从大规模轨迹中获取代表性路径

刘晓思

摘要

本文主要研究了如何在大规模的轨迹数据中找到 k 条最具代表性的路径，这个问题有益于许多城市应用，比如交通管控以及公共交通规划等，交通规划必须跟据交通流量的大小以及连通性改变规划结果。因为这是一个 NP 难问题，因此，本文提出了一个能快速得到结果的近似解范围。具体而言，先构建一个查找表，存储每一条路段对应的覆盖轨迹。在计算路径的时候，并没有使用深度优先搜索，而是通过权值计算，想办法获取最大的路径权重，即提出了最大权重算法。但是由于路径的每条路段覆盖多条轨迹，因此进一步提出覆盖优先算法来获取路径集合中由最大增益的路段。而后通过观察在现实世界中，每条边都只与路网中其他几条路段相连，因而设计了一种连接优先算法。最后，在数据集上进行实验，验证了三个算法的有效性和时间效率。

关键词：代表性路径；轨迹；路网

1 引言

随着城市交通应用的发展，生成了大规模的轨迹数据，在提高交通管理和袁术规划等许多下游任务的准确性方面进行了大量的研究[1-4]。尽管在许多交通问题上取得了不错的进展，但现有的工作并没有解决如何在地图上找到能够代表多条轨迹的路径问题。获取到能够代表许多轨迹的路径有益于许多实际的城市应用。例如，考虑到以下两个场景：一是交通监控，二是公共交通规划。

交通管理部门必须通过监控查看交通流量，快速识别车流量趋势和交通异常情况，并深入了解当前的交通模式。尽管现有的轨迹聚类方案[5-6]能够使用聚类质心作为代表性路径，但是聚类的结果不一定是路网中的真实道路，而且其连接性不能够保证。此外，轨迹聚类方案运行时间上成本较高，难以捕获实时的交通流量变化。为了实时发现通用的交通模式，代表性路径需要是真实路网上的路段，并且需要在路网上连通，以适应交通流的变化。

交通管理部门经常需要更新公交路线，以简化交通网络，并适应人们出行的偏好。即使需要定期做这些任务，但公交路线通常是手动规划和更新的，或者规划者通过小规模的数据集来进行规划[7-8]，这种方式需要该领域的专家通过手动比较区域内的所有道路，选择比较合适的路径。因此，能够从大规模数据集中找到代表性路径能使公交线路规划问题得到更加准确和灵活的解决。

给定一组轨迹、路网、距离阈值和成本预算，搜索到的代表性路径问题需要满足以下三个要求：(1) $Representativeness$ (代表性)：与代表性路径的距离满足小于距离阈值的轨迹数量最大。(2) $Connectivity$ (连通性)：代表性路径只包含路网中连通的边。(3) $Budget\ constraint$ (预算约束)：代表性路径的成本应该低于所设定的预算阈值，并非是路径总成本越小越好。

搜索一个区域大规模轨迹的代表性路径问题是 NP 难问题，因此在精确的计算上是不可行的。解决这个问题主要难点在于计算哪些轨迹接近路网上的候选代表性路径。一个直观的解决方案是，先进行代表性轨迹计算，再进行地图匹配。但是，地图匹配方法存在几个限制：(1) 原始轨迹存在误差，可

能包含地图上不连通的两个轨迹点，导致一条路径分割成两条。(2) 地图匹配计算难以实现实时更新，在线地图匹配计算量非常大。因此，需要设计一种高效且能搜索到结果比较好的算法来解决这个问题。

为了解决这挑战，本文构建了一个查找表的结构，用来存储路网中的路段对应的覆盖轨迹数，有益于消除重复的覆盖计算。由于不能确定搜索代表性路径的最优解，本文设计了最大权重算法 (Maximum-weight algorithm) 来计算近似的最优解。而后，为了评估路径中每条路段的代表性，本文设计了覆盖优先算法 (Coverage-first algorithm) 来贪心地选择最大化路径的覆盖数量，使搜索到的路径集合更具有代表性。此外，为了进一步提高算法效率，本文设计了连接优先算法 (Connect-first algorithm)，通过剪枝操作，来寻找每条路径中有最大路段增益的路段，以此来寻找最具代表性的路径。

2 相关工作

2.1 轨迹聚类

现有的轨迹聚类工作可以分为两类：(1) 对所有的轨迹进行聚类[6,9-11]，(2) 对子轨迹进行聚类[5,12-16]。

第一类通常采用传统的聚类技术，如 k-means 和 DBSCAN。将每个轨迹看作单独的聚类对象，定义聚类的方式，寻找聚类中心作为代表性轨迹。Huang 等人[11]首先提出了一种”clue-aware” 轨迹聚类算法，通过轨迹的相似性将轨迹聚成一类，然后获取对应的聚类中心。Brankovic 等人[9]设计了基于中心的轨迹聚类算法，用以计算 l-简化的中心轨迹，其聚类中心为 l 轨迹。

第二类聚类方法考虑到对所有的轨迹聚类可能无法检测到轨迹中的相似子轨迹片段[5]，因此提出了 partition-and-group 框架用于子轨迹聚类。Agarwal 等人[12]将每个簇建模为一条路径，并将路径进行分组，组成 k 条最具代表性路径。其他一些研究[15-16]用投票方式对代表性子轨迹进行抽样，然后使用其子轨迹进行分组。在许多已有工作中，代表性轨迹是聚类中心，一般不是坐落于路网的轨迹。而本文提出的算法能够生成基于路网的路径。

2.2 轨迹模式挖掘

轨迹模式挖掘用于识别历史轨迹数据中的频繁轨迹模式。一些研究[17-20]使用顺序模式挖掘算法探索两个目标位置之间的频繁轨迹模式。其他研究[21-24]设计了用于频繁轨迹模式挖掘的新算法。例如，Sacharidis 等人[23]提出了一种在线算法，以保持多个物体经常跟随的轨迹模式。Zheng 等人[24]使用基于最长公共子序列距离的 DBSCAN 聚类算法提取大规模轨迹数据的交通流模式。然而，这些方法并不能用于本文提出的问题，一是因为本文提出的搜索最具代表性路径并没有指定路径应该经过的位置，而轨迹模式挖掘问题中的搜索指定了应该经过的地点，二是轨迹模式挖掘给定轨迹现有模式，本文提出的代表性路径基于路网生成，不一定经过原始轨迹。

2.3 轨迹距离度量

常见的轨迹数据距离度量包括 Dynamic Time Warping (DTW)[25]、Hausdorff Distance[26]、Discrete Frechet Distance[27]、Edit Distance with Real Penalty (ERP)[28]、Edit Distance on Real Sequence (EDR)[29]和 Longest Common Subsequences (LCSS)[30]。然而，现有的基于点的测量都对位置误差、采样率和点的偏移敏感，在使用原始轨迹时，也会忽略方向信息。其他研究[31-32]仅仅基于方向来定义两条轨迹的距离，而本研究采用[5,33]的思想结合点和方向信息来计算距离度量。

表 1: 符号表

符号	描述
t, T	轨迹，轨迹集合
$G(V, E)$	路网图，顶点为 V，边为 E
c_e	路网边 $e \in E$ 的代价
r, R	路径，路径集合
$S(R)$	路径 R 的代表性得分
B	成本预算
r	距离阈值
$d(e, l)$	路段边 e 和轨迹线段 $l \in t$ 的距离

3 本文方法

3.1 问题定义

本小节将介绍文中常用的一些符号表示，具体如表 1所示.

轨迹点 $p = (x, y)$ 由纬度 x 和经度 y 组成。轨迹 t 是轨迹点序列 $\langle p_1, ..., p_n \rangle$。轨迹 t 的长度表示为 $|t|$。轨迹集合表示为 $T = \{t_1, ...t_n\}$。集合 T 的大小表示为 $|T|$ 一条有向线段表示为 $l = \boldsymbol{p_s p_e}$, 是起点为 $l.p_s$ 终点为 $l.p_e$ 的直线。l 的长度为 ℓ_l 路网图为 $G(V, E)$，V 是路网的路段交点，E 是路网路段。$e \in E$ 的两端为 $e.p_s$ 和 $e.p_e$。$e \in E$ 长度为 l_e。$e \in E$ 的代价为 c_e。路径 r 是 G 中的连续路段序列, 表示为 $r : e_1 \to e_2 \to ... \to e_m$。$r$ 的大小为 $|r|$ 。r 的长度表示为 $l_r = \sum_{i=1}^{m} \ell_{e_i}$，是所有路段长度的总和。$R = \{r_1, ...r_k\}$ 是路径集合，大小为 $|r|$。

为了更好获取轨迹的空间信息，本文使用更通用的距离计算方式[5,33]，具体而言，路段 e 和轨迹线段 l 的距离表示为 $d(e, l)$，包含三个方面的距离，一是垂直距离 $d_\perp(e, l)$，二是平行距离 $d_\parallel(e, l)$ 三是角度距离 $d_\theta(e, l)$。图 1展示了对应的距离计算方式。

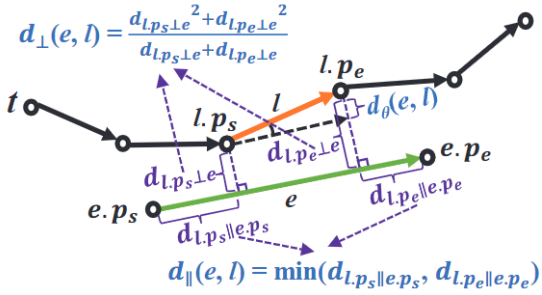

图 1: $d(e, l)$ 的计算方式

垂直距离：给定一条路段 e 和一条线段 l，其中，$l_e > \ell_l$，$d_{l.p_s\perp e}(d_{l.p_e\perp e})$ 为 $l.p_s(l.p_s)$ 与 e 的垂直距离。e 和 l 的垂直距离可由 $d_{l.p_s\perp e}$ 和 $d_{l.p_e\perp e}$ 计算得到，具体表达式为：

$$d_\perp(e, l) = ({d_{l.p_s\perp e}}^2 + {d_{l.p_e\perp e}}^2)/(d_{l.p_s\perp e} + d_{l.p_e\perp e}) \tag{1}$$

平行距离：给定一条路段 e 和一条线段 l，其中，$l_e > \ell_l$，$d_{l.p_s\parallel e.p_e}(d_{l.p_e\parallel e.p_e})$ 是 $l.p_s(l.p_s)$ 与 $e.p_e$ 对齐时的偏移量。e 和 l 的平行距离的表达式为：

$$d_\parallel(e, l) = min(d_{l.p_s\parallel e.p_e}, d_{l.p_e\parallel e.p_e}) \tag{2}$$

角度距离：给定一条路段 e 和一条线段 l，其中，$l_e > \ell_l$，e (或 l) 的方向表示为 $\theta(e)$ (或 $\theta(l)$), 是

$e.p_s$(或 $l.p_s$) 沿着 x 轴逆时针旋转到 $e.p_e$(或 $l.p_e$) 的角度。$\theta(e)$ 和 $\theta(l)$ 的角度差为：

$$\Delta(\theta(e),\theta(l)) = min\{|\theta(e)-\theta(l)|, 2\pi - |\theta(e)-\theta(l)|\} \tag{3}$$

e 和 l 的角度距离的表达式为：

$$d_\theta(e,l) = \begin{cases} \ell_l \times sin(\Delta(\theta(e),\theta(l))), & \Delta(\theta(e),\theta(l)) \in [0,\pi/2) \\ \ell_l, & \Delta(\theta(e),\theta(l)) \in [\pi/2,\pi] \end{cases} \tag{4}$$

定义 2.1 路段与线段的距离：给定一条路段 e 和一条线段 l，e 和 l 的距离表示为：

$$d(e,l) = d_\perp(e,l) + d_\parallel(e,l) + d_\theta(e,l) \tag{5}$$

由于轨迹的复杂性，很难保证代表性路径中每条边都满足与给定的所有轨迹距离较小。因此，轨迹和路径的关系用定义 2.2 表示。

定义 2.2 路径和轨迹的距离：路径与轨迹的距离：给定一条路径 r 和一条轨迹 t，r 和 r 的距离表示为路径中任意路段与轨迹中任意线段的最小距离，即：

$$d(r,t) = min_{e\in r, l\in t}(e,l) \tag{6}$$

当路径 r 与轨迹 t 的距离小于阈值 ε，即 $d(r,t) < \varepsilon$ 时，视为轨迹被路径覆盖。给定轨迹集合 T 和路径集合 R，当 R 中的路径能够更多地覆盖 T 中的轨迹，则视为 R 是 T 的比较好的代表性路径表示。

定义 2.3 代表性得分：给定轨迹集合 T 和路径集合 R，R 的代表性得分是其路径的覆盖总轨迹数与原始轨迹集合的轨迹数比值，表示为：

$$S(R) = \sum_{t\in T} \prod(t,R)/|T| \tag{7}$$

$$\prod(t,R) = \begin{cases} 1, & \exists r \in R, d(r,t) \leqslant \varepsilon \\ 0, & otherwise \end{cases} \tag{8}$$

定义 2.4 最具代表性路径搜索问题 ($MDDR$)：给定一组轨迹集合 T、路网 $G(V,E)$、成本预算 B 和目标路径数 k,$MDDR$ 的目标是找到 k 条最具代表性的路径 $R = \{r_1, ..., r_k\}$，S(R) 是路径集合的代表性得分，是在成本预算的控制下的路径的最大轨迹覆盖数，即 $R = argmax_{\sum_{e\in r} c_e \leqslant B \forall r\in R} S(R)$

3.2 查找表的设计

由于文中提出的算法需要频繁寻找路段和轨迹的覆盖关系，因此本文引入了一种数据结构叫查找表 I。由于路径的覆盖关系可以细分为路径中的路段与轨迹的覆盖关系，因此设计查找表预先存储路段与轨迹的覆盖关系可以减少路段与轨迹的搜索次数。查找表简单理解为是一个字典结构 $I = dict(key : value)$，其 key 是路段，$value$ 存储对应路段覆盖的轨迹。

3.3 Maximum-weight 算法

最大权重算法通过动态规划的方式寻找 k 条满足成本约束的理论上最优的路径。给定路网 $G = (V,E)$，构造一个边图 $\mathcal{G} = (\mathcal{V},\mathcal{E})$，其中，$v_i \in \mathcal{V}$ 表示路段 $e_i \in E$，有向边 $e_{ij} \in \mathcal{E}$ 表示两条路段的连接，$e_i \in E, e_j \in E$ 且 $e_i.p_e = e_j.p_s$。对于边 $e_{ij} \in \mathcal{E}$，权重 w_{ij} 表示 e_i 覆盖的轨迹数量，即 $w_{ij} = |\{t \in T | \exists l \in t, d(e_i,l) \leqslant \varepsilon\}|$。定义了一个根顶点 r 和底部顶点 b，其都是 $\mathcal{G}$ 中没有覆盖轨迹的路段顶点。然后从 $\mathcal{G}$ 中添加路段到 r 和 b 中。

Procedure 1 Maximum-weight 算法

Input: T, G, B and $G(V, E)$
Output: the k most representative routes R with representative score s^*

$R \leftarrow \emptyset$, $s^* \leftarrow 0$, $I \leftarrow \emptyset$, $\tau \leftarrow \emptyset$, remove e from E if $c_e < B$
build the lookup table I to maintain trajectories covered by each edge
construct a weighted adjacency matrix A based on I
while $\mid t \mid < k$ **do**
    initialize the state array mp
    for $i \leftarrow 0$ to $|A| - 1$ **do**
        for $j \leftarrow 0$ to $|A| - 1$ **do**
            retrieve the route r_j whose last edge is e_j by $mp[j]$
            if $A[i][j] \neq -1 \wedge c_r + c_e \leqslant B$ **then**
                $mp[i] = max(mp[i], mp[j] + A[i][j])$
            end if
        end for
    end for
    retrieve the route r by $mp[|A| - 1]$ and update A, R, and τ
end while
$s^* \leftarrow |\tau|/|T|$
return result

最大权重算法执行过程如 Procedure 1 所示。先初始化路径集合 R，代表性得分 s^*，被代表性路径覆盖的轨迹 $\mathcal{T}$，然后构造查找表 I 存储 $G(V, E)$ 中每条路段覆盖的轨迹。跟据查找表构造边图 $\mathcal{G}$ 和邻接矩阵 A，其中，$A[i][j] = w_{ij}$ 表示 e_i 和 e_j 的连通，如果不连通，则 $A[i][j] = -1$。用根顶点 r 作为矩阵第 0 行元素，其中，第一行元组的元素值全为 0，表示没有初始的时候没有轨迹覆盖。接着需要寻找 k 条有最大权重轨迹覆盖的轨迹。对于每条路径，首先初始化状态数组 mp 表示权重的累加值，然后用动态规划方式更新状态数组，规则为 $mp[i] = max(mp[i], mp[j] + A[j][i])$，对于更新后的路径，添加路径覆盖的新增轨迹到 $\mathcal{T}$ 中，查找完 k 条轨迹之后，更新代表性得分。

3.4 Coverage-first 算法

最大权重算法通过寻找 k 条路径 R，使路径覆盖的轨迹数最大化。但由于 R 最大加权路径不一定是覆盖轨迹数最多的路径，因此设计了覆盖优先算法来解决这个问题。

覆盖优先算法通过迭代计算当前路径中有最大增益的路段加入路径集中，来获取具有最大覆盖轨迹数的路径集合。覆盖优先算法执行过程如 Procedure 2 所示。先初始化路径集合 R，代表性得分 s^*，被代表性路径覆盖的轨迹 $\mathcal{T}$，然后构造查找表 I 存储 $G(V, E)$ 中每条路段覆盖的轨迹。在每次迭代中，将新增覆盖轨迹最多的边添加到路径中，直到无连接边或者超出成本预算值，然后搜索计算下一条最大覆盖轨迹路径。

3.5 Connect-first 算法

虽然覆盖优先算法能够获取最大轨迹覆盖数量的路径集合，但是由于其迭代次数比较多，导致时间开销比较大。通常来说，搜索路径的时候，只需要考虑少数与原路径相连接的若干条路径的情况，那些不相连的无法加入原路径中。因此在搜索最大覆盖路段增益边的时候，可以只考虑路径前后相连的两个路口结点相连的路段，通过贪心算法，来获取覆盖轨迹比较多的代表性路径，这种方法称为连接优先算法。

连接优先算法执行过程如 Procedure 3 所示。先初始化路径集合 R，代表性得分 s^*，被代表性路

Procedure 2 Coverage-first 算法

Input: T, G, B and $G(V, E)$
Output: the k most representative routes R with representative score s^*

$R \leftarrow \emptyset$, $s^* \leftarrow 0$, $I \leftarrow \emptyset$, $\tau \leftarrow \emptyset$, remove e from E if $c_e < B$
build the lookup table I to maintain trajectories covered by each edge
while $E \neq \emptyset$ **do**
    $e \leftarrow argmax_{e \in E} |I[e] \cup T| - |T|$
    $\Delta s \leftarrow 0, R' \leftarrow \emptyset, T' \leftarrow \emptyset, E \leftarrow E \setminus e, E_m \leftarrow E_m \cup e$
    for $r \in R$ **do**
        $r' \leftarrow r, T'' \leftarrow T, c'_r \leftarrow c_r, \Delta s \leftarrow 0, E'_m \leftarrow E_m, flag \leftarrow true$
        while flag **do**
            find edges $E_c \subseteq E'_m$ that connect with r
            remove e from E_c if $c'_r + c_e < B$
            if $E_c \neq \emptyset$ **then** $e \leftarrow argmax_{e \in E} |I[e] \cup T''| - |T''|$
            end if
            else $flag \leftarrow false$ and break
            end else
            update $r', T'', E'_m, \Delta s' \leftarrow |T''| - |\mathcal{T}|$
        end while
        if $\Delta s' > \Delta s$ **then** $\Delta s \leftarrow \Delta s', R' \leftarrow R \setminus r \cup r', T' \leftarrow T''$
        end if
    end for
    if $R' \neq R$ **then** $R \leftarrow R', T \leftarrow T'$,remove e in $r \in R$ from E_m
    end if
    else if $|R| < k$ **then** $R \leftarrow R \cup \{e\}, T \leftarrow I[e] \cup T, E_m \leftarrow E_m \setminus e$
    end else
end while
$s^* \leftarrow |T|/|\mathcal{T}|$
return result

径覆盖的轨迹 $\mathcal{T}$，然后构造查找表 I 存储 $G(V, E)$ 中每条路段覆盖的轨迹。在最开始搜索路径的时候，选择一条覆盖轨迹数量最多的路段，将其添加到路径中，然后更新路径前后结点相连的路段集合，在下一个迭代中，添加候选路段集合中新增覆盖轨迹数量最多的路段，直到没有新增覆盖轨迹的连接边或者超出成本预算，以此计算出 k 条路径，得到结果。

4 复现细节

4.1 与已有开源代码对比

原文只开源了 C++ 实现的 Connect-first 算法，在北京和 Porto 出租车数据集上进行实验。本项目用 Python 复现了文中提出的三个算法，并且在另外的数据集，成都出租车上进行实验，并且增加了 KMeans 聚类方式进行实验对比。具体而言，本项目重写了所有的地图以及轨迹的预处理方式，在进行距离计算的时候，原文代码通过经纬度投影到平面直角坐标系中进行计算，而本项目在 WGS84 坐标系下进行计算，角度距离计算的时候，计算的是相关的方位角，省去了全部坐标的投影操作，不仅提供了更加精确的计算，省去了许多计算步骤。

4.2 实验环境搭建

数据集是成都出租车轨迹数据，轨迹数据区域范围为 7464.3487×8313.2479m^2，共有 224288 条轨迹数据。地图用的是 OpenStreetMap 的地图，选取的地图范围是经度 104.037 104.129，纬度 30.653 30.732，包含路段数量为 20063 条。

Procedure 3 Connect-first 算法

Input: T, G, B and $G(V, E)$

Output: the k most representative routes R with representative score s^*

$R \leftarrow \emptyset$, $s^* \leftarrow 0$, $I \leftarrow \emptyset$, $\tau \leftarrow \emptyset$, remove e from E if $c_e < B$
build the lookup table I to maintain trajectories covered by each edge
while $|R| < k$ **do**
    $e \leftarrow argmax_{e \in E} |I[e] \cup T| - |T|$
    $r \leftarrow \{e\}, T \leftarrow I[e] \cup T$
    while $c_r < B$ **do**
        $E_c \leftarrow \{e \in E | r.p_s = e.p_e \vee r.p_e = e.p_s\}$
        $\Delta s \leftarrow 0, r' \leftarrow \emptyset, T' \leftarrow \emptyset, c'_r \leftarrow 0$
        for $e \in E_c$ **do**
            $\Delta s \leftarrow 0$
            if $c_r + c_e \leqslant B$ **then**
                $\Delta s' c_r + c_e |I[e] \cup T| - |T|$
            end if
            if $\Delta s' > \Delta s$ **then**
                $r' \leftarrow r \cup e, \Delta s \leftarrow \Delta s', T' \leftarrow I[e] \cup T, c'_r \leftarrow c_r + c_e$
            end if
        end for
        if $r' \neq r$ **then**
            $r \leftarrow r', c_r \leftarrow c'_r, T \leftarrow T'$
        end if
        else break
        end else
    end while
    $R \leftarrow R \cup r, E \leftarrow E \setminus e$, for each $e \in r$
end while
$s^* \leftarrow |T|/|\mathcal{T}|$
return result

实验在 python3.7 下运行，操作系统为 Windows11，处理器 12th Gen Intel(R) Core(TM) i9-12900K，内存大小 32G。

4.3 创新点

优化的查找表存储。在查找表加载使用的时候，原文直接对查找表进行搜索，特别是在挑选出路径的第一条路段的时候，需要查找覆盖轨迹数量最多的路段，而随机存储的查找表搜索起来时间很长，因此在查找表的建立和存储的时候，本项目使用了排序算法根据查找表 key 值对应的数量从大到小排序，因此查找覆盖轨迹最多的路段只需要取字典的第一个存储对象即可。

优化的路网搜索方案。一般来说，路网路段数量很多，在数据预处理部分，需要反复对路网的路段进行搜索，以便确定轨迹和路段的覆盖关系。而文章中普通做法就是进行遍历，这样很耗费大量时间去搜索。因此，在本项目中，使用了网格划分，具体而言，对路网数据，根据路网边界进行特定长度的网格划分，代码中使用的是网格边长为 50m 的划分，然后通过路网边界和网格序号，能确定一小块地图范围，然后预先根据小的地图网格，可以确定网格附近的路段，并构建字典存储下来，在轨迹与路段计算的时候，就可以根据网格字典很快确定候选的周围路段了，不需要一遍一遍进行遍历。

优化的距离计算方式。前面也提到了原文中计算距离，即垂直距离、平行距离以及角度距离的时候是先将经纬度坐标投影到平面直角坐标系中进行计算，这样的计算方式无法避免会带来一些计算误差。而本项目用的是 haversine 公式进行距离和角度的计算，这是基于经纬度的距离计算方式，计算结

果会更加精确。

5 实验结果分析

算法的可视化分析。实验轨迹路径可视化结果如图 2所示。可以看到，使用 KMeans 轨迹聚类得到的结果，轨迹都聚集在一起，并且得到的结果轨迹没有落在路网上，因此并不具备代表性路径的特征。最大权重算法能得到一条比较长的代表性轨迹，但是另外的一条路径却比较短，从图中可以发现，最大权重算法有的路径覆盖范围并不广，因此其覆盖轨迹不一定是最多的路径。而覆盖优先算法得到的代表性路径比较多，而且大多数路径长度比较短，覆盖轨迹的范围也会比较广。连接优先算法结果则比最大权重算法和覆盖优先算法稍微好一些，有较长的代表性路径，其结果范围也比较广，但不可避免地，会出现搜索到最后，路径长度变得比较短，这与数据集的质量以及 k 的设置过大有一定的关系。

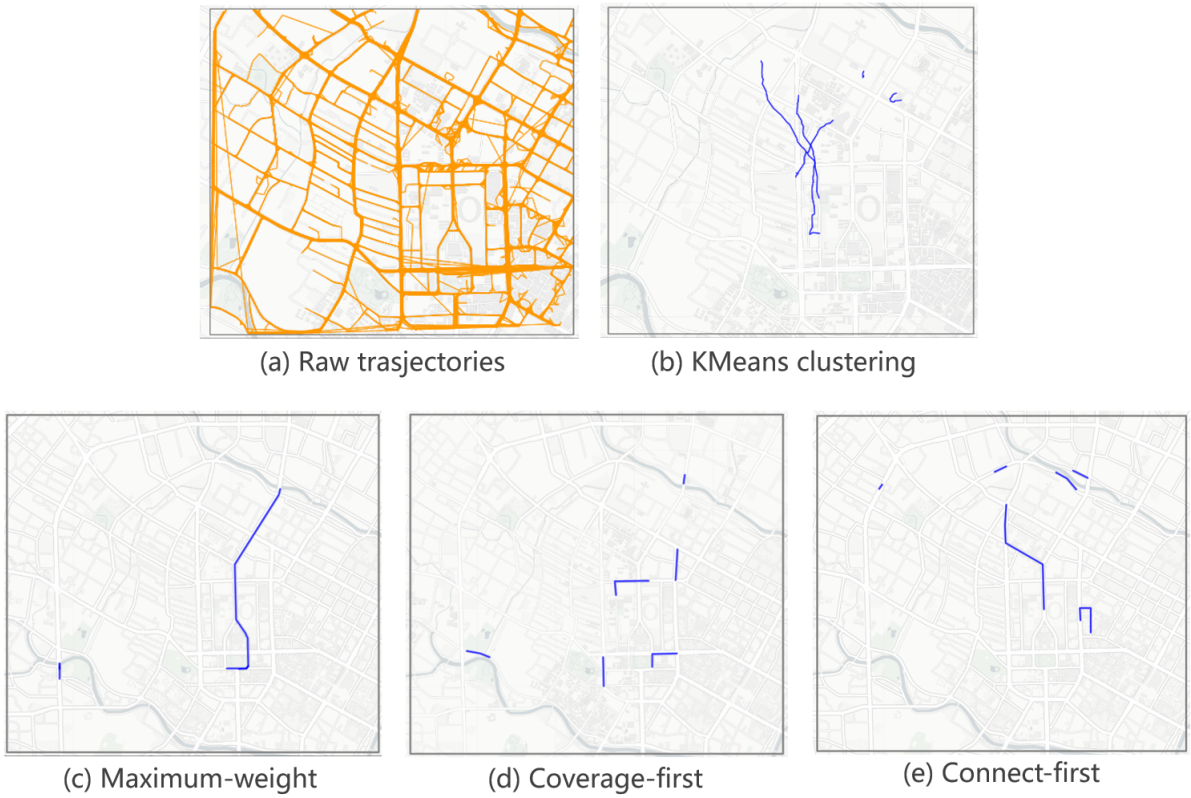

图 2: 轨迹和路径可视化结果，(a) 是进行实验的一部分轨迹，(b) 是用 KMeans 轨迹聚类得到的结果，(c) 是最大权重算法计算得到的代表性路径，(d) 是覆盖优先算法计算得到的代表性路径，(e) 是连接优先算法计算得到的代表性路径

算法的性能分析。由图 3可以发现，在距离阈值的比例 e_ratio 和路径代价比例 b_ratio 以及需要搜索的代表性路径数量 k 的变化下，连接优先算法的效率始终优于覆盖优先算法和最大权重算法，而最大权重算法次之。这是优于连接优先算法在每次搜索的时候，主要关注与收尾路径的路口点相连的路段情况，然后进行路径的扩增，而最大权重算法根据邻接矩阵进行搜索，覆盖优先则每次都要查找最大覆盖轨迹数的路段，因而运行时间吗会比较长。图 4描述了三个算法的性能，即前面定义的代表性的分变化情况。可以看到，e_ratio 和 b_ratio 的改变对结果的影响不大，连接优先算法的代表性得分均高于其他两个。而当 k 变大时，覆盖优先算法得分会持续增大，这是因为覆盖优先算法是找到剩余没有添加到路径中的路段里面覆盖轨迹数量最多的路段到结果集中，不考虑其连接性等因素，因此可以随着寻找的路径数增大而一直增大，而连接优先和最大权重算法则会拘束于其连接性。

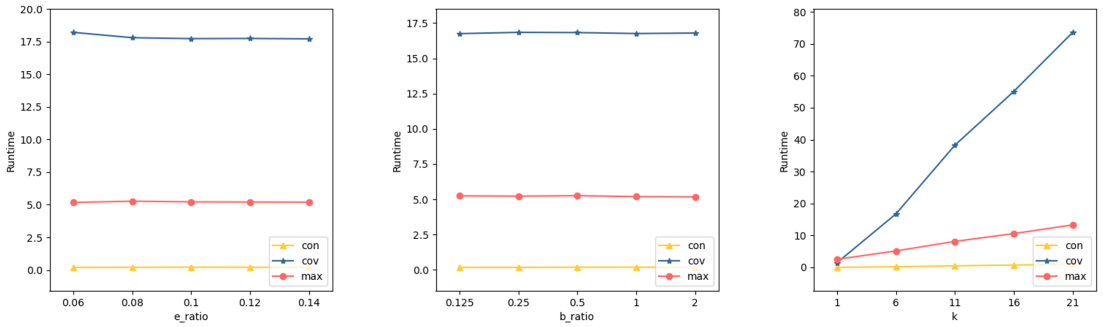

图 3: 三种算法运行时间对比

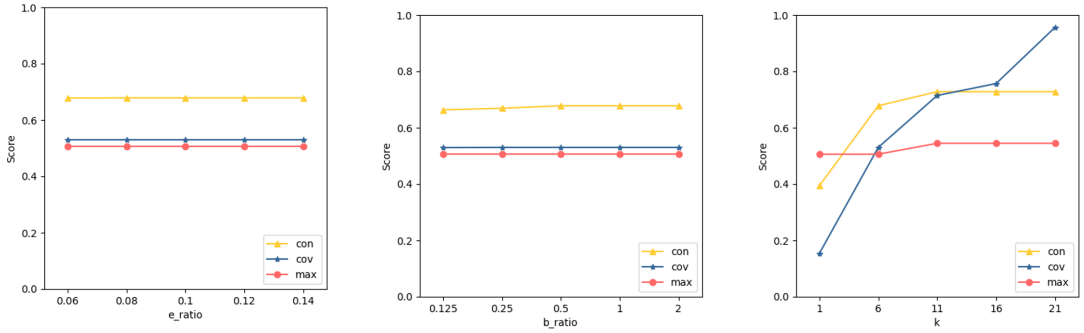

图 4: 三种算法代表性得分对比

6 总结与展望

本文设计了 Maximum-weight、Coverage-first、Connect-first 三个算法，用以从大规模的轨迹数据中获取 k 条代表性路径。与传统的轨迹聚类方法相比，设计的三种算法能找到非聚集的最具代表性的路径，而且省了地图匹配这一过程，算法时间效率远超轨迹聚类方法。原作只开源了 C++ 实现的 Connect-first 算法，本项目复现工作用 Python 复现了文章中提出的三个算法，与原作将坐标投影到平面坐标相比，本项目在 WGS84 坐标系下进行更精确的距离计算，并且在新的数据集成都出租车轨迹数据上进行实验，从结果和效率方面都验证了三种算法的性能。

本文所实现的三个算法均能找到基于路网生成的若干路径，但是后搜索到的路径往往比较短，考虑到有的轨迹比较长，只要轨迹中的一个小线段被路段覆盖，则视为轨迹被路段覆盖，因此，对于长轨迹很容易被多条轨迹覆盖，因此，如果原始轨迹都比较长，由于增益的连续边变少甚至没有，因此找到的代表性路径也会比较短。在未来改进的时候，可以对长轨迹限定范围进行分割，让路段获取更好的轨迹覆盖关系，用以完善现有三个算法的不足。

参考文献

[1] LI M, TONG P, LI M, et al. Traffic Flow Prediction with Vehicle Trajectories[C]//. 2022.

[2] LIN F, HSIEH H P, FANG J Y. A Route-Affecting Region Based Approach for Feature Extraction in Transportation Route Planning[J]. european conference on machine learning, 2020.

[3] WANG S, SUN Y, MUSCO C, et al. Public Transport Planning: When Transit Network Connectivity Meets Commuting Demand[J]. international conference on management of data, 2021.

[4] ZHU F, LV Y, CHEN Y, et al. Parallel Transportation Systems: Toward IoT-Enabled Smart Urban Traffic Control and Management[J]. IEEE Transactions on Intelligent Transportation Systems, 2020.

[5] LEE J G, HAN J, WHANG K Y. Trajectory clustering: a partition-and-group framework[J]. international conference on management of data, 2007.

[6] WANG S, BAO Z, CULPEPPER J S, et al. Fast large-scale trajectory clustering[J]. very large data bases, 2019.

[7] FAROOQ A, XIE M, STOILOVA S, et al. Transportation Planning through GIS and Multicriteria Analysis: Case Study of Beijing and XiongAn[J]. Journal of Advanced Transportation, 2018.

[8] MEES P, STONE J, IMRAN M, et al. Public Transport Network Planning: A Guide to Best Practice in NZ Cities[J]. New Zealand Transport Agency Research Report, 2010.

[9] BRANKOVIC M, BUCHIN K, KLAREN K, et al. (k, l)-Medians Clustering of Trajectories Using Continuous Dynamic Time Warping[J]. advances in geographic information systems, 2020.

[10] BUCHIN K, DRIEMEL A, van de L'ISLE N, et al. klcluster: Center-based Clustering of Trajectories [J]. advances in geographic information systems, 2019.

[11] HUNG C C, PENG W C, LEE W C. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes[J]. very large data bases, 2015.

[12] AGARWAL P K, FOX K, MUNAGALA K, et al. Subtrajectory Clustering: Models and Algorithms[J]. symposium on principles of database systems, 2018.

[13] Da SILVA T L C, LETTICH F, de MACÊDO J A F, et al. Online Clustering of Trajectories in Road Networks[J]. mobile data management, 2020.

[14] Da SILVA T L C, ZEITOUNI K, de MACÊDO J A F. Online Clustering of Trajectory Data Stream[J]. mobile data management, 2016.

[15] PANAGIOTAKIS C, PELEKIS N, KOPANAKIS I, et al. Segmentation and Sampling of Moving Object Trajectories Based on Representativeness[J]. IEEE Transactions on Knowledge and Data Engineering, 2012.

[16] PELEKIS N, TAMPAKIS P, VODAS M, et al. In-DBMS Sampling-based Sub-trajectory Clustering[J]. extending database technology, 2017.

[17] GIANNOTTI F, NANNI M, PINELLI F, et al. Trajectory pattern mining[J]. knowledge discovery and data mining, 2007.

[18] GONZALEZ H, HAN J, LI X, et al. Adaptive fastest path computation on a road network: a traffic mining approach[J]. very large data bases, 2007.

[19] LUO W, TAN H, CHEN L, et al. Finding time period-based most frequent path in big trajectory data[J]. international conference on management of data, 2013.

[20] ZHENG Y, ZHANG L, XIE X, et al. Mining interesting locations and travel sequences from GPS trajectories[J]. the web conference, 2009.

[21] CHEN Z, SHEN H T, ZHOU X. Discovering popular routes from trajectories[J]. international conference on data engineering, 2011.

[22] GOMES G A M, SANTOS E, VIDAL C A, et al. Real-time discovery of hot routes on trajectory data streams using interactive visualization based on GPU[J]. Computers & Graphics, 2018.

[23] SACHARIDIS D, PATROUMPAS K, TERROVITIS M, et al. On-line discovery of hot motion paths[J]. extending database technology, 2008.

[24] ZHENG L, FENG Q, LIU W, et al. Discovering Trip Hot Routes Using Large Scale Taxi Trajectory Data[J]. advanced data mining and applications, 2016.

[25] YI B K, JAGADISH H V, FALOUTSOS C. Efficient retrieval of similar time sequences under time warping[J]. international conference on data engineering, 1998.

[26] NUTANONG S, JACOX E H, SAMET H. An incremental Hausdorff distance calculation algorithm[J]. very large data bases, 2011.

[27] AGARWAL P K, AVRAHAM R B, KAPLAN H, et al. Computing the discrete Fréchet distance in subquadratic time[J]. SIAM Journal on Computing, 2013.

[28] CHEN L, NG R T. On the marriage of Lp-norms and edit distance[J]. very large data bases, 2004.

[29] CHEN L, ÖZSU M T, ORIA V. Robust and fast similarity search for moving object trajectories[J]. international conference on management of data, 2005.

[30] VLACHOS M, KOLLIOS G, GUNOPULOS D. Discovering similar multidimensional trajectories[J]. international conference on data engineering, 2002.

[31] LONG C, WONG R C W, JAGADISH H V. Direction-preserving trajectory simplification[J]. very large data bases, 2013.

[32] ZHANG D, DING M, YANG D, et al. Trajectory simplification: an experimental study and quality analysis[C]//. 2018.

[33] CHEN J, LEUNG M K H, GAO Y. Noisy logo recognition using line segment Hausdorff distance[J]. Pattern Recognition, 2003.