TrajGAT: 基于图建模长距离依赖的轨迹相似度计算方法

孙政泽

摘要

轨迹相似度计算是各种时空数据挖掘应用的基础，如聚类、预测和异常检测等。传统的相似度计算方法，如 DTW、Hausdorff 等，其算法复杂度通常是平方级的，因此无法处理大规模数据。为了解决这一问题，研究者提出了许多轨迹表示学习方法来估计度量空间，同时降低相似度计算的复杂度。然而，这些工作是基于 RNN 设计的，导致在长轨迹上性能严重下降。在本文中，我们提出了一种新的基于图的方法 TrajGAT，来显式地建模分层空间结构，提高长轨迹相似度计算的性能。TrajGAT 主要包括两个模块，即图构造模块和轨迹编码模块。对于图的构造，TrajGAT 首先使用 PR 四叉树构建整个空间区域的分层结构，然后基于原始轨迹记录和四叉树的叶子节点为每条轨迹构造一个图。对于轨迹编码，我们将 Transformer 中的自注意机制替换为图注意力机制，并设计了一个编码器来表示生成的图轨迹。通过这两个模块，TrajGAT 可以捕获轨迹的长距离依赖关系，同时减少 Transformer 的 GPU 内存占用。在两个真实数据集上的实验表明，TrajGAT 不仅提高了长轨迹上的性能，而且在混合轨迹上的性能显著优于最先进的方法。

关键词：轨迹相似度计算；长距离依赖；图注意力网络；transformer

1 引言

轨迹相似度计算是时空数据分析中的一项重要工作。经典的相似度计算方法如 DTW[1]、Hausdorff[2]、ERP[3]等被提出用于量化轨迹的内在相似度，可应用于轨迹聚类、位置预测、异常检测等。然而，这些方法的复杂度为平方级，难以应用在大规模轨迹数据分析中。为了解决这个问题，人们提出了各种相似性度量的策略，例如局部敏感哈希的 Hausdorff[4]、带窗口限制的 DTW[5]等。然而，这些方法都是为一种特定的场景而设计的，不适用于其他的场景。深度表示学习 (Deep representation learning, DRL) 近年来已成功应用于轨迹相似度计算。为了度量相似性，DRL 用向量表示轨迹，并学习向量的度量空间。与传统的方法相比，DRL 方法速度快，且适用于各种场景。然而，我们评估了现有的 DRL 方法在 top-K 相似性搜索上的性能，发现它们在长轨迹上的性能显著下降。如图 1所示：最先进的方法 (即 NeuTraj 和 Traj2SimVec) 的 top-10 命中率在长轨迹上至少下降了 40%。对基于 DRL 的方法而言，解决该问题至关重要。

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

图 1: 不同 DRL 方法在西安轨迹数据集上的性能比较

现有的 DRL 方法无法对长距离的轨迹依赖进行建模，这导致了在长距离轨迹上的性能下降。一方面，根据相似性度量的定义，两个轨迹的相似度通常由轨迹点的对齐方式决定，而长轨迹和短轨迹之间的对齐跨越了不同的区域。目前的方法使用循环神经网络 (RNNs) 在保持相似关系的同时将轨迹进行编码嵌入。这些 RNN 模型采用时间反向传播 (BPTT) 进行优化，难以扩展到长距离序列上[6]。因此，现有模型无法获取长轨迹的对齐信息。另一方面，DRL 方法通过学习等大小网格的共享表示来对空间信息建模[7]，导致序列建模中位于较远位置的轨迹点之间存在弱连接。此外，轨迹点在空间上分布不均匀。一些网格单元缺乏足够的数据来训练它们的表示，这进一步恶化了在长轨迹上的性能。因此，获取长轨迹的长距离依赖关系对于计算相似度至关重要。

然而，轨迹相似度计算的长距离依赖关系建模并非易事。现有的长轨迹序列建模工作可以分为三类，即基于 RNN 的方法[8]、基于记忆网络的方法[9]和基于 Transformer 的方法[10]，但它们都不适用于我们的任务。对于基于 RNN 的方法，在优化时使用了辅助损失，这不仅使模型难以训练，而且导致在度量近似上的性能不佳。基于记忆网络的方法依赖于记忆结构的启发式设计，通常无法捕获顺序关系。Transformer 已经证明了它在捕获长距离依赖方面的优越性。但是，随着序列长度的增加，Transformer 对 GPU 内存的需求呈平方级增长。

在本文中，我们提出了一种新的方法，即 TrajGAT，以获取轨迹相似度计算的长距离依赖关系。TrajGAT 在轨迹编码时考虑层次空间结构，不仅明确模拟了长轨迹的跨区域关系，而且减少了 Transformer 中自注意力机制对 GPU 的内存需求。具体来说，TrajGAT 首先采用 PR 四叉树[11]构建层次结构。所有四叉树叶子网格中的位置记录都是平衡的，这保证了网格表示的等效训练。基于四叉树，我们通过在原始记录和相关网格之间包含额外的边来构造所有轨迹的图。然后，设计了一种基于图注意力 (GAT) 的 Transformer，将轨迹图编码为向量。基于 GAT 的 Transformer 不再计算所有成对记录的注意力，而是只在轨迹图的边缘聚集信息，以降低 GPU 的内存成本。最后，将嵌入向量输入度量学习框架来计算相似度。本文的主要贡献总结如下：

- 提出了一种解决长轨迹相似度计算性能下降问题的新方法。据我们所知，TrajGAT 是第一个可以捕捉长距离依赖的、用于 GPS 轨迹建模的深度学习方法。
- 我们设计了基于 GAT 的 Transformer，它考虑了层次空间结构，并将轨迹转换为图结构进行轨迹编码。它可以模拟跨区域的关系，同时减少 GPU 内存的使用。
- 在两个公共轨迹数据集上的大量实验表明，TrajGAT 不仅能提高长轨迹上的相似度计算性能，而且在混合轨迹上的相似度计算性能优于现有方法。

2 相关工作

在本节中，我们从三个角度概述现有的相关研究:(1) 轨迹相似度计算;(2) 长序列建模;(3) 基于图的 Transformer。

2.1 轨迹相似度计算

现有的轨迹相似度计算方法大致可分为两类，非深度学习方法和深度学习方法。大多数非深度学习方法[4,12-13]将每个轨迹视为空间曲线，并使用几何的方法来简化计算。这些技术主要依赖于人工制定的启发式规则，可能会导致性能较差。此外，它们是针对特定距离而设计的，因此很难将这些技术

用于其他相似性度量。近年来，随着 AI[14]的发展，许多基于深度学习的方法[7,15-16]被提出，将轨迹的时空特征嵌入到向量中，并计算相似度。然而，这些方法依赖于 RNN 进行序列建模，并利用 BPTT 进行参数优化。虽然这些方法在短轨迹数据集上表现良好，但在长轨迹数据集上性能下降。

2.2 长序列建模

为了对长距离依赖进行建模，现有的工作可以分为三类，即基于 RNN 的方法、基于记忆网络的方法和基于 Transformer 的方法。通过添加无监督损失，Trinh 等人[17]提高了 RNN 捕获长距离依赖关系的能力。Francois 等人[8]提出了一个长距离依赖的原则性估计程序，并设计了一个用于长序列建模的 EvolutiveRNNs。这些基于 RNN 的方法还涉及外部优化目标，这将使模型难以训练且不能达到最好的性能。基于记忆网络的方法[9,18]通过共享记忆张量对长距离依赖进行建模。然而，记忆张量的结构是基于人工设计的，扩展性不强。最近，许多基于 Transformer 的方法[19-20]被提出用于长序列建模。但是这些模型的 GPU 内存成本随着序列长度的增加呈平方级增加，这限制了它们的应用场景。此外，现有的长序列建模模型都无法捕获空间信息，这对轨迹相似度计算至关重要。

2.3 基于图的 Transformer

许多研究[21-22]探索了将图神经网络与 Transformer 相结合来获取序列的结构信息。然而，这些方法都需要对轨迹数据进行图的建模，而这在轨迹数据中是不容易得到的。有一些工作是为顺序数据构造图，例如句子[23]。遗憾的是，由于时空特征的特殊性，这些方法无法对轨迹进行建模。

3 本文方法

在本节中，我们将详细介绍 TrajGAT。它包括三个关键模块: 层次结构建模、基于图的轨迹编码和基于度量学习的优化。

3.1 层次结构建模

如图 2所示，我们首先利用 $\mathcal{T}$ 中的轨迹构建四叉树 $\mathcal{H}$，用于表示层次空间结构。然后，对网格单元在 $\mathcal{H}$ 中的嵌入进行预训练。以下是对这两个过程的描述。

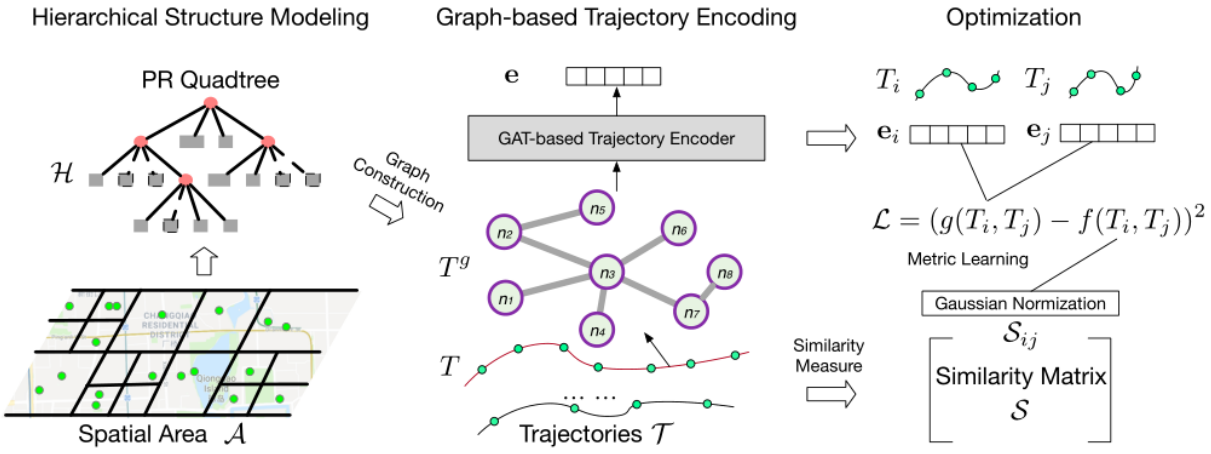

图 2: TrajGAT 架构图

TrajGAT 采用 PR 四叉树[11]对层次空间信息进行建模。如图**??**所示，它递归地将区域分解为四个相等的象限、子象限等。对于区域 $\mathcal{A}$ 中的 $\mathcal{T}$，我们用 PR 四叉树建立层次结构，如下:

1. 我们首先从所有轨迹中提取位置记录。
2. 根据经验设置一个阈值 δ，我们递归地分解区域并将子区域作为子节点，直到没有超过 δ 个

位置的叶子节点。

3. 最后我们省略了叶子节点的位置列表，只使用 $\mathcal{A}$ 的分层分区作为分层结构。我们将该四叉树记为 $\mathcal{H}$。

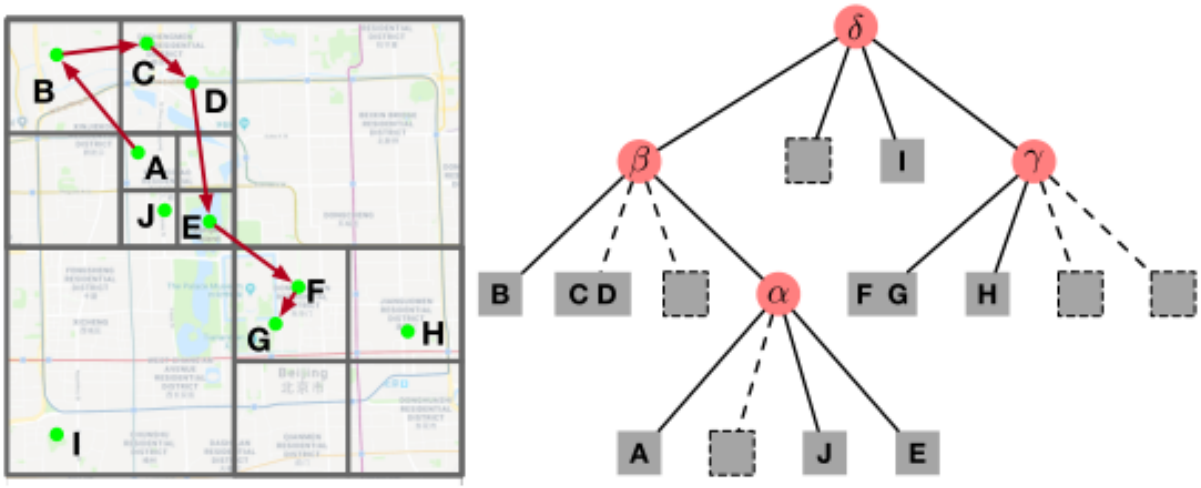

图 3: 用于层次结构建模的 PR 四叉树

为了整合层次信息进行轨迹编码，我们对 $\mathcal{H}$ 进行预训练，得到网格单元 $\mathbf{M}_{\mathcal{H}}$ 的嵌入。假设 $\mathcal{H}$ 中存在 K 个网格单元，我们将 $\mathcal{H}$ 视为一个图，并使用 Node2Vec 对其进行处理，以获取其编码。

3.2 基于图的轨迹编码

对于轨迹编码，我们首先为 $\mathcal{T}$ 中的每条轨迹构造一个图，然后提出了一个基于 GAT 的 Transformer 来生成图的嵌入。给定一个轨迹 T，我们通过考虑 $\mathcal{H}$ 中的网格单元来构造一个图 $T^g = (\mathrm{N}, \mathrm{E})$，其中 N 是节点集，E 是边集。如图 4所示，$T^g$ 不仅包含原始轨迹点记录，还包含 $\mathcal{H}$ 的层次结构信息。下面我们将分别详细介绍如何构建图结构和节点特征。

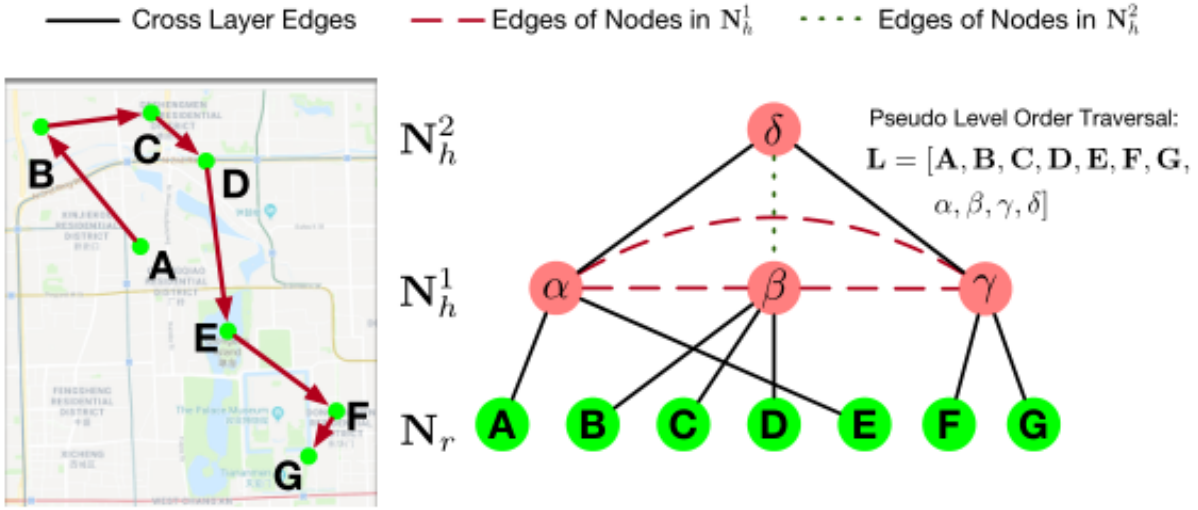

图 4: 轨迹图的构造过程

3.2.1 图的构建

在这一部分中，我们将依次介绍节点和边的构造过程。T^g 中的节点是从 T 的原始轨迹点以及 $\mathcal{H}$ 中与其相关的不同层的网格单元中提取出来的，即 $\mathbf{N} = \mathrm{N}_r \cup \mathrm{N}_h$。具体来说，假设 T 的长度为 L，我们首先为所有轨迹点构建节点，即 $\mathbf{N}_r = \{\mathbf{n}_1, \ldots \mathbf{n}_L, \}$。每个节点 n_i 包含原始轨迹点的所有信息。以 N_r 为第 0 层节点，我们递归地包含与前一层节点相关的层次网格单元。如图 4所示，在构造第 1 层节点

时，TrajGAT 根据第 0 层节点相对于 $\mathcal{H}$ 的叶子网格单元的坐标对第 0 层节点进行分组。对于从 $\mathcal{H}$ 中提取的所有节点，我们记其为：$\mathbf{N}_h = \{\mathbf{N}_h^1, \ldots, \mathbf{N}_h^\eta\}$，其中 η 为提取的层数，$\mathbf{N}_h^i$ 为每一层的子集节点。

在构造了 T^g 中的节点之后，我们构造了边 E。我们在 T^g 中设计了两种边。第一个是跨层边 E_c，它连接来自不同层的节点。因为 $\mathcal{H}$ 的叶子网格单元是 $\mathcal{A}$ 的一部分，我们找到所有轨迹点的相关网格单元，并添加从 N_r 到 $\mathbf{N}_h^1$ 的边。相应地，从 $\mathcal{H}$ 的树形结构中提取其他交叉层的边。第二类边是层内边，为了提高 GPU 的内存效率和降低 TrajGAT 的计算成本，我们只在 N_h 中构造层内边。对于每一层 $\mathbf{N}_h^i$，我们在节点之间添加完全连接的边。值得注意的是，E 中的所有边都是无向的。图 4说明了构造图的过程。

3.2.2 节点特征的构建

我们将轨迹的图 $T^g = (\mathrm{N}, \mathrm{E})$ 视为齐次图，并同等的对待 N_r 和 N_h 中的节点。N 中的节点特征包括三个方面: 坐标特征 f^l，区域特征 f^r，层次结构特征 f^h。接下来，我们将分别描述它们的构造过程。

N 中的每个节点代表区域 $\mathcal{H}$ 中的一个空间元素，可以是一个位置，也可以是一个矩形区域。对于 N_r 中的节点，我们直接使用其相关轨迹点的位置作为相关位置。对于 N_h 中的节点，我们选择它们的中心位置作为相关位置。轨迹上的所有位置都是 GPS 坐标，我们首先用最小-最大归一化函数对其进行归一化，然后使用多层感知器 (MLP) 进行非线性变换。假设 $X_i = (\mathrm{lat_i}, \mathrm{lon_i})$ 是节点 $\mathbf{n}_i \in \mathrm{N}$ 的相关位置，位置特征构造如下:

$$\mathrm{x_i}, \mathrm{y_i} = \mathrm{Normalize}\left(\mathrm{lat_i}, \mathrm{lon_i}\right)$$

$$\mathrm{f}_i^l = \mathrm{MLP}\left(\mathrm{x_i}, \mathrm{y_i}\right)$$

此外，我们还利用区域特征对区域大小进行建模。设 n_i 是 N_h 中的节点，我们获取其相关网格单元的宽度 w_i 和高度 h_i，并进行非线性变换来获取区域特征。公式如下:

$$\mathrm{f}_i^r = \mathrm{MLP}\left(\mathrm{w_i}, \mathrm{h_i}\right)$$

对于层次特征，我们直接利用预先训练好的 $\mathcal{H}$ 的特征矩阵。对于 N_r 中的节点，我们在 $\mathbf{M}_\mathcal{H}$ 中增加一个特殊键，表示所有轨迹点的层次特征。这样，不同轨迹中空间临近的轨迹点通过共享相同的层次特征而被连接起来。

$$\mathrm{f}_i^r = \mathrm{Embedding}\left(\mathbf{M}_\mathcal{H}, \mathrm{n_i}\right)$$

综上，对于一个特定节点 n_i，我们将三部分特征拼接在一起，得到节点的特征:

$$\mathbf{f}_i = \mathrm{concat}\left(\mathbf{f}_\mathrm{i}^\mathbf{l}, \mathrm{f}_\mathrm{i}^\mathrm{r}, \mathbf{f}_\mathrm{i}^\mathbf{h}\right)$$

3.3 基于 GAT 的轨迹编码器

该编码器遵循了 Transformer 的主要思想，同时以图作为输入，解决了 GPU 内存占用高的问题。如图 5所示，其核心思想在于并不是每个轨迹点都需要与所有其他轨迹点进行相似度计算。与普通的 Transformer 相比，本模型有两个新颖的设计:(1) 位置编码，同时保留轨迹的顺序和位置信息;(2) 基于 GAT 的 Transformer，利用图注意力机制对长轨迹进行建模。下面将分别介绍这两种设计。

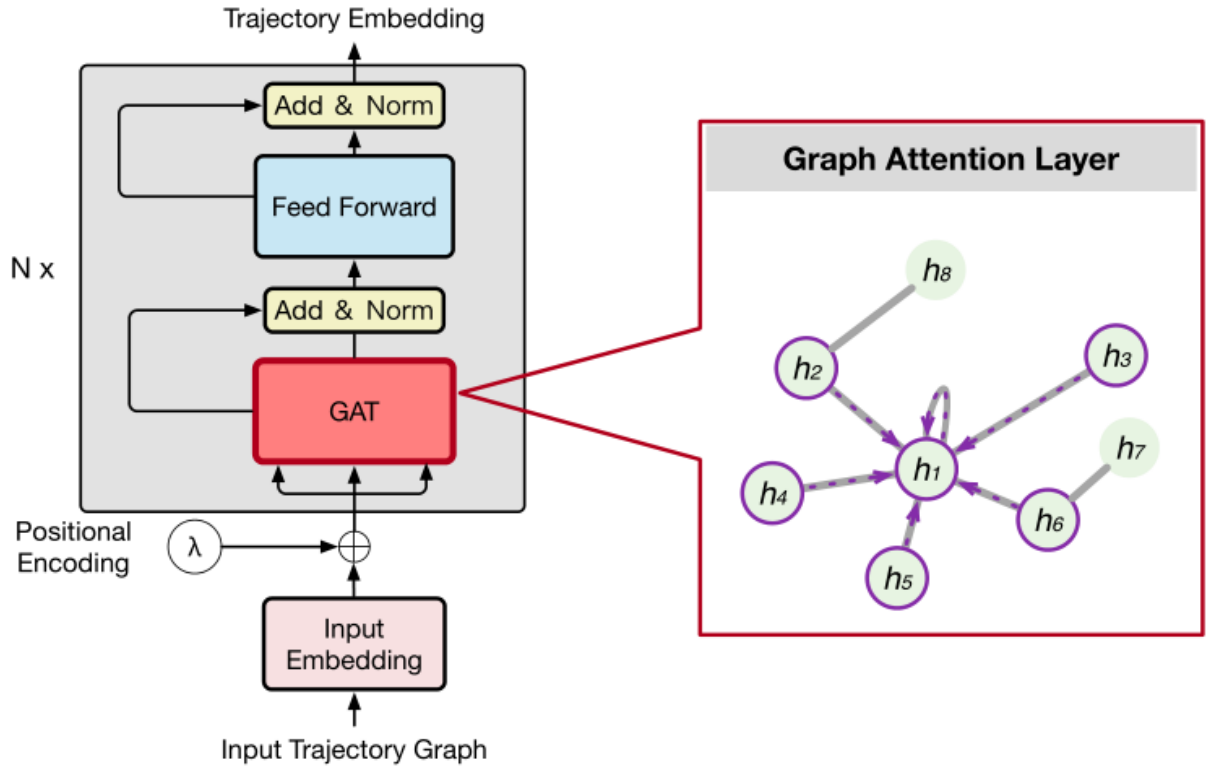

图 5: 基于 GAT 的 Transformer 架构

3.3.1 位置编码

我们首先使用一个空列表 L 来存储节点序列，并分别从 level-0 到 level-η 将 T^g 的节点添加进去。对于 level-0 (N_r) 中的节点，我们采用 T 的原始序列，将相关节点依次添加到 L 中。对于 level-i 中的节点，按照其在 N_{i-1}^h 中连接节点的出现顺序，将其添加到 L 中。L 的构造如图 4所示。对于 L 中的序列节点，我们使用 Vaswani 等[24]提出的正弦值来编码位置信息。我们记该位置编码为 λ_i^s。

为了对 T^g 中的图关系进行建模，我们使用拉普拉斯位置编码[25]来编码相对距离信息，即临近节点位置特征相似，较远节点位置特征不同。

$$\Delta = \mathrm{I} - \mathrm{D}^{-1/2}\mathrm{A}\mathrm{D}^{-1/2} = \mathrm{U}^T \Lambda \mathrm{U}$$

$$\lambda_i^g = \mathrm{MLP}\left(\mathrm{top_p}(\mathrm{U})\right)$$

其中 A 是邻接矩阵，D 是度矩阵，Λ 和 U 分别对应特征值和特征向量。λ_i 为节点 i 的拉普拉斯位置编码，top_p 为切片 Λ 的 p 个最小非平凡值的操作，它与节点 n_i 位置的特征向量相关。通过将特征向量输入到全连通层，我们得到图的位置编码。

在分别得到 λ_i^s 和 λ_i^g 后，将它们进行拼接，得到最终的位置编码向量 λ_i，并将其与节点特征相加。

$$\lambda_i = \mathrm{concat}\left(\lambda_{\mathrm{i}}^{\mathrm{s}}, \lambda_{\mathrm{i}}^{\mathrm{g}},\right)$$

$$\mathbf{i}_i = \lambda_i + \mathbf{f}_i$$

3.3.2 基于 GAT 的 Transformer 层

由于自注意机制，vanilla Transformer 需要计算所有的成对注意力权重，并且存在 GPU 内存占用高的问题。对于长轨迹，这个问题可能更严重。因此，以 i_i 作为输入 h_i^0，我们将自注意力层替换为图注意力层，得到轨迹的嵌入。对于一个特定的节点 n_i，基于 GAT 的 Transformer 层的定义如下:

$$\mathbf{h}_i'^{\ell+1} = \mathbf{O}_h^\ell \,\text{concat}_{k=1}^{H}\left(\sum_{j\in\mathcal{N}_i} \mathbf{w}_{ij}^{k,\ell}\mathbf{V}^{k,\ell}\mathbf{h}_j^\ell\right)$$

$$\text{where, } \mathbf{w}_{i,j}^{k,\ell} = \text{softmax}_j\left(\frac{\mathrm{Q}^{k,\ell}\mathbf{h}_i^\ell \cdot \mathbf{K}^{k,\ell}\mathbf{h}_j^\ell}{\sqrt{d_k}}\right),$$

在经过 P 层基于 GAT 的 Transformer 的处理后，我们得到了所有节点的特征表示，即 $\left[\mathbf{h}_1^P, \ldots, \mathbf{h}_i^P, \ldots, \mathbf{h}_M^P\right]$ 为了生成轨迹图 T^g 的最终特征，我们在节点特征上使用均值读出函数。轨迹 T 的嵌入 e 计算为：$\mathbf{e} = \sum_{i=1}^{M} \mathbf{h}_i^P / M$。基于图的轨迹编码的输出是轨迹特征，可用于度量轨迹的相似性。

3.4 基于度量学习进行优化

TrajGAT 基于轨迹图的嵌入，采用深度度量学习框架对模型参数进行优化。已知包含 $\mathcal{T}$ 中轨迹对的成对距离的距离矩阵 $\mathcal{D}$，我们按照[26]中介绍的方法，首先将 $\mathcal{D}$ 转变为相似度矩阵 $\mathcal{S}$，并将 $\mathcal{S}$ 作为监督信息，即 $\mathcal{S}_{ij} = \dfrac{\exp\left(-\theta\mathcal{D}_{i,h}\right)}{\max(\exp(-\theta\mathcal{D}))}$，其中 θ 是控制相似值分布的可调节参数。然后依次取 $\mathcal{T}$ 中的轨迹作为锚点轨迹进行模型优化。假设对 T_a 采样 n 条轨迹以拟合相似点，我们计算 T_a 的损失如下:

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

4 复现细节

轨迹数据挖掘领域使用的数据通常为 GPS 数据，我们通常将轨迹定义为：

$$\text{Tra} = \left\{\left(\ell_1, t_1\right), \left(\ell_2, t_2\right), \ldots, \left(\ell_n, t_n\right)\right\}$$

其中 ℓ_i 代表该轨迹点的位置，通常由经度和纬度构成，t_i 代表该轨迹点的时间。

在分析轨迹数据时，我们通常会分空间和时间两个维度来考虑。本文显然只考虑了空间维度的信息，一方面没有充分利用数据的特征，另一方面可能会造成相似度计算的结果不够准确。例如两辆车在不同的时段分别经过同一路段，如果只考虑空间维度信息，就会赋予两条轨迹一个较高的相似度，但实际上两条轨迹在时间维度上的相似度并不高。

2021 年，Fang 等[27]提出了一种新的基于表示学习的路网时空轨迹相似度计算方法 ST2Vec，是第一个基于深度学习的时空相似性计算方法。ST2Vec 能够根据不同的使用场景，为空间维度特征和时间维度特征灵活地分配权重。通过对比可以发现，TrajGAT 的优势在于基于图结构进行建模，且应用了 Transformer encoder 的结构进行编码，能有效避免 RNN 类模型导致的长距离退化问题；其缺点在于没有考虑时间维度信息。而 ST2Vec 的优势在于分别对空间维度和时间维度的特征进行建模，之后通过融合机制为二者分别权重，且其时间建模模块有着良好的通用性，可以与其他空间建模方法相结合，缺点在于提取特征时使用了 LSTM，应用到长距离轨迹时会出现性能衰退。因此，我们考虑将两种方法相结合，将 ST2Vec 的时间建模模块与 TrajGAT 结合。下面将介绍时间建模模块 TMM（Temporal Modeling Module）。

4.1 时间嵌入

受 BERT[28]中的位置嵌入的启发，对于时间序列中的每个时间点 t，我们学习其时间嵌入 t'。

$$t'[i] = \begin{cases} \omega_i t + \varphi_i, & \text{if } i = 0 \\ \cos\left(\omega_i t + \varphi_i\right), & \text{if } 1 \leqslant i \leqslant q \end{cases}$$

4.2 时间序列嵌入

我们将 $\langle t'_1, t'_2, \ldots, t'_m \rangle$ 输入到 LSTM 中以建模其时间依赖：

$$h_i = \text{LSTM}\left(t'_i, h_{i-1}, i_i, f_i, o_i, m_i\right)$$

其中，i_i，f_i，o_i，s_i 分别表示输入门、遗忘门、输出门和存储单元。LSTM 单元利用上一步中的时间嵌入、隐藏状态和单元状态来计算新的隐藏状态并更新单元状态。最后，我们将最后一个隐藏状态 h_t 作为时间序列的表示，因为它包含了轨迹所有时间点的信息。

4.3 注意力机制

轨迹上不同的时间点在计算中应当有着不同的权重，我们使用注意力机制来捕捉轨迹点之间的相关性，以提高模型的有效性。具体来说，我们提出了一种自注意机制来计算同一轨迹上时间点之间的注意分数：

$$\tilde{h}_i^{(p)} = \sum_{k=1}^{i} \text{att}\left(h_i^{(p)}, h_k^{(p)}\right) \cdot h_k^{(p)}$$

其中，$\tilde{h}_i^{(p)}$ 代表改进后的状态表示，是一个注意力函数：

$$\text{att}\left(h_i^{(p)}, h_k^{(p)}\right) = \frac{\alpha_{i,k}}{\sum_{k'=1}^{i} \exp\left(\alpha_{i,k'}\right)}$$

其中，$\alpha_{i,k} = w_1^{\top} \cdot \tanh\left(W_1 \cdot h_k^{(p)} + W_2 \cdot h_i^{(p)}\right)$，$w_1$、$W_1$、$W_2$ 是要学习的参数向量和矩阵。通过将注意力机制纳入时间序列嵌入，我们可以发现更多重要的时间点，进而改善模型性能。需要注意的是，我们仍然使用最后一步的隐藏表示来编码整条时间轨迹。

5 实验

5.1 实验配置

为了评估 TrajGAT 的性能，我们通过 Top-K 轨迹相似度搜索和轨迹聚类两个任务进行实验，并将 TrajGAT 的性能与其他流行的基于深度学习的轨迹相似度模型进行了比较。本实验使用的数据集为波尔图市出租车数据集[29]，轨迹类型为长距离轨迹，在度量学习中选择的传统度量方法为豪斯多夫距离。TrajGAT 采用三层基于 GAT 的 Transformer，每层的 GAT 有 8 个注意力头，轨迹的特征维度为 32 维。

5.2 实验结果

代码运行在服务器上，调用了一块显存为 11G 的 GeForce RTX 2080Ti，训练 100 个 epoch，训练总时长为 76 小时 53 分钟。实验结果如下：

表 1: 实验结果

Dataset	Method	Hausdorff on Long		
		HR@10	HR@50	R10@50
Porto	traj2vec	0.0291	0.0715	0.1706
	t2vec	0.0429	0.0775	0.1361
	NeuTraj	0.1020	0.1774	0.2700
	Transformer	0.1265	0.2057	0.3547
	TrajGAT	**0.5344**	**0.6350**	**0.9037**
	ours	0.4870	0.6114	0.8811

5.3 实验结果分析

表中 traj2vec、t2vec、NeuTraj、Transformer 为四种 baselines 的结果，TrajGAT 为原文结果，ours 是修改后结果。TrajGAT 在 HR@10、HR@50 和 R10@50 这三项指标中均取得了最好的成绩，ours 虽略逊于 TrajGAT，但也远优于其他 baselines 的表现。ours 略逊于 TrajGAT 的可能原因分析如下：首先，由于隐私原因，原文作者未分享原数据集，而是提供了处理过的.pkl 文件。其次，由于资源有限，ours 在训练时只使用了一块显存为 11G 的 GeForce RTX 2080Ti，训练了 100 个 epoch，训练过程需要耗费大量的时间，同时，时间建模模块 TMM 对机器 RAM 的要求也比较高，运行时会导致内存不足。但是，ours 的整体表现仍是远好于其他四个 baselines，因此也证明了 TrajGAT 对空间特征的建模以及 TMM 对时间特征的建模的有效性。

6 总结与展望

我们观察到以往的基于深度学习的方法在长距离轨迹上会出现性能衰退的问题，为了解决这一问题，提出了一种基于图的方法 TrajGAT。本方法明确地对 PR-四叉树构建的空间区域层次结构进行建模，并基于四叉树的网格单元构建轨迹的图结构。之后设计了基于 GAT 的 Transformer 来捕获轨迹的空间和序列信息，同时降低 GPU 的内存需求。此外，考虑到原本的 TrajGAT 并未考虑时间维度的特征，我们在此基础上增加了时间建模模块 TMM 作为创新点，以更加深入的挖掘轨迹数据在时间维度上的特征。最后，通过对比实验证明了模型的有效性，表现优于其他对照方法。

在今后的工作中，我们计划根据特定的下游任务探索更多的时空特征建模方法，并将所提出的框架扩展到其他类别的轨迹数据，例如 POI 签到轨迹等，以支持更多的应用场景。

参考文献

[1] YI B K, JAGADISH H V, FALOUTSOS C. Efficient retrieval of similar time sequences under time warping[C]//Proceedings 14th International Conference on Data Engineering. 1998: 201-208.

[2] ATEV S, MILLER G, PAPANIKOLOPOULOS N P. Clustering of vehicle trajectories[J]. IEEE transactions on intelligent transportation systems, 2010, 11(3): 647-657.

[3] CHEN L, NG R. On the marriage of lp-norms and edit distance[C]//Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. 2004: 792-803.

[4] DRIEMEL A, SILVESTRI F. Locality-sensitive hashing of curves[J]. arXiv preprint arXiv:1703.04040, 2017.

[5] RAKTHANMANON T, CAMPANA B, MUEEN A, et al. Searching and mining trillions of time series subsequences under dynamic time warping[C]//Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. 2012: 262-270.

[6] ZHAO J, HUANG F, LV J, et al. Do RNN and LSTM have long memory?[C]//International Conference on Machine Learning. 2020: 11365-11375.

[7] LI X, ZHAO K, CONG G, et al. Deep representation learning for trajectory similarity computation[C]//2018 IEEE 34th international conference on data engineering (ICDE). 2018: 617-628.

[8] BELLETTI F, CHEN M, CHI E H. Quantifying long range dependence in language and user behavior to improve RNNs[C]//Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019: 1317-1327.

[9] CHANG Y Y, SUN F Y, WU Y H, et al. A memory-network based solution for multivariate time-series forecasting[J]. arXiv preprint arXiv:1809.02105, 2018.

[10] DAI Z, YANG Z, YANG Y, et al. Transformer-xl: Attentive language models beyond a fixed-length context[J]. arXiv preprint arXiv:1901.02860, 2019.

[11] SAMET H. An overview of quadtrees, octrees, and related hierarchical data structures[J]. Theoretical Foundations of Computer Graphics and CAD, 1988: 51-68.

[12] AGARWAL P K, FOX K, PAN J, et al. Approximating dynamic time warping and edit distance for a pair of point sequences[J]. arXiv preprint arXiv:1512.01876, 2015.

[13] BACKURS A, SIDIROPOULOS A. Constant-distortion embeddings of hausdorff metrics into constant-dimensional l_p spaces[C]//Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016). 2016.

[14] XU Y, LIU X, CAO X, et al. Artificial intelligence: A powerful paradigm for scientific research[J]. The Innovation, 2021, 2(4): 100179.

[15] HAN P, WANG J, YAO D, et al. A graph-based approach for trajectory similarity computation in spatial networks[C]//Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2021: 556-564.

[16] WANG Z, LONG C, CONG G, et al. Effective and efficient sports play retrieval with deep representation learning[C]//Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019: 499-509.

[17] TRINH T, DAI A, LUONG T, et al. Learning longer-term dependencies in rnns with auxiliary losses[C]//International Conference on Machine Learning. 2018: 4965-4974.

[18] FERNANDO T, DENMAN S, MCFADYEN A, et al. Tree memory networks for modelling long-term temporal dependencies[J]. Neurocomputing, 2018, 304: 64-81.

[19] WU H, XU J, WANG J, et al. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting[J]. Advances in Neural Information Processing Systems, 2021, 34: 22419-22430.

[20] ZHOU H, ZHANG S, PENG J, et al. Informer: Beyond efficient transformer for long sequence time-series forecasting[C]//Proceedings of the AAAI conference on artificial intelligence: vol. 35: 12. 2021: 11106-11115.

[21] HU Z, DONG Y, WANG K, et al. Heterogeneous graph transformer[C]//Proceedings of the web conference 2020. 2020: 2704-2710.

[22] YUN S, JEONG M, KIM R, et al. Graph transformer networks[J]. Advances in neural information processing systems, 2019, 32.

[23] YE Z, GUO Q, GAN Q, et al. Bp-transformer: Modelling long-range context via binary partitioning[J]. arXiv preprint arXiv:1911.04070, 2019.

[24] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[25] DWIVEDI V P, JOSHI C K, LAURENT T, et al. Benchmarking graph neural networks[J]., 2020.

[26] YAO D, CONG G, ZHANG C, et al. Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach[C]//2019 IEEE 35th international conference on data engineering (ICDE). 2019: 1358-1369.

[27] FANG Z, DU Y, ZHU X, et al. ST2Vec: Spatio-Temporal Trajectory Similarity Learning in Road Networks[J]. arXiv preprint arXiv:2112.09339, 2021.

[28] WANG B, SHANG L, LIOMA C, et al. On position embeddings in bert[C]//International Conference on Learning Representations. 2021.

[29] MOREIRA-MATIAS L, GAMA J, FERREIRA M, et al. Time-evolving OD matrix estimation using high-speed GPS data streams[J]. Expert systems with Applications, 2016, 44: 275-288.