on RNN for sequence modeling and utilize BPTT to optimize the parameters. Although these methods perform well on short trajectory datasets, they suffer from performance erosion on long trajectory datasets.

Long sequence modeling. To model the long-term dependence, existing works can be categorized into three groups, *i.e.* RNN-based methods, memory network-based methods, and Transformer-based methods. By adding an unsupervised loss, Trinh *et al.* [22] improved the ability of RNNs to capture long-term dependencies. Francois *et al.* [4] proposed a principled estimation procedure of long-term dependencies and designed an EvolutiveRNNs for long sequence modeling. These RNN-based methods also involved external objectives for optimization which would make the model hard to

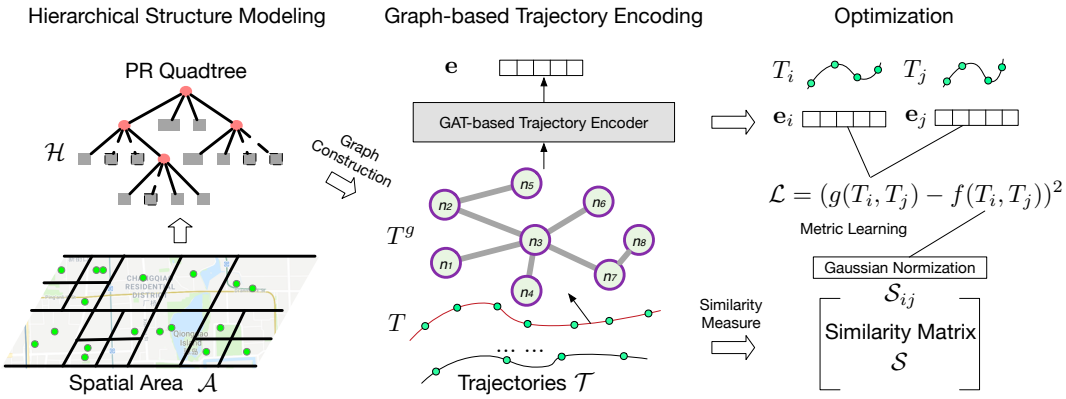


Figure 2: Architecture of TrajGAT.

train and lead to suboptimal performance. Memory network-based methods [5, 11] model the long-term dependencies by sharing the memory tensor. However, the structure of memory tensor is based on hand-crafted designs and cannot be extended. Recently, many Transformer-based approaches [26, 39] have been proposed for long sequence modeling. But the GPU memory cost of these models are quadratically expanded with the increase of sequence length, which limits their use. Moreover, all existing long-sequence modeling models are unable to capture spatial information, which is critical for trajectory similarity computation.

Graph transformers. Many studies [13, 34, 38] have explored combining graph neural network and Transformer to capture structure information of sequences. Nevertheless, all of these methods need the modeling of a graph, which is not easy to obtained in trajectory data. There are works that construct graphs for sequential data, *e.g.* sentences [32]. Unfortunately, these methods are unable to model trajectory due to the particularities of spatio-temporal characteristics.

3 PRELIMINARY

3.1 Problem Definition

We consider a trajectory database $\mathcal{T}$ containing N trajectories and a trajectory similarity measure $f(\cdot,\cdot)$. Each trajectory $T \in \mathcal{T}$ is a set of locations recording the trace of a moving object. Without losing generality, we consider two-dimensional trajectories. Each trajectory $T = [X_1, ..., X_t, ...]$ is a sequence of tuples where $X_t(lat_t, lon_t)$ is the $t-th$ location of the object. For any two trajectories $T_i, T_j \in \mathcal{T}$, $f(T_i, T_j)$ measures the similarity between T_i and T_j. Here, $f(\cdot,\cdot)$ could be the DTW, Hausdorff, Fréchet distances, or any other trajectory similarity measure.

Our task is to compute the similarity for an ad-hoc pair of trajectories from $\mathcal{T}$ under the similarity function $f(\cdot,\cdot)$. However, for most prevailing similarity measures, computing the similarity between a pair of trajectories incurs quadratic time complexity. Existing DRL-based methods [16, 30, 35] suffer from a serious performance drop on long trajectories. Hence, the research question is: how can we learn an approximate similarity function $g(\cdot,\cdot)$, such that computing $g(T_i, T_j)$ is efficient while the differences $|f(T_i, T_j) - g(T_i, T_j)|$ on both long and short trajectories are minimized.

3.2 Overview of TrajGAT

As shown in Figure 2, TrajGAT is a graph-based metric learning framework. It consists of three key modules: hierarchical structure modeling, graph-based trajectory encoding, and metric learning-based optimization. Assuming that trajectories in $\mathcal{T}$ are distributed in an area $\mathcal{A}$, TrajGAT computes trajectory similarities as follows:

- We first employ Point-Region(PR) quadtree to construct a hierarchical partition of $\mathcal{A}$, denoted as $\mathcal{H}$. The grid cells in $\mathcal{H}$ captures the hierarchical spatial information. To integrate such information in the trajectory encoder, we first conduct a pre-training on $\mathcal{H}$ to learn the embeddings of grid cells which preserve the hierarchical information.
- In graph-based trajectory encoding, TrajGAT utilizes the tree structure of $\mathcal{H}$ to convert each trajectory T into a graph T^g. The leaf nodes of T^g are the original records, while the non-leaf nodes are the relative grid cells that form a substructure of $\mathcal{H}$. To encode T^g and reduce the GPU memory usage, we proposed a graph-based Transformer layer that models the sequential relationship as well as the hierarchical structure to generate the trajectory embeddings.
- For optimization, we utilize the ranking loss to fit the similarities of trajectory pairs and propose a normalization approach to make the supervised information easier to learn.

4 METHODOLOGY

In this section, we will introduce the details of TrajGAT. It consists of three key modules, hierarchical structure modeling, graph-based trajectory encoding, and the metric learning-based optimization. And we will define the structure of each module respectively.

4.1 Hierarchical Structure Modeling

As shown in Figure 2, we first utilize the trajectories in $\mathcal{T}$ to build a quadtree $\mathcal{H}$ which represents the hierarchical spatial structure. Then, the embeddings of grid cells in $\mathcal{H}$ are pre-trained. The following is a description of the two procedures:

4.1.1 Hierarchical Structure Construction. TrajGAT employs PR quadtree[21] to model the hierarchical spatial information. As shown in Figure 3, it recursively decomposes the region into four equal quadrants, sub-quadrants, and so on. For $\mathcal{T}$ in an area $\mathcal{A}$, we build the hierarchical structure with PR quadtree as follows:

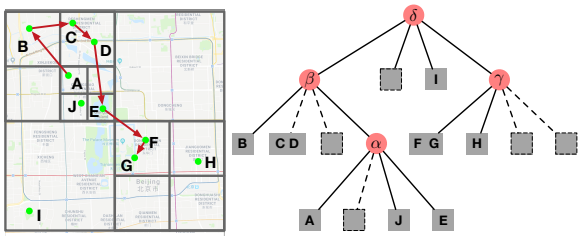


Figure 3: PR quadtree for hierarchical structure modeling.

1. We first extract the location records from all trajectories and construct a location set with equal geo-coordinates for duplicate locations.
2. With an empirical threshold δ, we recursively decompose the region and yield subregions as children nodes until no leaf node has more than δ locations.
3. Finally, we omit the location list of leaf nodes and only employ the hierarchical partitions of $\mathcal{A}$ as the hierarchical structure. We denote it as $\mathcal{H}$.

As illustrated, Figure 3 provides a toy hierarchical construction process with $\delta = 2$. The whole area is decomposed into 3 layers and 13 grids cell with no overlap.

Although $\mathcal{H}$ changes as $\mathcal{A}$ increases, we argue that the appropriate structure for similarity computation trends to be stable. It indicates that $\mathcal{H}$ can be constructed with partial data and is generalized for other new trajectories. We will prove this by an experiment in Appendix A.3. Thus, the following modules of TrajGAT are designed based on $\mathcal{H}$.

4.1.2 Learning Cell Embeddings of $\mathcal{H}$. To integrate the hierarchical information for trajectory encoding, we conduct pre-training on $\mathcal{H}$ to obtain the embeddings of grid cells $\mathbf{M}_{\mathcal{H}}$. Assuming there exist K grid cells in $\mathcal{H}$, we treat $\mathcal{H}$ as a graph and conduct Node2Vec on it. Specifically, we sample a set of paths from $\mathcal{H}$. The embedding vectors of grid cells are learned by maximizing the likelihood of preserving network neighborhoods while exploring diverse neighborhoods. With the sampled cross-layer paths, the learned embedding matrix $\mathbf{M}_{\mathcal{H}} \in d_h \times N_h$ captures the hierarchical information of $\mathcal{H}$, where d_h is the dimension of grid cell embedding. After that, we utilize $\mathbf{M}_{\mathcal{H}}$ to enhance the trajectory encoding.

4.2 Graph-based Trajectory Encoding

For trajectory encoding, we first construct a graph for each trajectory in $\mathcal{T}$, and then propose a GAT-based Transformer to generate the embedding of trajectory graph.

4.2.1 Trajectory Graph Construction. Given a trajectory T, we construct a graph $T^g = (\mathbf{N}, \mathbf{E})$ by considering the grid cells in $\mathcal{H}$, where $\mathbf{N}$ is the node set and $\mathbf{E}$ is the edge set. As shown in Figure 4, T^g not only contains the original records but also the hierarchical information of $\mathcal{H}$. We specify the construction of graph structure and node features respectively.

Graph Structure Construction. In this part, we sequentially introduce the construction process of nodes and edges. Nodes in T^g are extracted from the original records of T and its related grid cells in $\mathcal{H}$ from different layers, *i.e.* $\mathbf{N} = \mathbf{N}_r \cup \mathbf{N}_h$. Specifically, assuming that the length of T is L, we first build nodes for all records, *i.e.*, $\mathbf{N}_r = \{\mathbf{n}_1, ...\mathbf{n}_L, \}$. Each node $\mathbf{n}_i$ contains all the information of the

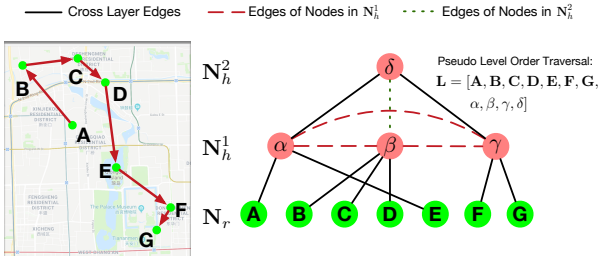


Figure 4: The structure construction of trajectory graph.

original records. Taking $\mathbf{N}_r$ as the layer-0 nodes, we recursively involve the hierarchical grid cells which related to the previous layer nodes if they are not be added. As shown in Figure 4, for constructing layer-1 nodes, TrajGAT groups the layer-0 nodes by its coordinates with respect to the leaf grid cells of $\mathcal{H}$. Similarly, for layer-2 nodes, TrajGAT groups the layer-1 nodes according to the first non-leaf nodes in $\mathcal{H}$. For all the nodes extracted from $\mathcal{H}$, we denote them as $\mathbf{N}_h = \{\mathbf{N}_h^1, ..., \mathbf{N}_h^\eta\}$, where η is the extracted layers and $\mathbf{N}_h^i$ represents the subset nodes of each layer. Note that η is a key hyperparameter, and we study the effects of it in Appendix A.3 due to the space limit.

After constructing the nodes in T^g, we build the edges $\mathbf{E}$. We design two kinds of edges in T^g. The first is cross-layer edges $\mathbf{E}_c$, which connects nodes from different layers. As the leaf grid cells of $\mathcal{H}$ is a partition of $\mathcal{A}$, we find the related cells of all records and add edges from $\mathbf{N}_r$ to $\mathbf{N}_h^1$. Accordingly, other cross layers edges are extracted from the tree structure of $\mathcal{H}$. The second type of edge is the inner-layer edge. To improve GPU memory efficiency and reduce the computation cost of TrajGAT, we only construct the inner-layer edges in $\mathbf{N}_h$. For each layer $\mathbf{N}_h^i$, we add fully-connected edges between the nodes if they are not connected in the lower layers. It's worth noting that all the edges in $\mathbf{E}$ are undirected, allowing the messages to bidirectionally pass through the nodes. The process of constructing a graph is illustrated in Figure 4.

Node Feature Construction. We consider the trajectory graph $T^g = (\mathbf{N}, \mathbf{E})$ as the homogeneous graph, and treat nodes in both $\mathbf{N}_r$ and $\mathbf{N}_h$ equally. The node features in $\mathbf{N}$ consist of three aspects: coordinate features $\mathbf{f}^l$, region features $\mathbf{f}^r$, and hierarchical structure features $\mathbf{f}^h$. Next, we will describe their construction procedure, respectively.

Each node in $\mathbf{N}$ represents a spatial element in area $\mathcal{A}$, which is either a location or a rectangle region. For nodes in $\mathbf{N}_r$, we directly utilize the location of their related records as the related location. For nodes in $\mathbf{N}_h$, we choose their center locations as the related locations. Note that all locations in the trajectory are GPS coordinates, we first normalize them with a min-max normalization function and employ the Multi-layer Perceptron(MLP) to conduct non-linear transformation. Assuming $X_i = (\text{lat}_i, \text{lon}_i)$ is the related location of node $\mathbf{n}_i \in \mathbf{N}$, the location feature is constructed as follows:

$$\text{x}_i, \text{y}_i = \text{Normalize}(\text{lat}_i, \text{lon}_i)$$

$$\mathbf{f}_i^l = \text{MLP}(\text{x}_i, \text{y}_i)$$

Moreover, we employ region features to model the region size. If $\mathbf{n}_i$ is the node in $\mathbf{N}_h$, we extract the width w_i and height h_i of its related grid cell, and conduct a non-linear transformation to obtain

the region feature. If $\mathbf{n}_i$ in $\mathbf{N}_r$, the default value $w_i = 0; h_i = 0$ are set as its region features. The operation is formulated as follows:

$$\mathbf{f}_i^r = \text{MLP}(w_i, h_i)$$

For the hierarchical features, we directly utilize the pre-trained embeddings matrix of $\mathcal{H}$. For nodes in $\mathbf{N}_r$, we add one special key in $\mathbf{M}_{\mathcal{H}}$ which represents the hierarchical feature of all original records. In this way, spatial-nearby records in different trajectories are connected by sharing the same hierarchical features.

$$\mathbf{f}_i^h = \text{Embedding}(\mathbf{M}_{\mathcal{H}}, \mathbf{n}_i)$$

In summary, for a specific node $\mathbf{n}_i \in \mathbf{N}$, we concatenate the three parts of features together and obtain the node features:

$$\mathbf{f}_i = \text{concat}(\mathbf{f}_i^{\text{l}}, \mathbf{f}_i^{\text{r}}, \mathbf{f}_i^{\text{h}}) \tag{1}$$

Without loss generality, we assume the dimensions of $\mathbf{f}_i^l, \mathbf{f}_i^r$, and $\mathbf{f}_i^h$ equal to d_f. The dimension of node feature $\mathbf{f}_i$ is $d = 3 \times d_f$.

4.2.2 GAT-based Trajectory Encoder. In this section, we introduce the GAT-based trajectory encoder which is the core part for encoding the trajectories. It follows the main idea of Transformer while taking a graph as the input and solving the high GPU memory cost problem. As illustrated in Figure 5, the key insight is that not every record needs to be attended to all other ones for similarity computation. By cooperating with the constructed trajectory graph, GAT-based Transformer enables the records attending to different grid cells spanning away from them. Compared with vanilla Transformer, our model has two novel designs:(1) Position Encoding for preserving both sequential and position information of trajectory, and (2) GAT-based Transformer which models long trajectories by utilizing the graph-attention operation. We specify the two designs below.

Position Encoding. Before feeding the trajectory graph into encoder, we first introduce the position encoding, which includes both sequential information of records and graph relationship of nodes.

To model the sequential information, we design a pseudo-level order traversal on T^g. We first utilize an empty list $\mathbf{L}$ to store the node sequence and add nodes of T^g from level-0 to level-η, respectively. For the nodes in level-0 ($\mathbf{N}_r$), we employ the original sequence of T and add the related nodes to $\mathbf{L}$ sequentially. For nodes in level-i where $\eta \geq i > 0$, we add them to $\mathbf{L}$ without repeat according to the appearing order of their connected nodes in $\mathbf{N}_{i-1}^h$. The construction of $\mathbf{L}$ is shown in Figure 4.

With the sequential nodes in $\mathbf{L}$, we employ the sinusoidal values proposed by Vaswani *et al.* [23] for encoding the position information. This approach is an absolute position encoding called sinusoidal position embeddings. Assume $\mathbf{L}$ contains M nodes, we construct the sequential position embeddings as follows:

$$\lambda_{ij} = \begin{cases} sin(10000^{\frac{j}{i}}), & \text{if i is even} \\ cos(10000^{\frac{j-1}{i}} i), & \text{if i is odd} \end{cases}$$

where $i = [1, ..., M]$ and $j = [1, ..., d/2]$. For the sequential position encoding of node $\mathbf{n}_i \in \mathbf{N}$, we denote it as $\lambda_i^s \in \mathbb{R}^{d/2}$.

To model the graph relationship in T^g, we employ Laplacian position encoding[9] to encode relative distance information, *i.e.*

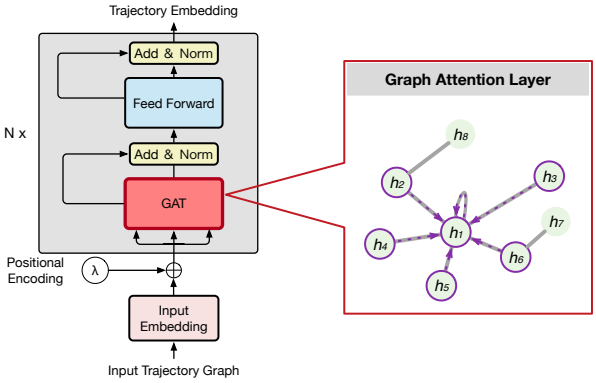


Figure 5: The architecture GAT-based Transformer

nearby nodes have similar positional features and farther nodes have dissimilar positional features.

$$\mathbf{\Delta} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} = \mathbf{U}^T\mathbf{\Lambda}\mathbf{U}$$
$$\lambda_i^g = \text{MLP}(\text{top}_{\text{p}}(\mathbf{U})))$$

where $\mathbf{A}$ is the adjacency matrix, $\mathbf{D}$ is the degree matrix, and $\mathbf{\Lambda}, \mathbf{U}$ correspond to the eigenvalues and eigenvectors, respectively. Besides, λ_i denotes the Laplacian positional encoding for node i, and top_{p} indicates the operation that slicing the p smallest non-trivial of $\mathbf{\Lambda}$, and it related eigenvectors at the position of node $\mathbf{n}_i$. By feeding the eigenvectors to a fully connected layer, we obtain the graph position encoding, denote as $\lambda_i^g \in \mathbb{R}^{d/2}$.

After the generation of λ_i^s and λ_i^g, we concatenate them to obtain the final position encoding vector λ_i and add it with the node features. The result is taken as the input of the next GAT-based Transformer Layer.

$$\begin{aligned} \lambda_i &= \text{concat}(\lambda_{\text{i}}^{\text{s}}, \lambda_{\text{i}}^{\text{g}},) \\ \mathbf{i}_i &= \lambda_i + \mathbf{f}_i \end{aligned} \tag{2}$$

where $\mathbf{i}_i \in \mathbb{R}^d$ is the input vector of node $\mathbf{n}_i$.

GAT-based Transformer Layer. Due to the self-attention operation, vanilla Transformer needs to compute all the pair-wise attention weights and suffers from the high GPU memory usage problem. For long trajectories, this problem could be even worse. We have conducted a toy experiment on trajectories whose max length is 1000. On Nvidia 3090 GPU with 24G memory, the max batch size of vanilla Transformer equals 1, which is extremely inefficient. In contrast, we observe most similarity measures are defined on the partial alignments of record pairs, such as Hausdorff, Fréchet, and w-DTW. Computing all pair-wise attention is neither efficient nor necessary in trajectory similarity computation. Thus, taking the $\mathbf{i}_i$ as the input $\mathbf{h}_i^0$, we replace the self-attention layer with the graph-attention layer to yield the trajectory embeddings. For one specific node $\mathbf{n}_i$, the operations in GAT-based Transformer layer are defined as follows:

$$\begin{aligned} \mathbf{h'}_i^{\ell+1} &= \mathbf{O}_h^\ell \, \text{concat}_{k=1}^{H}\left(\sum_{j\in\mathcal{N}_i} \mathbf{w}_{ij}^{k,\ell}\mathbf{V}^{k,\ell}\mathbf{h}_j^\ell\right), \\ \text{where, } \mathbf{w}_{i,j}^{k,\ell} &= \text{softmax}_j\left(\frac{\mathbf{Q}^{k,\ell}\mathbf{h}_i^\ell \cdot \mathbf{K}^{k,\ell}\mathbf{h}_j^\ell}{\sqrt{d_k}}\right), \end{aligned} \tag{3}$$

where $\mathcal{N}_i$ is the neighborhoods of node n_i, $\mathbf{Q}^{k,\ell}, \mathbf{K}^{k,\ell}, \mathbf{V}^{k,\ell} \in \mathbb{R}^{d_k \times d}, \mathbf{O}_h^{\ell} \in \mathbb{R}^{d \times d}, k = 1$ to H indicates the number of attention heads. For numerical stability, the outputs after taking exponents of the terms inside $softmax$ are limited to the range of $(-5, 5)$. Then the attention output $\mathbf{h'}_i^{\ell+1}$ are fed into a Feed Forward Network (FFN) which contains the residual connections and batch normalization modules. For concise, we omit the equations of identical operations with Transformer.

After the P layers of GAT-based Transformer, we obtain the hidden representations of all nodes *i.e.* $[\mathbf{h}_1^P, ..., \mathbf{h}_i^P, ..., \mathbf{h}_M^P]$. To generate the final embedding of trajectory graph T^g, we employ the mean readout function on node embedding. The embedding $\mathbf{e}$ of trajectory T is computed as $\mathbf{e} = \sum_{i=1}^{M} \mathbf{h}_i^P / M$. The outputs of Graph-based Trajectory Encoding are trajectory embeddings which can be used to measure the trajectory similarities.

4.3 Metric Learning for Optimization

Based on the embeddings of trajectory graphs, TrajGAT employs deep metric learning framework to optimize the model parameters. Given the distance matrix $\mathcal{D}$ containing the pair-wise distances of trajectory pairs in $\mathcal{T}$, we follow the method introduced in [28] and first transform $\mathcal{D}$ to a similarity matrix $\mathcal{S}$ and use the $\mathcal{S}$ as the supervised information, *i.e.*, $\mathcal{S}_{ij} = \frac{\exp(-\theta\ \mathcal{D}_{i,h})}{\max(\exp(-\theta\ \mathcal{D}))}$, where θ is a tunable parameter controlling the similarity value distribution. Then, trajectories in $\mathcal{T}$ are sequentially taken as the anchor trajectory for model optimization.

Previous methods [28, 35] directly employed the similarity values in $\mathcal{S}$ as the ground truth. However, most of the similarities of T_a trend to aggregate in a small region, which cannot provide clear supervised information. We conduct Gaussian normalization on the similarity list of T_a before computing the loss:

$$\mu_a = \frac{\sum_{i \in [1,...,N]/a} \mathcal{S}_{ai}}{N-1}; \sigma_a = \sqrt{\frac{\sum_{i \in [1,...,N]/a} (\mathcal{S}_{ai} - \mu_a)^2}{N-1}}$$
$$f(T_a, T_j) = (\mathcal{S}_{aj} - \mu_a)/\sigma_a$$

After the normalization, the similarity distribution of T_a conforms to a standard Gaussian distribution. TrajGAT can capture the supervised information more easily.

Assume that n trajectories are sampled for T_a to fit the similarities, we compute the loss of T_a as follows:

$$L_a = \sum_{i=1}^{n} r_i \ (g(T_a, T_i) - f(T_a, T_i))^2 \tag{4}$$

where $g(T_i, T_j) = exp(-\text{Euclidean}(\mathbf{e}_i, \mathbf{e}_j))$ computes the similarity between two trajectory embeddings; r is the sample weight calculated by the weighted rank loss [28]. Finally, the overall loss of $\mathcal{T}$ is the sum of all anchor trajectory losses, *i.e.*, $L_{\mathcal{T}} = \sum_{a \in \mathcal{T}} L_a$. All of the parameters in TrajGAT can be updated in an end-to-end way. We update the parameters with back-propagation algorithm and employ Adam optimizer for optimization.

4.4 Complexity Analysis of TrajGAT.

The computation of TrajGAT includes two parts, *i.e.*, the hierarchical structure construction and the trajectory encoding. The first part is the preprocessing procedure and does not influence the efficiency of similarity computation. The complexity of the second part is $O(\eta \cdot L + L \cdot log(L))$, which is higher than that of RNN-based models. However, the encoding time of TrajGAT is comparable with these methods in practice, because TrajGAT is more efficient on GPU. All of the operations in TrajGAT can be computed in parallel while the RNN-based models cannot. More details of complexity analysis can be found in the Appendix A.1.

5 EXPERIMENTS

In this section, we evaluate the performance of TrajGAT and answer the following questions:

- **Q1:** What is the performance of TrajGAT comparing with existing trajectory similarity computation methods?
- **Q2:** How efficient does TrajGAT perform on GPU memory usage to generate the trajectory embeddings?
- **Q3:** What are the capabilities of the proposed hierarchical structure modeling and graph-based trajectory encoding?

5.1 Experimental Settings

We briefly introduce the experimental settings below. The detailed experimental settings can be found in the Appendix A.2.

5.1.1 Data Descriptions. We employ two public trajectory datasets to evaluate the performance of TrajGAT. The first dataset is taxi trajectories in Xian provided by DiDi Inc. After the data prepossessing method proposed in [28], we obtain 5.25 million trajectories for the Xian dataset and over 0.6 million trajectories for the Porto dataset. For each dataset, we construct two sub-datasets, denoted as **Mix** and **Long**, to verify the performance of TrajGAT on different trajectory lengths. In our experiments, both **Mix** and **Long** of the two datasets contain 10, 000 trajectories.

5.1.2 Experimental Protocol. To evaluate the performance of TrajGAT, we conduct experiments on two tasks, *i.e.*, Top-K trajectory similarity search (Section 5.2) and trajectory clustering (Section 5.3), and compare the performance of TrajGAT with many learning-based similarity computation methods. To measure the efficiency of TrajGAT, we compare the GPU memory cost and report the results in Section 5.4. The results of ablation studies are detailed in Section 5.5. Moreover, we also test the sensitivity of key parameters in the Appendix A.3.

5.1.3 Compared Baselines. We compare TrajGAT with six representative works, including **NT-No-SAM** [28], **traj2vec** [30], **t2vec** [16], **NeuTraj** [28], **Traj2SimVec** [35] and **Transformer** [23]. The details of these methods are specified in the Appendix A.2.3.

5.1.4 Parameter Settings. TrajGAT employs 3-layers stack of GAT-based Transformer. The graph attention operation in each layer has 8 attention heads. We set the embedding dimension as 32 (i.e., $d_{model} = 32$). In addition, we set the sampling size n as 20.

5.2 Performance of Top-K Trajectory Similarity Search

To answer question **Q1**, we compare the performance of TrajGAT with the baselines on both Xian and Porto datasets, and show the experimental results in Table 1. From the results on **Mix** datasets, we observe: (1) TrajGAT significantly outperforms the baselines in

Table 1: Performance comparison on mixture trajectory dataset.

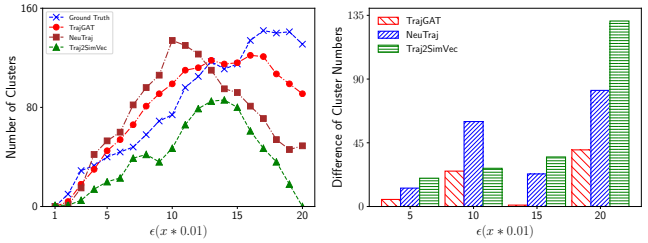
Dataset	Method	**DTW** on **Mix**			**Hausdorff** on **Mix**			**DTW** on **Long**			**Hausdorff** on **Long**		
		HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50
Xian	**NT-No-SAM**	0.4490	0.5272	0.8322	0.4476	0.6179	0.8695	0.0578	0.1170	0.2157	0.1082	0.1661	0.2604
	traj2vec	0.1845	0.2197	0.3939	0.1416	0.1841	0.2691	0.0425	0.0822	0.1903	0.0415	0.0813	0.2080
	t2vec	0.2511	0.3059	0.4909	0.2497	0.2910	0.4620	0.0479	0.0714	0.1870	0.0699	0.1011	0.1992
	NeuTraj	0.4635	**0.5519**	0.8494	0.4537	0.6409	0.8823	0.0585	0.1213	0.2188	0.1096	0.1723	0.2642
	Traj2SimVec	0.2628	0.3045	0.5702	0.3448	0.3414	0.5593	0.1476	0.1577	0.2812	0.1490	0.1990	0.2445
	Transformer	0.3259	0.4556	0.7725	0.7096	0.7993	0.9791	0.3044	0.4319	0.7439	0.4696	0.5602	0.8205
	TrajGAT	**0.4659**	0.5442	**0.9159**	**0.7476**	**0.8290**	**0.9849**	**0.3896**	**0.4702**	**0.8405**	**0.5405**	**0.6702**	**0.7805**
Porto	**NT-No-SAM**	0.1476	0.1601	0.3190	0.0893	0.1032	0.1768	0.0193	0.0260	0.1321	0.0223	0.0621	0.1618
	traj2vec	0.1456	0.1799	0.3519	0.1031	0.1245	0.2369	0.0251	0.0602	0.1500	0.0291	0.0715	0.1706
	t2vec	0.2451	0.2795	0.4590	0.2199	0.2479	0.4229	0.0337	0.0551	0.1037	0.0429	0.0775	0.1361
	NeuTraj	0.3281	0.4614	0.7547	0.3499	0.4458	0.7388	0.0994	0.1652	0.2529	0.1020	0.1774	0.2700
	Traj2SimVec	0.1768	0.1623	0.3461	0.1664	0.1332	0.3047	0.1976	0.2059	0.3737	0.1803	0.1759	0.3219
	Transformer	0.1061	0.1477	0.2548	0.5606	0.6442	0.9183	0.0332	0.0668	0.0867	0.1265	0.2057	0.3547
	TrajGAT	**0.4946**	**0.5344**	**0.8769**	**0.6569**	**0.7181**	**0.9589**	**0.3667**	**0.4205**	**0.7146**	**0.5344**	**0.6350**	**0.9037**

almost all metrics. Taking the Hausdorff distance on Porto dataset as an example, TrajGAT gains about two times performance improvements (from 34.99% to 65.69%) on HR@10 comparing the strongest baseline **NeuTraj**. Under the fact that TrajGAT utilizes the same supervised information, such improvements are impressive. (2) The advantages of TrajGAT are also obvious in both similarity measures. It achieves the best performance on both DTW and Hausdorff, which indicates TrajGAT is general for different trajectory similarity metrics. (3) Although all the compared methods employ deep neural networks to approximate the similarity function, TrajGAT has two advantages to obtain the best performance. First, the hierarchical structure is explicitly modeled, which enables TrajGAT capturing the location density distribution of the whole spatial area. Second, instead of using RNN for trajectory encoding, TrajGAT employs the GAT-based Transformer to model the sequential information, which can not only model long-term dependencies but also aggregate the spatial information from different layers.

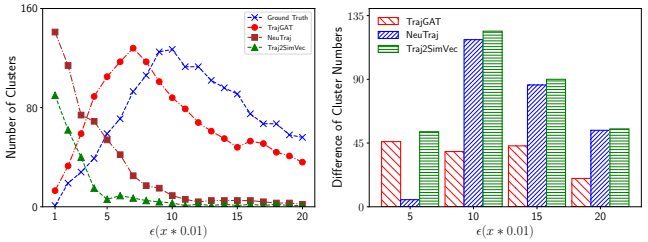
According to results on **Long** datasets, we note that: (1) TrajGAT achieves total supremacy compared with all baselines. On both Xian and Porto datasets, the performance of TrajGAT is at least two times higher than other methods. For example, TrajGAT improves HR@10 of Hausdorff on Porto dataset from 18.03% to 53.44%. This result indicates TrajGAT can capture the long-term dependency of trajectories which is useful for similarity computation. (2) Among all the compared methods, the performance of **Traj2SimVec** is better than others. It is because **Traj2SimVec** takes sub-trajectory similarities as the supervised information for model training, which implicitly captures the hierarchical relationship of long trajectories spanning across different regions. (3) Comparing the results of TrajGAT on **Mix** and **Long**, we find the performance on **Long** dataset is inferior to its on **Mix** dataset. This phenomenon indicates approximating the similarities on long trajectories is more challenging than it on mixture trajectories.

5.3 Performance of Trajectory Clustering

To verify the effectiveness of TrajGAT in computing pair-wise similarities and answer **Q1** better, we conduct the trajectory clustering experiment on the Porto dataset. The clustering algorithm



(a) **Mix** Dataset



(b) **Long** Dataset

Figure 6: The number of clusters changing with the increase of ϵ on Porto.

is DBSCAN which has two key parameters, *i.e.* the minimum samples m and the maximum distance ϵ within sample pairs of one cluster. As shown in Figure 6, we compare the number of clusters between accurate similarities (Ground Truth) and embedding-based similarities under various of ϵ. Compared with the result of other learning-based similarity computation methods, the cluster numbers differences of TrajGAT are relatively small (illustrated in the right part of Figure 6 (a)) indicating that the metric space of embeddings generated by TrajGAT is more approximate to the accurate metrics than other baselines. From Figure 6 (b), we observe the performances of **NeuTraj** and **Traj2SimVec** are poor on almost all cluster results. It proves existing learning-based methods can hardly approximate the metric space on long trajectories. Overall, TrajGAT achieves better performance on **Long** dataset which proves the long-term dependency captured by TrajGAT is vital for computing the similarities of long trajectories.

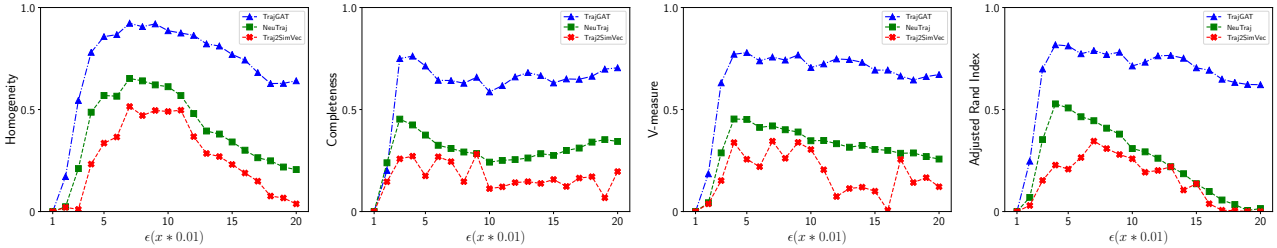


Figure 7: The comparison of cluster metrics changing with the increase of ϵ on Mix dataset of Porto.

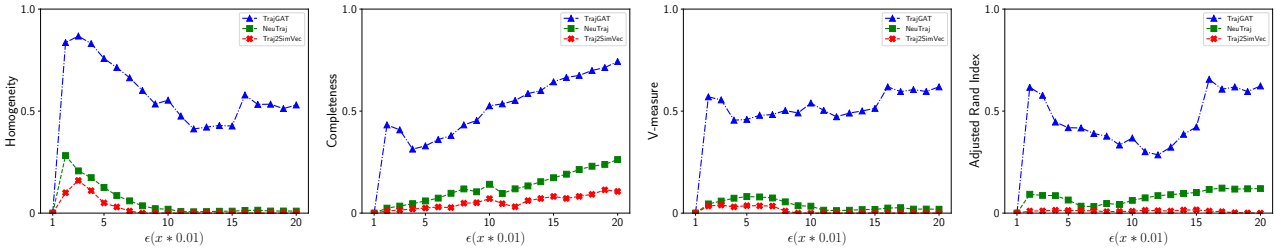


Figure 8: The comparison of cluster metrics changing with the increase of ϵ on Long dataset of Porto.

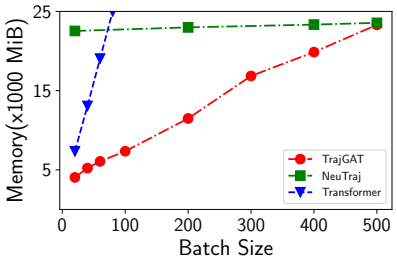


Figure 9: The results of GPU memory efficiency.

Moreover, we compute four clustering metrics for the embedding-based methods to measure the consistency of trajectories in different cluster results. As illustrated in Figure 7 and 8, TrajGAT significantly outperforms both **NeuTraj** and **Traj2SimVec** under all clusters. Taking the *Homogeneity* on **Mix** dataset as an example, TrajGAT achieves over 0.8 in most cases while the performances of compared methods are around 0.5. A similar phenomenon can also be observed on the other three clustering metrics. This result further shows the superiority of TrajGAT in approximating the metric space. Comparing the results between Figure 7 and 8, we observe the performance gaps between TrajGAT and other baselines on **Long** are higher than those on **Mix**, which is consistent with the number of clusters. One thing to note is that the clustering metrics of baselines are low on **Long** dataset indicating the compared methods can hardly learn meaningful embeddings for long trajectories. TrajGAT solves this problem by modeling the long-term dependency and obtains substantial performance improvements.

5.4 Efficiency Experiments

To answer **Q2**, we test the efficiency of TrajGAT on GPU memory usage. We evaluate the GPU memory usage of TrajGAT with the change of batch size. We compare TrajGAT with **Transformer** and **NeuTraj** by denoting the allocated GPU memory in model inference. The results are shown in Figure 9. As illustrated, the GPU memory usage of TrajGAT is much less than the two baselines. Comparing TrajGAT with **Transformer**, we observe that **Transformer** encounters the out-of-memory problem at batch size > 100 while TrajGAT takes only about 20% GPU memory(500 MiB). It proves that TrajGAT significantly decreases the GPU memory usage by transforming original trajectories to trajectory graphs and modeling them with graph attention. Comparing TrajGAT with **NeuTraj**, we note the memory usage of **NeuTraj** is always high, *i.e.*, over 90%.

The reason is **NeuTraj** employ a spatial memory tensor to model the spatial correlation, which requires a large amount of memory to store.

Table 2: Ablation Results

Dataset	Method	DTW		
		HR@10	HR@50	R10@50
Mix	Transformer	0.1017	0.1425	0.2621
	TrajGAT-graph	0.4091	0.4599	0.7867
	TrajGAT-tree	0.4774	0.5207	0.8603
	TrajGAT-trans	0.4447	0.4907	0.8347
	TrajGAT	0.4946	0.5344	0.8769
Long	Transformer	0.0346	0.0616	0.0795
	TrajGAT-graph	0.2559	0.3151	0.5347
	TrajGAT-tree	0.2752	0.3366	0.5811
	TrajGAT-trans	0.3252	0.3964	0.6608
	TrajGAT	0.3667	0.4205	0.7146

5.5 Ablation Results

To answer **Q3**, we compare TrajGAT with its three ablations, *i.e.* TrajGAT-tree, TrajGAT-graph and TrajGAT-trans. The architectures of these ablations are specified as follows:

- TrajGAT-graph employs the leaf nodes of quadtree as the spatial partition and utilizes the Transformer to model the grids.
- TrajGAT-tree divides the spatial area with equal-sized grids and employs GAT-based Transformer to model the 2-layers trajectory graph. We employ $50m \times 50m$ as the grid size.
- TrajGAT-trans constructs trajectory graphs and employs graph attention networks[24] to generate the trajectory embeddings.

The experiment is conducted on the Porto dataset and the results are shown in Table 3. We observe: (1) By integrating the hierarchical spatial structure explicitly, the performance increase of top-k

similarity search is remarkable (*i.e.*, from **Transformer** 10.17% to TrajGAT-graph 40.91% on **Long** under DTW), which proves the effectiveness of the long-term dependency modeling. (2) Comparing the results of TrajGAT-tree with **Transformer**, we observe the graph construction procedure can extract useful information and model the long term dependencies in trajectory, which are useful for the similarity computation on both **Mix** and **Long**. For example, the HR@10 of Hausdorff on **Mix** improves from 25.28% to 60.95%. (3) From the results of TrajGAT-trans and TrajGAT, we can conclude that the GAT-based Transformer is more effective than graph attention networks for trajectory similarity computation. (4) TrajGAT achieves the best performance compared to all ablations, which proves the effectiveness of the proposed techniques.

6 CONCLUSION

In this paper, we are the first to observe the performance gap between long and short trajectories for current DRL-based methods. We attribute this problem to the lack of modeling long-term dependency. To solve the problem, a graph-based method, TrajGAT, is proposed. It explicitly models the hierarchical structure of spatial area built by PR-quadtree and constructs the trajectory graph based on the grid cells of quadtree. Besides, a GAT-based Transformer is designed to capture both spatial and sequential information of trajectories while reducing the GPU memory demand. Extensive experiments on two public datasets show that TrajGAT achieves significant performance improvements on long trajectories and outperforms all comparing baselines.

ACKNOWLEDGMENTS

This work has been supported by the National Natural Science Foundation of China under Grant No.: 62002343, 62077044, 61702470. This work is also supported in part by a grant awarded by AI.SG with award No. AISG2-TC-2021-001.

REFERENCES

[1] Pankaj K Agarwal, Kyle Fox, Jiangwei Pan, and Rex Ying. 2015. Approximating dynamic time warping and edit distance for a pair of point sequences. *arXiv preprint arXiv:1512.01876* (2015).
[2] Stefan Atev, Grant Miller, and Nikolaos P. Papanikolopoulos. 2010. Clustering of Vehicle Trajectories. *TITS* 11, 3 (2010), 647–657.
[3] Arturs Backurs and Anastasios Sidiropoulos. 2016. Constant-distortion embeddings of hausdorff metrics into constant-dimensional l_p spaces. In *APPROX-/RANDOM 2016*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[4] Francois Belletti, Minmin Chen, and Ed H Chi. 2019. Quantifying long range dependence in language and user behavior to improve rnns. In *KDD*. 1317–1327.
[5] Yen-Yu Chang, Fan-Yun Sun, Yueh-Hua Wu, and Shou-De Lin. 2018. A memory-network based solution for multivariate time-series forecasting. *arXiv:1809.02105* (2018).
[6] Lei Chen and Raymond T. Ng. 2004. On The Marriage of Lp-norms and Edit Distance. In *VLDB*. 792–803.
[7] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv:1901.02860* (2019).
[8] Anne Driemel and Francesco Silvestri. 2017. Locality-sensitive hashing of curves. *arXiv preprint arXiv:1703.04040* (2017).
[9] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2020. Benchmarking graph neural networks. *arXiv:2003.00982* (2020).
[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. 226–231.
[11] Tharindu Fernando, Simon Denman, Aaron McFadyen, Sridha Sridharan, and Clinton Fookes. 2018. Tree memory networks for modelling long-term temporal dependencies. *Neurocomputing* 304 (2018), 64–81.
[12] Peng Han, Jin Wang, Di Yao, Shuo Shang, and Xiangliang Zhang. 2021. A Graph-based Approach for Trajectory Similarity Computation in Spatial Networks. In *KDD*. 556–564.
[13] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *The Web Conference*. 2704–2710.
[14] Quanliang Jing, Di Yao, Chang Gong, Xinxin Fan, Baoli Wang, Haining Tan, and Jingping Bi. 2021. TrajCross: Trajecotry Cross-Modal Retrieval with Contrastive Learning. In *IEEEBigData*. IEEE, 344–349. https://doi.org/10.1109/BigData52589.2021.9671305
[15] Rikard Laxhammar and Göran Falkman. 2013. Online learning and sequential anomaly detection in trajectories. *TPAMI* 36, 6 (2013), 1158–1173.
[16] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S Jensen, and Wei Wei. 2018. Deep representation learning for trajectory similarity computation. In *ICDE*. IEEE, 617–628.
[17] Yang Liu, Xiang Ao, Linfeng Dong, Chao Zhang, Jin Wang, and Qing He. 2022. Spatiotemporal Activity Modeling via Hierarchical Cross-Modal Embedding. *IEEE Trans. Knowl. Data Eng.* 34, 1 (2022), 462–474.
[18] Yiding Liu, Kaiqi Zhao, Gao Cong, and Zhifeng Bao. 2020. Online anomalous trajectory detection with deep generative sequence modeling. In *ICDE*. IEEE, 949–960.
[19] Luís Moreira-Matias, João Gama, Michel Ferreira, João Mendes-Moreira, and Luis Damas. 2016. Time-evolving OD matrix estimation using high-speed GPS data streams. *Expert systems with Applications* 44 (2016), 275–288.
[20] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*. 262–270.
[21] Hanan Samet. 1988. An overview of quadtrees, octrees, and related hierarchical data structures. *Theoretical Foundations of Computer Graphics and CAD* (1988), 51–68.
[22] Trieu Trinh, Andrew Dai, Thang Luong, and Quoc Le. 2018. Learning longer-term dependencies in rnns with auxiliary losses. In *ICML*. PMLR, 4965–4974.
[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 5998–6008.
[24] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv:1710.10903* (2017).
[25] Zheng Wang, Cheng Long, Gao Cong, and Ce Ju. 2019. Effective and efficient sports play retrieval with deep representation learning. In *KDD*. 499–509.
[26] Jiehui Xu, Jianmin Wang, Mingsheng Long, et al. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *NeurIPS* 34 (2021).
[27] Yongjun Xu, Xin Liu, Xin Cao, and etc. 2021. Artificial intelligence: A powerful paradigm for scientific research. *The Innovation* 2, 4 (2021), 100179.
[28] Di Yao, Gao Cong, Chao Zhang, and Jingping Bi. 2019. Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach. In *ICDE*. IEEE, 1358–1369.
[29] Di Yao, Chao Zhang, Jian-Hui Huang, and Jingping Bi. 2017. SERM: A Recurrent Model for Next Location Prediction in Semantic Trajectories. In *CIKM*. 2411–2414.
[30] Di Yao, Chao Zhang, Zhihua Zhu, Qin Hu, Zheng Wang, Jianhui Huang, and Jingping Bi. 2018. Learning deep representation for trajectory clustering. *Expert Systems* 35, 2 (2018), e12252.
[31] Di Yao, Chao Zhang, Zhihua Zhu, Jian-Hui Huang, and Jingping Bi. 2017. Trajectory clustering via deep representation learning. In *IJCNN*. IEEE, 3880–3887.
[32] Zihao Ye, Qipeng Guo, Quan Gan, Xipeng Qiu, and Zheng Zhang. 2019. Bp-transformer: Modelling long-range context via binary partitioning. *arXiv:1911.04070* (2019).
[33] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. 1998. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *ICDE*. 201–208.
[34] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *NeurIPS* 32 (2019), 11983–11993.
[35] Hanyuan Zhang, Xingyu Zhang, Qize Jiang, Baihua Zheng, Zhenbang Sun, Weiwei Sun, and Changhu Wang. 2020. Trajectory similarity learning with auxiliary supervision and optimal matching. (2020).
[36] Yifan Zhang, An Liu, Guanfeng Liu, Zhixu Li, and Qing Li. 2019. Deep representation learning of activity trajectory similarity computation. In *ICWS*. IEEE, 312–319.
[37] Jingyu Zhao, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin, Guodong Li, and Guangjian Tian. 2020. Do rnn and lstm have long memory?. In *ICML*. PMLR, 11365–11375.
[38] Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. 2020. A data-driven graph generative model for temporal interaction networks. In *KDD*. 401–411.
[39] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI*.
[40] Zhihua Zhu, Di Yao, Jianhui Huang, Hanqiang Li, and Jingping Bi. 2018. Sub-trajectory- and Trajectory-Neighbor-based Outlier Detection over Trajectory Streams. In *PAKDD*, Vol. 10937. Springer, 551–563.

A APPENDIX

A.1 Complexity Analysis of TrajGAT

The computation of TrajGAT includes two parts, *i.e.*, the hierarchical structure construction and the trajectory encoding. We analyze the complexity of them respectively.

TrajGAT employs PR quadtree to recursively construct the hierarchical structure of the $\mathcal{A}$. Thus, the complexity of constructing $\mathcal{H}$ is proportional to the depth of quadtree which is usually $log(N_{all})$, where N_{all} is the number of location records in $\mathcal{T}$. The time complexity of hierarchical construction is $O(N_{all} \cdot log(N_{all}))$. The construction can be treated as the preparing procedure. When preparing the new trajectories, the hierarchical structure is shared. Therefore, the efficiency of similarity computation is not affected by this additional step.

For the trajectory encoding, we first convert each trajectory into a trajectory graph T^g based on the preconstructed $\mathcal{H}$. As introduced in Section 4.2.1, η layers hierarchical information are involved to construct the trajectory graph. The time complexity of this procedure is $O(\eta \cdot L)$, where L is the length of trajectory. Taking T^g as the input, TrajGAT employs GAT-based Transformer to yield the trajectory representations. For each GAT-based Transformer layer, the computation cost is $L \cdot N_{\mathrm{E}}$ where N_{E} is the number of edges in T^g and usually proportional to $log(L)$. Hence, the overall complexity of trajectory encoding is $O(\eta \cdot L + L \cdot log(L))$.

A.2 Experimental Settings

A.2.1 Data Descriptions. We employ two public trajectory datasets to evaluate the performance of TrajGAT. The first dataset is taxi trajectories in Xian [**?**], which contains 17,621 trajectories of human mobility from 2007 to 2010. The second dataset Porto[19] contains over 1.7 millions of taxi trajectories from 2013 to 2014. We follow the data prepossessing procedure of [28] which selects trajectories in the center area of the city and removes trajectories which are less than 10 records. After the processing, we obtain 7641 trajectories for the Xian dataset and over 600, 000 trajectories for the Porto dataset.

To verify the performance of TrajGAT on different trajectory lengths, we construct two sub-datasets for each dataset, denoted as **Mix** and **Long**. Trajectories having more than 200 records are referred to as long trajectories and assigned to **Long**. Under this setting, we obtain 30, 031 long trajectories for the Porto dataset and 85 long trajectories for the Xian dataset. Due to the high computation cost of computing pair-wise distances, it is intractable to compute all pair-wise distances for the two dataset. We randomly sample a subset of trajectories from the original dataset for performance evaluation. In our experiments, both **Mix** and **Long** of the two datasets contain 10, 000 trajectories. The pair-wise distance matrixes are precomputed to supervise the model training.

A.2.2 Experimental Protocol. To evaluate the performance of TrajGAT, we employ two tasks, Top-K trajectory similarity search and trajectory clustering, and compare the performance of TrajGAT with many learning-based similarity computation methods. Besides, we also compare the GPU memory cost to verify the efficiency of TrajGAT. The the experimental protocol are described in detail below.

Top-K Trajectory Similarity Search. We study the Top-k similarity search problem on both Xian and Porto datasets and evaluate TrajGAT under two distance measures, *i.e.*, Dynamic Time Warping (DTW) and Hausdorff distance. The Hausdorff distance is a metric, which means it is symmetric and meets the triangle inequality. Therefore, we learn the models to approximate the metrics directly. DTW is not a metric. We add the distance matrix with its transformation matrix to make it symmetric, and explore the performance of TrajGAT on non-metric similarity measure.

The ground truth of this problem is the exact top-k results based on precomputed accurate similarity, i.e. the Hausdorff distance and the DTW distance. To verify the ability of models on trajectories of different length ranges, we evaluate the performances of TrajGAT and all compared methods on both **Mix** and **Long** datasets. Furthermore, we randomly choose 20% trajectories as the training set, 10% trajectories as the validation set, and 70% trajectories as the test set.

For performance evaluation in the top-k similarity search experiment, we employ three different metrics: HR@10, HR@50, and R10@50. The HR@k represents the top-k hitting ratio, refers to the overlap percentage between produced top-k results and the ground truth. The R10@50 indicates the top-50 recall for the top-10 ground truth, which evaluates how many of the top-10 ground-truth trajectories are recovered by top 50 lists produced using different methods. The performance comparison of top-k trajectory similarity search is shown in Section 5.2.

Trajectory Clustering. To evaluate the performance of TrajGAT on computing pair-wise similarities, we conduct trajectory clustering experiments on Porto datasets and compare the difference of clusters between accurate similarities and embedding-based similarities. As for the clustering algorithm, we utilize DBSCAN[10], which is a widely used density-based clustering algorithm on spatial data, to obtain clustering results in various parameter settings. Specifically, we fix the minimum number of trajectories for a cluster and increase the ϵ to generate different clusters on both accurate similarities and learned embeddings. The clustering results are evaluated under five clustering metrics: the number of clusters, Homogeneity, Completeness, V-measure, and Adjusted Random Index. The result of trajectory clustering is shown in Section 5.3.

Efficiency. As shown in Section 4.4, the computation of trajectory similarities is constant after generating the trajectory embeddings. Thus, the inference cost for trajectory embeddings can be used to describe the efficiency of TrajGAT. In our experiments, we compare the GPU memory usage of TrajGAT to that of existing learning-based similarity computing methods to verify the efficiency. The experimental results are analyzed in Section 5.4

A.2.3 Compared baselines. We compare TrajGAT with six representative works, including **NT-No-SAM**[28], **traj2vec** [30], **t2vec** [16], **NeuTraj** [28], **Traj2SimVec** [35], and **Transformer** [23]. The main ideas of these methods are listed as follows:

- **NT-No-SAM** [28] uses a recurrent neural network to capture the sequential relationship, and regards the output hidden state as the embedding of trajectories.
- **traj2vec**[30] represents the trajectory features via the moving behavior of trajectories and reconstruction loss. This method

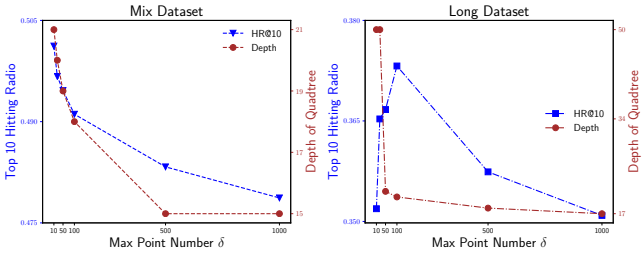


Figure 10: The performance changes with the maximum points for constructing PR-quadtree δ.

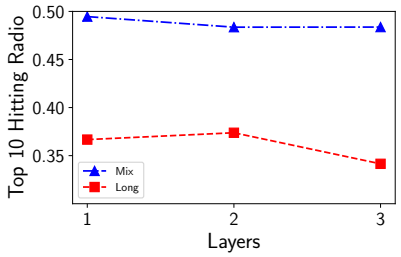


Figure 11: The performance changes with the layers η.

is designed for trajectory clustering. And it can also adapt to similarity computation without huge modifications.

- **t2vec**[16] is a seq2seq-based model, and adopts novel embedding techniques to deal with sampling rate variations and noisy sample points.
- **NeuTraj**[28] is a metric learning approach, which combines a spatial attention memory module and a distance-weighted ranking loss, and employs seed-guided neural network to compute trajectory similarity.
- **Traj2SimVec**[35] designs a strategy to extract distance information from sub-trajectories and limits the matching relationship between trajectory points based on different distance metrics.
- **Transformer**[23] is proposed for neural language processing, but it can also be adapted to model sequential data. Inspired by this work, we design our trajectory encoding method.

For methods having public code[16, 28, 30], we directly use their implementations. For other methods, we follow the settings of the relevant publications and implement them by ourselves.

A.2.4 Parameter Settings. For all datasets, we apply min-max normalization before feeding the input features. For the architecture of TrajGAT, we employ 3-layer stack of GAT-based Transformer in trajectory encoder. The graph attention operation in each layer has 8 attention heads. We set the embedding dimension as 32 (i.e., $d_{model} = 32$). In addition, we set the sampling size n as 10. The whole model is optimized by Adam optimizer with learning rate $\alpha = 1 \times 10^{-3}$. There also exists some hyperparameters in TrajGAT, *i.e.* δ and η. We study the influence of them in Appendix A.3.

Table 3: Ablation Results

Dataset	Method	Hausdorff		
		HR@10	HR@50	R10@50
Mix	Transformer	0.2528	0.3295	0.4981
	TrajGAT-graph	0.3594	0.4297	0.7125
	TrajGAT-tree	0.6095	0.6901	0.9431
	TrajGAT-trans	0.6268	0.7002	0.9486
	TrajGAT	0.6569	0.7181	0.9589
Long	Transformer	0.1301	0.2194	0.3600
	TrajGAT-graph	0.1788	0.2912	0.4309
	TrajGAT-tree	0.5471	0.6539	0.9122
	TrajGAT-trans	0.4620	0.5744	0.8328
	TrajGAT	0.5344	0.6350	0.9037

A.3 Parameter Sensitivity Analysis

In this experiment, we study the sensitivity of hyperparameters and answer **Q4**. Two key model-specified parameters *i.e.*, δ and η, are analyzed. We compared the top-k trajectory similarity search performance on different parameter settings and try to give the strategy to select them. All the experiments in this section are conducted on a Nvidia 3090 RTX.

Maximum points for constructing PR-quadtree δ. With the same area $\mathcal{A}$, we build the PR-quadtree with different trajectories and evaluate the influence of δ on **Long** and **Mix** trajectories of Porto. The **Long** has 2, 100, 219 records and the **Mix** has 502, 556 points. The performances of different $\mathcal{H}$ are compared with the increase of δ. As illustrated in Figure 10, the performance of TrajGAT in **Long** first increase and then decrease and the depth of the conducted quadtree decreases with the increase of δ. TrajGAT achieves the best performance on $\delta = 50$ and the relative depth of quadtree is about 21. For **Mix**, TrajGAT performs best on $\delta = 10$ and depth is also 21. We do not evaluate it on more small δ cause the dataset contains duplicated locations whose max number is 9. Thus, both **Long** and **Mix** achieve the best performance on similar hierarchical structures, suggesting that there exist a proper $\mathcal{H}$ for $\mathcal{A}$. We calculate the minimum grid cells of leaf nodes when the depth is 21. The scale of leaf grid cells is about $50m \times 50m$ indicating this scale is appropriate to capture hierarchical spatial information of Porto. For new trajectory datasets, we can first determine the minimum grid cells in quadtree and tune the δ to satisfy the scale.

Layers to generate the trajectory graph η. As illustrated in 11, the performance of TrajGAT increases with η and trends to be stable. On **Mix** dataset, TrajGAT achieves the best performance at $\eta = 1$. This result is conformed with our observation that the long-range connections are not significant in **Mix** dataset. On the contrary, involving more layers of hierarchical structure benefits the performance on **Long** dataset, which indicates that hierarchical spatial information is useful for improving long trajectory similarity computation. With the increase of η, the nodes of trajectory graph are expanded multiply, which would involves more computation to generate the trajectory embedding. Thus, we prefer to choose minimum η to achieve stable performance on new trajectory datasets.