中国科学技术大学

学士学位论文

大规模轨迹数据的高效聚类算法设计与实现

作者姓名： 吴极端

学科专业： 信息与计算科学

导师姓名： 韩恺教授

完成时间： 二〇二〇年五月二十一日

University of Science and Technology of China

A dissertation for bachelor's degree

Design and Implementation of Efficient Clustering Algorithm for Large Scale Trajectory Data

Author: Jiduan Wu

Speciality: Information and Computational Science

Supervisor: Prof. Kai Han

Finished time: May 21, 2020

致　谢

感谢我的毕设导师韩恺教授予我的悉心指导和及时反馈，使我在此次毕设过程中收获良多。感谢我的毕设的主要参考文献第一作者王盛博士就他的论文和我讨论与不吝赐教。

感谢我的家人对我一直以来的支持，特别是我的母亲，这段日子以来每天都会询问我毕设完成否，让我这一段时间都活在毕不了业的恐惧中，更有动力完成这篇论文；还有糊糊，撸它的毛极大地缓解了我的焦虑。

还要感谢我的朋友曹颖，曹戈，李京琪，吴子晨，余颖雪同学为我答疑解惑，有效地推进了我的毕设进度。感谢我的舍友和王乔同学，她们有意无意的询问对我的毕设起到了很好的督促作用。

感谢科大四年来的培养，感谢科大老师的教诲，他们是我一生的榜样。

感谢在疫情中帮助过武汉的人们，让被封在城中的我能衣食无忧，平安健康地度过这段艰难的时期。

目 录

中文内容摘要

本文主要研究了城市交通背景下的大规模轨迹数据聚类问题，k-paths，文章致力于实现从众多出租车轨迹数据中提取出具有代表性的 k 条路径的高效算法。k-paths 聚类算法在交通管控，站点设置等城市规划情景中有着广泛应用。

首先，本文重点重现了一个新颖的高效聚类算法，此算法应用了基于边的距离度量 (Edge-Based Distance, EBD) 来计算轨迹之间的距离，并且不需要使用者提供除了 k 以外的任何参数。EBD 度量计算的简易迅捷以及反向索引的应用，利用 EBD 度量满足三角不等式而进行距离计算次数的削减让百万级别的轨迹数据的聚类平均能在一两百秒内完成；而提取代表路径的贪心算法，利用宽度搜索从边开始延拓成路径，能使得目标函数进一步得到优化，从而得到了更具代表性的的图心路径。

在原算法的基础上，本文对其进行了以下方面的改进：一、使用并行化技巧提高了算法的运行效率。二、抛弃了原算法中用于简化问题的假设：道路网络中所有的边长度均为 1。力图得到基于边真实长度的相应算法来使实验更加合理。而解决这个问题的难点在于，如何设计出能够提取出能进一步优化目标函数的代表路径的算法，即针对图心路径提取问题 (Centroid Path Extraction Problem, CPEP) 的算法。基于原算法的简单的类推：计算待选路径未来延拓的目标值下界并不能得出收敛的算法。为了解决这个问题，本文创新地提出了加权平均频数 (Weighted Average Frequency, WAF) 来发掘 “最受欢迎” 的路段，以此为基础得到了新的收敛算法，并用实验证明了算法的可行性与高效性。

关键词：k-paths 问题，聚类算法，大规模轨迹数据，EBD 度量，图心路经提取问题，加权平均频数 (WAF)

Abstract

In this paper, we study the large-scale trajectory clustering problem, k-paths, in the setting of urban traffic. We aim to extract k "representative" paths from the whole data set. The k-paths clustering algorithms have a wide range of applications in traffic monitoring, site selection and many other situations in urban planning.

First of all, we managed to reproduce a novel clustering algorithm, which adopts a novel measure, Edge-based Distance (EBD), and the only hyperparameter we need to apply this algorithm is k. The high efficiency of computing the EBD measure, the adoption of inverted index and the distance computation pruning with EBD satisfying the triangle inequality make it possible to extract k centroid paths in less than a few hundreds of seconds. And the greedy algorithm extracts better centroid paths by optimizing the objective function further.

To improve the original algorithm, the parallelization technique is firstly used to improve it in terms of efficiency. To make experiments more reasonable, we abandoned the assumption in the original algorithm simplifying the model that the lengths of all edges in the road network are equal. The biggest difficulty lay in designing an scalable algorithm for centroid path extraction problem(CPEP) to optimize the objective value further, and the simple analogy of the original algorithm, which adopts the the lower bound of the "future objective value" doesn't converge. To conquer this challenge, we innovatively introduced the idea of weighted average frequency(WAF) to extract the most popular paths and obtained an effective algorithm.

Key Words: k-paths; Clustering Algorithm; Large-Scale Trajectory Data; Edge-based Distance Measure; Centroid Path Extraction Problem (CPEP); Weighted Average Frequency (WAF)

符 号 说 明

p 位置点

e 边

T 轨迹

$path$ 路径

D 总数据集

S_i 聚类 i

k k-paths 问题中的参数 k

S 所有聚类的集合

$P(s_i \rightarrow s_j)$ 从状态 s_i 转移到 s_j 的概率

$P(o_j|s_i)$ 由状态 s_i 观察到观测值 o_j 的概率

LH 长度直方图

EH 边直方图

ALH 累计长度直方图

WAF 加权平均频数

$CPEP$ 图心路径提取问题

S_j 聚类 S_j

$\|G_j\|$ 聚类 S_j 中所有轨迹长度之和

OV_j 聚类 S_j 的目标值

CM 图心路径的距离矩阵, 大小为 $k \times k$

CD_i 修正过程后, 聚类新旧图心路径间的距离

be 路径的起边

ee 路径的终边

$LB(ps)$ 路径 ps 的 “潜质”

第一章　简介

位置获取和移动计算等领域的技术进步产生了海量的空间轨迹数据，例如日本日立公司提供的网站上与之连接的汽车每小时就会上传多达 25GB 的行车数据到云端 [1]。轨迹数据来源众多，比如车载 BDS、GPS 设备、旅行日志、运动分析、登记入住、信用卡消费、移动电话信号、动物的迁移、龙卷风、海啸的移动都会产生铺天盖地的轨迹数据。一般的原始轨迹数据由一串地理位置点组成，点则一般由地理坐标和相应时间戳表示 p=(x,y,t)。针对这些轨迹数据，主要的研究领域有轨迹数据处理 (Trajectory Preprocessing)，轨迹数据索引与检索 (Trajectory Indexing and Retrieval)，轨迹数据不确定性 (隐私保护以及减少不确定度)，轨迹模式挖掘 (Trajectory Pattern Mining)，轨迹数据分类 (Trajectory Classification), 轨迹异常检测等 (Trajectory Anomaly Detection)[2]。而本文则重点研究轨迹模式挖掘中的聚类问题。

一般的聚类算法分为两种：基于划分的 (Partition-based) 聚类方法以及基于密度的 (Density-based) 聚类方法。一些经典的轨迹聚类算法，比如基于划分的 TRACLUS 算法 [3] 和基于密度的 DBSCAN 算法 [4] 等。它们往往存在的问题是比较适用于小规模轨迹，比如 DBSCAN 算法最初的提出是为了研究自然界龙卷风轨迹，龙卷风的轨迹规模可能只有上百条，甚至不能与几百辆出租车一天之中产生的轨迹数据规模相提并论。并且这些算法都需要提供一些依赖于数据集的超参数与阈值，而它们往往难以确定，因此这些传统算法对大规模轨迹数据的处理力不从心。另外一个重要区别是 DBSCAN 方法直接使用了原始 GPS 数据值，这样的处理方法针对龙卷风移动没有特定路线的特点而言是合情合理的，而对于城市交通中的轨迹聚类问题，车辆只能在既定的道路上行驶，这一特性使得车辆轨迹数据需要更加具有针对性的数据预处理方式。最近的研究表明将原始轨迹数据匹配到已有的地图模型上对于存储和索引是更加有效的研究方法 [5]。经过地图匹配处理后，轨迹就能用一串组成轨迹的边来表示，每条边均有唯一的序号。基于这些原因，在这篇论文中，我们使用经过地图匹配后的轨迹数据来进行下一步的研究。

k-paths 聚类问题，是指将所有轨迹分成 k 个聚类，并在每个聚类中提取一条属于现有道路网络模型的图心路径作为 “代表”。k-paths 问题在城市交通状况分析，公共交通工具站点选择等情境下均有广泛应用，而高效解决大规模车辆轨

迹的 k-paths 问题将在这些情景中提供有价值的建议与解决方案。

本文中，我们首先复现了一个新颖的高效聚类算法来研究城市交通背景下的 k-paths 问题，并将其进行改进，致力于得到更加合理的高效聚类算法。

k-paths 问题作为机器学习经典算法的直接变体，本文中实现的原算法采用了 Lloyd 算法作为基本算法框架，为了提高分配与修正过程的效率，使算法能够高效运行，原算法主要提出了以下几点创新：

- 提出了新颖的 EBD 度量用于计算轨迹之间的距离，此度量方式计算效率远高于一般度量方式。
- 简化聚类目标值计算公式，并建立直方图来记录相关数据。
- 建立了边 (路段) 与轨迹之间的反向索引，在距离计算前先进行相交判断，进一步简化了无交轨迹之间的距离计算。
- 利用 EBD 度量满足三角不等式的性质，记录了轨迹到各个图心路经的距离上下界，先利用上下界辅助轨迹 “归类”，从而达到了减少距离计算次数的目的。
- 提出了针对图心路径提取问题 (CPEP) 的贪心算法，不将图心路径限定为已有轨迹，而是由道路网络中相连边组成的路径，并利用宽度搜索从边一步步延拓为路径。在以现有轨迹为图心路径的修正算法的基础上进一步优化目标函数，得到更具代表性的 k 条路径。

本文在原算法的基础上进行了以下改进与创新：

- 提出原简化的聚类目标值公式使用的范围：轨迹中应尽量不存在重复经过的路段。
- 将原分配与修正算法并行化，提高了运行效率。
- 不再使用原算法中用来简化问题的假设：认为所有边均为相同长度。重点改进了针对 CPEP 的修正算法，主要工作包括：
  - 设计新的累计长度直方图的构建算法，提升修正过程计算聚类目标值效率。
  - 引入新的衡量待选路径的量：加权平均频数 (WAF)，来提取更受欢迎的，更具代表性的图心路径。

第二章　问题定义与预备知识

第一节　轨迹数据模型

本文所有讨论均在下面模型中进行。

定义 2.1　(位置点) 一个位置点 p 由它的经纬度表示，$p = \{lat, lng\}$，其中 $lat = latitude$，$lng = longitude$。

定义 2.2　(边) 一条边 e 由一串位置点表示，也可以认为是一条路段，$e = \{p_1, p_2, ..., p_m\}$，本文中的边均是有向边。

定义 2.3　(原始轨迹数据) 原始轨迹数据由一连串位置点组成，$T = \{p_1, p_2, ..., p_n\}$。

定义 2.4　(道路网络) 道路网络是一个有向图 $G = (V, E)$，其中 V 是包含着路段的交点和终点的点集，E 则是代表路段的边的集合，每一个点与每一条边都有着用于识别的唯一序号。

定义 2.5　(路径) 路径是道路网络中一系列相连边的集合，$path = e_1 \to e_2 \to ... \to e_s$。

定义 2.6　(地图匹配后轨迹) 每一条原始轨迹经过数据预处理之后都能用一系列边来表示，$T = e_1 \to e_2 \to ... \to e_t$，显然每一条匹配后轨迹都是一条路径。

在后文中讨论的轨迹均指地图匹配后的轨迹，使用边序号的集合来表示一条轨迹，例如 $T = \{1, 5, 7\}$。一条边属于某一条轨迹，即这条边是这条轨迹的一部分。

第二节　k-paths 轨迹聚类问题

本文的主要研究对象 k-paths 问题严格定义见定义2.7。

定义 2.7　k-paths 轨迹聚类问题 将轨迹数据集 $D = \{T_1, T_2, ..., T_n\}$ 分成 k 个聚类 $S = \{S_1, S_2, ..., S_k\}, (k \leqslant n)$, 并在每个聚类中选出一条图心路径, 使得下面的目标函数值最小.

$$\text{Objective Value} = \sum_{j=1}^{k} \sum_{T_i \in S_j} Dist(T_i, \mu_j) \tag{2.1}$$

$$\{\mu_1, \mu_2, ..., \mu_k\} = \underset{\mu_1 \in S_1, \mu_2 \in S_2, ..., \mu_k \in S_k}{arg\,min} \sum_{j=1}^{k} \sum_{T_i \in S_j} Dist(T_i, \mu_j) \tag{2.2}$$

其中 Dist 是定义两条轨迹间的距离的度量，图心路径不一定是已有轨迹，但是一定属于道路网络图 G。

k-paths 问题作为机器学习中 k-means 问题的直接变体，针对 k-means 问题的 Lloyd 算法框架对于 k-paths 问题也同样适用。

Lloyd 算法主要包括以下三步：

1. 初始化：选定 k 条初始图心路径 (从已有轨迹中选取)。
2. 分配：将每条轨迹分配到距离最近的图心路径所代表的聚类中。
3. 修正：在每个聚类中重新提取图心路径使得聚类的目标值最小。

$$\text{Objective Value of Cluster j} = \sum_{T_i \in S_j} Dist(T_i, \mu_j) \tag{2.3}$$

算法 2.1 Lloyd 算法框架

Data: 所有轨迹数据 D，以及相应生成的轨迹道路网络图 G

Output: k 条图心路径 $\mu = \{\mu_1, \mu_2, ..., \mu_k\}$

```
initialization();
Iteration = 1;
while 上次修正过程中存在聚类的图心路经发生了改变或者 Iteration = 1 do
    Assignment() ;
    foreach Cluster_i ∈ Clusters do
        Refinement (Cluster_i) ;
    end
end
```

第三节　轨迹间距离计算: EBD 度量

原算法高效的一大原因就是提出了新颖的 EBD 度量，现有的衡量轨迹到轨迹之间距离的度量，一般分为两种：基于边 (edge-based) 的度量以及基于点 (point-based) 的度量，Edit Distance on Real Sequences (EDR) 度量即是一种基于点的度量，因为 EBD 度量与 EDR 测度有着类似的思想，在这里对 EDR 测度进行进一步说明，假设轨迹 S 由一系列向量元素 $\vec{s_i}$(位置坐标) 以及对应时间戳 t_i 表示，$S = [(t_1, \vec{s_1}), ..., (t_n, \vec{s_n})]$，$\vec{s_i} = (s_{i,x}, s_{i,y})$。

定义 2.8 (向量元素匹配) 轨迹 R 中的向量元素 $\vec{r_i}$ 和轨迹 S 中的向量元素 $\vec{s_j}$ 相匹配，即 $match(\vec{r_i}, \vec{s_j}) = true$，是指 $|r_{i,x} - s_{j,x}| \leqslant \epsilon$ 并且 $|r_{i,y} - s_{j,y}| \leqslant \epsilon$，其中 ϵ 是人为设定的阈值。

定义 2.9 (EDR 度量) 轨迹 R, S 之间在 EDR 度量下的距离, 是将轨迹 R 变成轨迹 S 所需进行的插入, 删除或替换操作的次数, 其严格定义如下.

$$EDR(R,S) = \begin{cases} n, & 若 m = 0, \\ m, & 若 n = 0, \\ min\{EDR(Rest(R), Rest(S)) + subcost, \\ \quad EDR(Rest(R), S) + 1, EDR(R, Rest(S)) + 1\}, & 其他情况 \end{cases} \tag{2.4}$$

其中如果 $\vec{r_1}$ 与 $\vec{s_1}$ 匹配，subcost = 0；如果 $\vec{r_1}$ 与 $\vec{s_1}$ 不匹配，subcost = 1。$R = [(t_1, \vec{r_1}), ..., (t_n, \vec{r_n})]$，$S = [(t_1, \vec{s_1}), ..., (t_m, \vec{s_m})]$，$Rest(S)$ 指除去轨迹 S 的第一个元素，即 $Rest(S) = [(t_2, \vec{s_2}), ..., (t_n, s_n)]$.

而 EBD 相当于基于点的 EDR 度量在基于边度量中的类推，将边的匹配定义为完全重合，套用 EDR 度量的定义，即可得到 EBD 度量定义式2.5。

$$EBD(T_1, T_2) = max(|T_1|, |T_2|) - |T_1 \cap T_2| \tag{2.5}$$

上面的类比是为了方便直观理解 EBD 度量，更加严谨的类推证明见 [6]。

第三章　算法实现

第一节　数据预处理: 地图匹配

为了更加有效地进行后面的研究，对于城市交通轨迹问题，原始数据经常要进行图匹配预处理。地图匹配问题是指将一系列观测到的地理位置点 (比如 GPS 数据点) 与现有的地图模型中的位置点对应起来，达到将原始数据规范化的目的，以便进行后续地查询、索引和存储操作。常用的城市街道地图有 Open Street Map[7]。本文中使用的道路网络图 G(图3.1) 是通过 GraphHopper API[8] 完成的地图匹配预处理，它背后主要的算法原理是 Viterbi 算法。

图 3.1　Porto 数据集 (部分) 的道路网络

Viterbi 算法是一个动态规划算法，用于由一系列观测值推测出最可能的状态序列，在有关马尔可夫信息源和隐式马尔科夫模型的情境中应用广泛，具体算法模型由下例说明。

已知一系列观测值 $Y = (y_1, y_2, ..., y_T)$，想要得到最有可能的隐藏状态序列 $X = (x_1, x_2, ..., x_T)$，假设状态空间为 $S = \{s_1, s_2, ..., s_K\}$, 观测值空间 $O = \{o_1, o_2, ..., o_N\}$，初始状态的概率分布 $\Pi = \{\pi_1, \pi_2, ..., \pi_K\}$, 满足 $P(x_1 = s_i) = \pi_i$, 转移概率矩阵 (transition matrix) 由 $A_{K\times K}$ 记录，从状态 s_i 转移到状态 s_j 的概率 $P(s_i \to s_j) = A_{ij}$；观察概率矩阵 (emission matrix) 由 $B_{K\times N}$ 记录，由状态 s_i 观察到观测值 o_j 的概率 $P(o_j|s_i) = B_{ij}$

将 Viterbi 算法中的模型应用到本文研究的地图匹配问题中，状态空间就是道路网络中的路段，而观测值空间就是原始轨迹数据：GPS 位置点序列，见图3.2。观测概率矩阵记录的是在在噪音干扰下观测到已知观测点 z_t，实际此时车辆行驶路段为路段 r_i 的概率。而转移概率矩阵则记录的是在时间 t 到 t+1 之间，车辆

算法 3.1 Viterbi 算法: 算法中 $T_1[i,j]$ 存储的是满足 $x_j = s_i$ 的最大概率, 而 $T_2[i,j]$ 记录的则是使得 $x_j = s_i$ 概率最大的 x_{j-1} 的状态值.

Input: 观测矩阵 Y，状态空间 S，观测值空间 O，状态转移概率矩阵 A，观测概率矩阵 B，初始状态概率分布 Π

Output: X

for $i = 1{:}K$ **do**
  $T_1[i,1] = \pi_i \times B_{iy_1}$;
  $T_2[i,1] = 0$;
end
for $j = 2{:}T$ **do**
  for $i = 1{:}K$ **do**
    $T_1[i,j] = \max\limits_k(T_1[k,j-1] \times A_{ki} \times B_{iy_j})$;
    $T_2[i,j] = \operatorname{argmax}\limits_k(T_1[k,j-1] \times A_{ki} \times B_{iy_j})$
  end
end
$z_T = \operatorname{argmax}\limits_k(T_1[k,T])$;
$x_T = s_{z_T}$;
for $j = T{:}{-}1{:}2$ **do**
  $z_{j-1} = T_2[z_j, j]$;
  $x_{j-1} = s_{z_{j-1}}$
end

从路段 r_i 转移到 r_j 的概率 (具体定义方式见 [9])。注意，这里的转移概率矩阵不是像算法3.1中简化的那样为常矩阵，这里的概率转移矩阵会随时间推移，具体的观测点位置移动而导致的待选路段不同而变化。

经过了地图匹配之后，[5] 中的算法进一步通过区分地图模型中交点终点与非交点或终点的点完成道路网络的构建，本文中为了方便与参考文献中的结果进行对比，使用了 [5] 中匹配好的道路网络图数据进行实验。

第二节　初始化方法

本文中使用了两种初始化方法：随机初始化与 k-means++ 初始化方法进行实验，两种方法的主要区别在于，在 k-means++ 算法中，图心路径的选择会更加“分散”[10]，从而达到使得算法更快收敛，优化目标函数的目的。但是由算法3.2可知，此初始化方法计算量较大，时间复杂度为 $O(k|D|) \times O(dist)$，在随机初始化

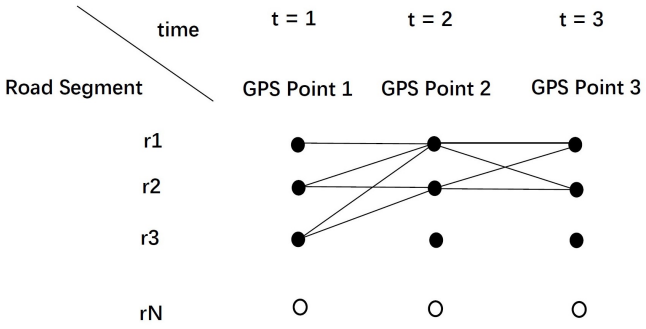


图 3.2　Viterbi 算法应用到地图匹配问题中

结果已经比较令人满意的情况下, 作为一个初始化方法 “性价比” 不高。所以在后续实验中，本文只使用一万条以下规模数据进行实验，探讨初始化方法对算法结果的影响。

算法 3.2　k-means++ 初始化算法

Input: 所有轨迹数据 D，参数 k

Output: k 条初始图心路径 $\mu = \{\mu_1, ..., \mu_k\}$

1 从 D 中随机选取第一条图心路径 μ_1;

2 **while** # 图心轨迹 $< k$ **do**

3 　以 $P(\mu_i = T_l \in D) = \frac{D(T_l)^2}{\sum_{T_j \in D} D(T_j)^2}$ 选取下一条图心路径;

4 **end**

其中 $D(T_l)$ 表示轨迹 T_j 到距离最近的已有图心路径的距离。

第三节　算法效率提升重点: 分配与修正算法

朴素的分配算法3.3和修正算法3.4，在理解上很直观，但直接用来实现聚类算法计算效率极低，分配过程的时间复杂度是 $O(k|D|) \times O(dist)$；如果类比 k-medoids 算法，选取各个聚类中已有的轨迹作为该聚类的图心路径 (centroid path)，修正步骤中的复杂度修正过程时间复杂度是 $O(|D|^2) \times O(dist)$。由此可见, 野蛮计算的代价是昂贵的。从计算复杂度的表达式容易看出，可以从加快单次距离计算和减少距离计算次数等角度切入来提高算法效率。

第四节　基线算法的高效性提升

综合本节效率优化方法，可得基线算法3.7

算法 3.3 分配算法框架

Input: 所有轨迹集合 D 以及所有图心路径集合 $\mu = \{\mu_1, ..., \mu_k\}$

Output: 所有轨迹分配到的距离最近图心路径所代表聚类，聚类 $S = \{S_1, ..., S_k\}$

```
1  minDistance = ∞;
2  foreach T_i ∈ D do
3      foreach μ_j ∈ μ do
4          tempDistance = ComputeDistance (T_i, μ_j) ;
5          if tempDistance < minDistance then
6              minDistance = tempDistance ;
7              T_i 被归类到聚类 S_j ;
8          end
9      end
10 end
```

算法 3.4 修正算法框架 (以已有轨迹为图心路径)

Input: 聚类 S_j，以轨迹 T 为图心路经的聚类目标值记为 $OV_j(T)$

Output: 聚类 S_j 的图心路经 μ_j

```
1  minOV = ∞;
2  foreach T_i ∈ S_j do
3      OV_j(T_i) = 0;
4      foreach T_k ∈ S_j do
5          OV_j(T_i) += Dist(T_i, T_k);
6      end
7      if OV_j(T_i) < minOV then
8          μ_j = T_i;
9      end
10 end
```

一、修正过程增速：聚类目标值简化计算公式

将 EBD 计算公式带入聚类目标值公式，并一步步简化。

$$OV_j = \sum_{T \in S_j} Dist(T, \mu_j) \tag{3.1}$$

$$= \sum_{T \in S_j} EBD(T, \mu_j) \tag{3.2}$$

$$= \sum_{T \in S_j} max(|T|, |\mu_j|) - |T \cap \mu_j| \tag{3.3}$$

$$= \sum_{T \in S_j} max(|T|, |\mu_j|) - \sum_{e \in \mu_j} \|e\| \tag{3.4}$$

$$= \sum_{T \in S_j} |T| + \sum_{T \in S_j'} (|\mu_j| - |T|) - \sum_{e \in \mu_j} \|e\| \tag{3.5}$$

$$\tag{3.6}$$

其中 OV_j 代表聚类 S_j 的目标值，$\|e\|$ 等于边 e 在聚类 S_j 轨迹中出现次数与边 e 的长度的乘积，S_j' 是聚类 S_j 中长度小于图心路径长度的轨迹组成的集合。针对简化后的目标值计算公式，发现第一个求和式对固定聚类来说是常数，于是用图权重 $\|G_j\|$ 来记录，并建立三个直方图：边直方图 (EH_j)，长度直方图 (LH_j) 和累计长度直方图 (ALH_j) 来记录相关数据，达到简便计算的目的。

$\|G_j\|$ 等于聚类 S_j 中所有轨迹的长度和，长度直方图记录聚类中所有的轨迹长度值，$LH_j[l]$ 等于聚类 S_j 中长度为 l 的轨迹条数，边直方图记录聚类中组成轨迹的所有的边，$EH(e) = |e\|$。累计长度直方图则是为了记录当图心路径长度为某些值时，S_j' 中轨迹长度与图心路径长度之差的绝对值之和，具体定义式见公式3.7.

$$ALH_j[m] = \begin{cases} 0, & 1 \leqslant m \leqslant min_{T \in S_j} |T| \\ ALH_j[m-1] + \sum_{1 \leqslant l \leqslant m-1} LH_j[l], & m \leqslant max_{T \in S_j} |T| \end{cases} \tag{3.7}$$

引入了这些量之后, 目标值计算公式则可以写为3.8。

$$OV_j = \|G_j\| + ALH_j[|\mu_j|] - \sum_{e \in \mu_j} EH_j(e) \tag{3.8}$$

1. **补充说明**：聚类目标值简化计算公式的适用条件

当单条轨迹中大量出现重复的边时，简化的聚类目标值计算公式就会出现不可忽略的误差而不再适用，下面举一个简单的例子进行说明

聚类 $S_j = \{path_1, path_2\}$，$path_1 = \{e_1, e_2, e_3\}, path_2 = \{e_1, e_1, e_2\}$，$path_2$ 中重复边比例达到了边集合的三分之二，假设所有边长度均为 1。

无论选取哪条路径作为图心路径，由 EBD 度量的对称性，聚类 S_j 的目标值都应该等于 $EBD(path_1, path_2) = 3 - 2 = 1$。

而使用简化后的目标值计算公式时，设以 $path_1$ 为图心轨迹的聚类目标值为 OV_1，以 $path_2$ 为图心轨迹的聚类目标值为 OV_2，经过简单的计算可以得出 $\|G\| = |path_1| + |path_2| = 6$，$ALH_j[n] = 0$，$\forall n \in N$，$EH_j(e_1) = 3$，$EH_j(e_2) = 2$，$EH_j(e_3) = 1$。

$$OV_1 = \|G\| + ALH[3] - \sum_{i=1}^{3} EH(e_i) \tag{3.9}$$

$$= 6 + 0 - (1 + 2 + 3) \tag{3.10}$$

$$= 0 \neq 1 \tag{3.11}$$

$$OV_2 = 6 + 0 - (3 + 2) = 1 \tag{3.12}$$

所以如果使用简化公式3.8，聚类算法会选择 $path_1$ 作为图心路径，并且此时聚类目标值严重偏小。由此例子可知，当轨迹中重复边比例占到相当部分时，由简化后的公式求得的目标值会偏小，且误差不可忽视，具体达到多少比例会对算法结果产生不可忽视的影响，我们这里不作定量讨论，可供后续研究，此说明仅提醒使用者如果要使用式3.8，应尽量选用分割合理的数据集。本文实验中使用的轨迹数据重复边占比很小，认为误差可以忽略。

二、加快单次距离运算: 高效交集算法

因为 EBD 度量的公式2.5中存在求交集运算，故使用更高效率的交集算法也是提升整个算法效率的一个切入点。本文中测试了以下三种求交集算法：

1. 普通求交集算法 (算法3.5).
2. 双重二分查找算法 (算法3.6).
3. 利用多重集合的交集函数.

算法3.5与算法3.6均要求输入数组事先被有序排列 (同为升序或降序排列)，且均使用了一个简单的优化：先进行两个判断，当数组 A 的最小值大于数组 B 的最小值或者数组 A 的最大值小于数组 B 的最小值时，直接返回空集。

双重二分查找的基本思想是，在集合 A 中二分查找 B 的中间位置值，此处

算法 3.5 普通交集算法

Input: 经过排序的数组 A 和 B

Output: 数组 A 和 B 的交集 R

```
if minA > maxA 或者 minB > maxB then
    return ∅;
end
index1 = 0, index2 = 0, R = ∅;
while index1 < A.length && index2 < B.length do
    if A[index1]==B[index2] then
        R = R ∪ A[index1];
        index1 ++, index2 ++;
        continue;
    end
    if A[index1]<B[index2] then
        index1 ++;
    else
        index2 ++;
    end
end
```

调用的二分查找函数返回查询值的最紧上界元素的索引，利用分而治之的思想将问题分成两个子问题 [11]。如图3.3.

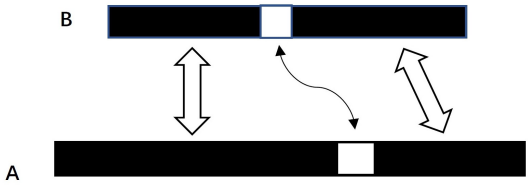


图 3.3 双重二分查找算法图示

第三种方法对输入数组无特殊要求，直接利用多重集合的交集函数 *Multi-sets.intersection(A，B)* 即可。

通过使用波尔图市的轨迹数据进行简单测试，当求前十万条轨迹间距离时，普通求交集算法和多重集合求交集算法效率相近，双重二分查找算法耗时接近于前二者的 1.5 倍，但是由于利用多重集合的交集不能简单的类推使用在考虑边的真实长度的情境下，故在后面的实验中, 均使用普通求交集算法。如果能够找到更好的求交集算法，则能进一步提升算法效率。

算法 3.6 DoubleBinarySearch(A，B，$minA$，$maxA$，$minB$，$maxB$s)

Input: 经过排序的数组 A 和 B，且 $A.length >= B.length$，A 查找范围：$minA$，$maxA$，B 查找范围：$minB$，$maxB$

Output: 数组 $A[minA : maxA]$ 和 $B[minB : maxB]$ 的交集 R

```
1  if minA > maxA 或者 minB > maxB then
2  |   return ∅;
3  end
4   midB = round((maxB + minB)/2);
5  midBValue = B[midB];
6  midA = BinarySearch(midBValue, A, minA, maxA);
7  if (midA - minA) > (midB - minB) then
8  |   R = R ∪ DoubleBinarySearc(A, B, minA, midA − 1, minB, midB − 1);
9  else
10 |   R = R ∪ DoubleBinarySearc(B, A, minB, midB − 1, minA, midA − 1);
11 end
12 if A[midA]==midBValue then
13 |   R = R ∪ midBValue
14 else
15 |   midA = midA - 1;
16 end
17 if (maxA-midA)>(maxB-midB) then
18 |   R = R ∪ DoubleBinarySearch(A, B, midA + 1, maxA, midB + 1, maxB);
19 else
20 |   R = R ∪ DoubleBinarySearch(B, A, midB + 1, maxB, midA + 1, maxA);
21 end
```

三、加快单次距离运算: 反向索引

一条轨迹对应着组成这条轨迹的边序号的集合，反过来，每一条边也都对应着穿过这条边的轨迹序号的集合，此为反向索引。通过事先建立反向索引可以判断两条轨迹是否有交集 (算法3.8)，如果没有交集，那么距离计算可以简化为一个求更大值运算 (式3.13)。在算法实现中，为了提高效率事先利用反向索引得到了每一条轨迹的相交轨迹集合。

$$EBD(T_1, T_2) = max(|T_1|, |T_2|) \qquad \text{如果 } T_1 \cap T_2 = \varnothing \tag{3.13}$$

四、减少距离计算次数: 削减距离计算

削减距离计算这一技巧建立在 EBD 度量满足三角不等式 (引理3.1) 的基础上。

引理 3.1 (EBD 度量满足三角不等式) 对任意轨迹 T_1, T_2 和 T_3, 下面的不等式均成立.

$$EBD(T_1, T_2) + EBD(T_2, T_3) \geqslant EBD(T_1, T_3) \tag{3.14}$$

$$|EBD(T_1, T_2) - EBD(T_2, T_3)| \leqslant EBD(T_1, T_3) \tag{3.15}$$

将三个集合用韦恩图表示，并利用 *max* 函数具有对称性，且满足不等式 $max(a,b) + max(c,d) \leqslant max(a+c, b+d)$，$a, b, c, d \geqslant 0$，即可得到证明，详细证明参考 [6]。

要利用三角不等式，对于每条轨迹 T_i，都要记录它到它所属聚类图心路径 $\mu_{a(i)}$ 的距离的上界 $ub(i)$，与到其他图心路径 $\mu_j, j \in \{1, 2, ..., k\}, j \neq a(i)$, 的距离下界 $lb(i, j)$，记其中的最小下界为 $lb(i)$。每次修正过程，每个聚类都记录了新旧图心路径之间的距离 (Centroid drift, CD)，利用三角不等式来更新每条轨迹的上下界，得到了更加 “宽松” 的上下界，用于下次分配过程，见图3.4。

$$ub(i) = ub(i)' + CD(a(i)') \tag{3.16}$$

$$lb(i, j) = |lb(i, j)' - CD(j)| \tag{3.17}$$

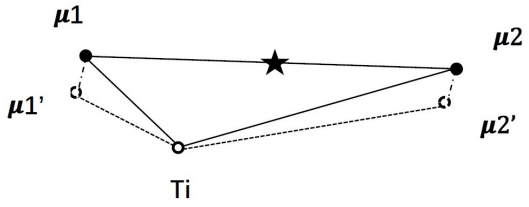


图 3.4 图心轨迹位移

注：五角星标注的是到图心轨迹 μ_1, μ_2 距离相等的位置

其中记号右上角标加了一撇均表示上一次迭代中对应的的旧值。利用这些上下界，提前进行 “归类” 预判的中心思想见式3.18，3.19。若3.18式成立，轨迹 T_i 到聚类 S_j 的距离一定大于到聚类 $S_{a(i)'}$ 的距离，那么轨迹 T_i 一定不会从原聚类

$S_{a(i)'}$ 移动到聚类 S_j 中；同理，若3.19式成立，轨迹不会被分配到聚类 $S_j, j \neq a(i)'$ 中，将仍被分配到原聚类中。

$$lb(i,j) \geqslant ub(i) \tag{3.18}$$

$$lb(i) \geqslant ub(i) \tag{3.19}$$

进一步，利用图心路径之间的距离矩阵 CM，$CM(i,j), j \neq i$ 等于聚类 S_i 的图心路径与聚类 S_j 的图心路径之间的距离，而 $CM(i,i) = \underset{j,j\neq i}{min} CM(i,j)$。如果3.20式成立，那么轨迹 T_i 一定不会从原聚类 $a(i)'$ 移动到聚类 j 中，原因见式3.22；同理，若3.21式成立，轨迹将仍被分配到原聚类中。升级后的不等式见3.26，3.27。

$$\frac{CM(a(i)', j)}{2} \geqslant ub(a(i)') \tag{3.20}$$

$$\frac{CM(a(i)', a(i)')}{2} \geqslant ub(a(i)') \tag{3.21}$$

$$EBD(T_i, \mu_j) \geqslant |EBD(\mu_{a(i)'}, \mu_j) - EBD(T_i, \mu_{a(i)'})| \tag{3.22}$$

$$\geqslant |EBD(\mu_{a(i)'}, \mu_j) - ub(a(i)')| \tag{3.23}$$

$$\geqslant \frac{CM(a(i)', j)}{2} \tag{3.24}$$

$$\geqslant ub(a(i)') \tag{3.25}$$

$$max\{\frac{CM(a(i)', a(i)')}{2}, lb(i)\} \geqslant ub(a(i)') \tag{3.26}$$

$$max\{\frac{CM(a(i)', j)}{2}, lb(i))\} \geqslant ub(a(i)') \tag{3.27}$$

第五节　基线算法有效性提升: 降低最终目标值

定义 3.1　(图心路径提取问题) 图心路径提取问题 (Centroid Path Extraction Problem, CPEP) 就是在道路网络 G 中找到一条路径使得到聚类 S_j 中的所有轨迹距离之和最小。

$$\mu_j = \arg\min_{path_i \in G} \sum_{T \in S_j} EBD(T, path_i) \tag{3.28}$$

其中 $|\mu_j| \in [l_{min}, l_{max}]$，即图心路径长度在聚类中最大最小轨迹长度之间。因为聚类中已有轨迹也是路径，所以理论上，解决 CPEP 问题能进一步优化目标函数。但是，由 [6] 中证明可知，CPEP 问题是 NP-hard 的，故只能利用贪心算法3.9得到近似解。

此贪心算法以聚类中的边为图心路径的 “种子”，利用广度优先搜索从邻边进行延拓，找到符合要求的图心路径，如图3.5所示。

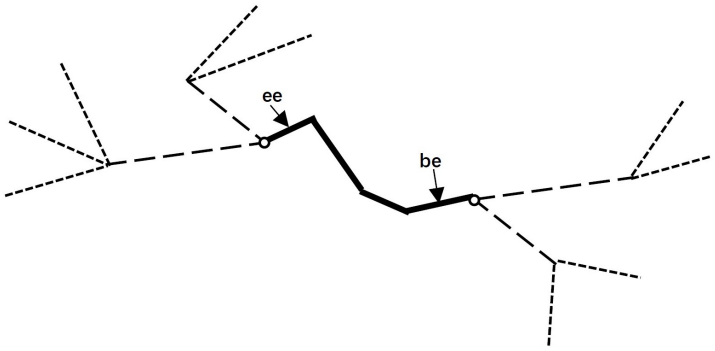


图 3.5　待选路径的延拓

注：be 代表路径的起边, ee 代表路径的终边

算法通过评估一条路径的 “潜质”，优先延拓 “潜质” 大的待选路径。广度优先搜索一般使用队列 (queue) 来实现，先进先出。为了提高搜索效率，算法中使用了优先队列 (Priority Queue, PQ) 来自主定义待选路径的出队列顺序。

定义 3.2　(CPEP 问题中路径的 “潜质”) 在聚类 S_j 中，一条路径的 “潜质”(threshhold potential)，记为 $LB_j(ps)$，是该路径未来延拓后的目标值下界，下界越低，路径越有可能被延拓为使得聚类目标值更低的理想图心路径.

记当前待选路径为 ps，以 ps 为图心路径，聚类 S_j 的目标值记为 $OV_j(ps)$。

$$OV_j(ps) = \|G_j\| - \sum_{e \in ps} EH_j(e) + ALH_j[l] \tag{3.29}$$

假设延拓后 $ps' = ps \cup e_{new}$。

$$OV_j(ps') = \|G_j\| - \sum_{e \in ps'} EH_j(e) + ALH[l'] \tag{3.30}$$

$$= \|G_j\| - \sum_{e \in ps} EH_j(e) + ALH[l] \tag{3.31}$$

$$- \underbrace{(EH(e_{new}) - ALH[l'] + ALH[l])}_{delta} \tag{3.32}$$

取 e_{new} 使得 *delta* 为尽可能大的正数，不断选取下一个满足这个条件的 e_{new} 直到 *delta* 变号，就可以得到未来延拓目标值的下界，也就是 $LB_j(ps)$。而对于有着相同起边和终边的路径，只保留有着更低下界的路径作为待选图心路径来加快延拓速度。此算法中 $LB_j(ps)$ 是至关重要的一个量，它决定了算法的收敛与否，与收敛速度。意识到一条路径的 “潜质” 可以有不同的定义方式为后文算法的改进埋下伏笔。

$$LB_j(ps) = \|G_j\| - \sum_{e \in ps} EH_j(e) + ALH_j[l] \tag{3.33}$$

$$- \sum_i (EH_j(e^i_{new}) - ALH_j[l'] + ALH_j[l]) \tag{3.34}$$

可以将求未来目标值下界的问题看成一个背包问题，剩余空间是 $\max_{T \in S_j} |T| - |ps|$，不断选取满足要求的物品 e_{new}，物品的价值为 $EH(e_{new}) - ALH[l'] + ALH[l]$，目标是使得背包所装物品总价值最大，也就是 $LB_J(ps)$ 最小。

算法 3.7 baseline algorithm

Input: 轨迹数据 D，k，函数 $ComputeEBD(\mu_j, T_i)$ 是利用了反向索引的 EBD 计算函数

Output: k 条图心路径，$\mu = \{\mu_1, \mu_2, ..., \mu_k\}$

```
1  Iteration = 1, Initialization(μ);
2  foreach T_i ∈ D do
3      minDist = ∞;
4      foreach μ_j ∈ μ do
5          tempDist = ComputeEBD(|T_i|, |μ_j|), lb(i,j) = tempDist;
6          if tempDist < minDist then
7              T_i.CentroidId = μ_j;
8          end
9      end
10 end
11 ConstructHistogram(S,D), Refinement(S,D);
12 while μ 发生改变并且 Iteration < maxIteration do
13     foreach S_i ∈ S do
14         foreach T_j ∈ S_i do
15             UpdateBounds(CD_i, T_j);
16             if max{CM(i,i)/2, lb(j)} ⩾ ub(i) then
17                 minDist = ComputeEBD(T_j, μ_i);
18                 foreach μ_k ∈ μ, k ≠ i do
19                     if max{CM(i,k)/2, lb(i))} ⩾ ub(i) then
20                         tempDist = ComputeEBD(T_j, μ_k);
21                         if tempDist < minDist then
22                             newCentroidId = k;
23                             minDist = tempDist, lb(j,k) = tempDist;
24                         end
25                     end
26                 end
27                 if newCentroidId neq i then
28                     T_j.CentroidId = newCentroidId;
29                     UpdateHistogram(S_i, S_newCentroidId, T_i);
30                 end
31             end
32         end
33     end
34     Refinement(S, D);
35 end
```

算法 3.8 运用反向索引判断两轨迹有无交集

Input: 道路网络 G，轨迹 $T_1, T_2 = e_1 \rightarrow e_2 \rightarrow ... \rightarrow e_m$，由 G 建立的反向索引 *InvertedInd*

Output: 若有交集返回 true, 否则返回 false

```
foreach e_i ∈ T_2 do
    if InvertedInd.get(e_i).contains(T_1) then
        return true;
    end
end
return false;
```

算法 3.9 CPEP 问题贪心算法

Input: 道路网络 G，聚类 S_j

Output: 图心路径 μ_j

```
PQ = ∅，Buffer = ∅，minOV = ∞，Iteration = 1;
foreach edge e∈ EH_j do
    PQ.push(e, LB(e));
end
while !PQ.IsEmpty() 并且 Itertion < MaxIterationInCPEP do
    (ps, LB(ps)) = PQ.poll();
    foreach neighbor edge e of ps do
        ps = ps∪e;
        if minOV(ps) < minOV then
            minOV = OV_j(ps), μ_j = ps;
            continue;
        end
        if Buffer.get(ps) > LB(ps) then
            Buffer.put(ps, LB(ps));
        end
        if LB(ps) < minOV then
            PQ.push(ps, LB(ps));
        end
    end
end
```

第四章　算法改进

第一节　原算法高效性

一、算法并行化

利用分配过程中每条轨迹的 “归类” 判断相互独立与每个聚类的修正过程相互独立，可以将程序并行化。见算法4.1,4.2.

算法 4.1　Baseline Algorithm(并行化)

Data: 所有轨迹数据 D，轨迹道路网络 G

Output: k 条图心路径 $\mu = \{\mu_1, ..., \mu_k\}$

```
initialization();
Iteration = 1;
while 上次修正过程中存在聚类的图心路径发生改变或者 Iteration = 1 do
    并行化 Assignment() ;
    foreach 并行化 Cluster_i ∈ Clusters do
        Refinement (Cluster_i) ;
    end
end
```

第二节　算法合理性

前述所有算法均在假定所有边长度均为 1 的前提下进行，而利用边的真实长度显然会使实验更加合理。

例如图4.1，黑色轨迹与绿色轨迹，黑色轨迹与红色轨迹同样是公共了一条边，显然黑色轨迹与红色轨迹 “相似度” 更高，理应 EBD 距离更小。

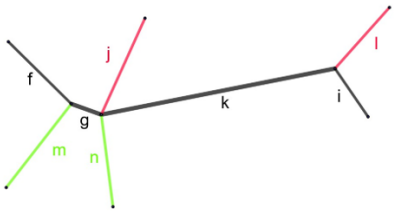


图 4.1　考虑边的真实长度

对于分配，修正 (以现有轨迹为图心路径) 算法的相应改变大同小异，但对

算法 4.2 分配算法 (并行化)

Input: 所有轨迹集合 D 以及所有图心路径集合 $\mu = \{\mu_1, ..., \mu_k\}$

Output: 所有轨迹分配到的聚类序号

```
1  foreach 并行化 S_i ∈ S do
2      foreach 并行化 T_j ∈ S_i do
3          minDistance = ComputeDistance (T_j, μ_i);
4          priorDistance = minDistance;
5          foreach μ_k ∈ μ, k ≠ i do
6              tempDistance = ComputeDistance (T_j, μ_k) ;
7              if tempDistance < minDistance then
8                  newClusterId = k;
9                  minDistance = tempDistance ;
10             end
11         end
12         if newClusterId≠i then
13             batchAddTrajectories.put(newClusterId, T_j);
14             batchRemoveTrajectories.put(i,T_j);
15             priorDists.put(T_j, priorDistance);
16             newDists.put(T_j,minDistance);
17         end
18     end
19 end
20 统一进行聚类目标值及轨迹成员, 直方图的更新;
```

于 CPEP 中的修正算法, 简单的类推并不能得到好的结果，甚至不能得到收敛的结果。所以本文的工作集中在对 CPEP 修正算法的改造.

一、合理化算法: CPEP 修正算法有效性

1. 简单类推

这里简单的类推是将指仍然试图得到待选路径未来延拓的目标值下界 LB，以路径延拓顺序的衡量标准。这仍可看作是一个背包问题，聚类中的边是物品，边的价值是边在轨迹中出现的次数，物品储量则是边的长度，背包的剩余空间则是 $\max_{T \in S_j} |T| - |ps|$。

下界容易理解，但是也忽视了待选路径之间的连接关系。在使用 LB 作为延拓顺序标准的算法中，一条假设将来会被延拓到路径上的边，可能正好和一条路

径相连，又可能和另一条路径相隔很远。也就是说并不是所有的背包可获取的物品，或者说获取物品的难度是相同的，那么这样衡量出来的潜质也就是没有代表意义的，这就体现在算法无法收敛的结果中。

2. 加权平均频数 (WAF) 的提出

在发现原方法的缺陷后，考虑到我们需要的图心路径是有代表性的，即多次出现的，而 “受欢迎” 的路段往往出现在同一片邻近区域。而且我们认为同样 “受欢迎” 的两条边，边的长度更大的那一条在路径中的地位更重要，因为它在求 EBD 距离中作用更大，于是将频数用边的长度加权，引入加权平均频数 (Weighted Average Frequency, WAF)，作为衡量待选路径出优先队列顺序的指标。并为了提高延拓效率，排除了具有相同起边与终边的待选路径中 WAF 最低的那条，具体算法见算法4.5。

$$WAF(ps) = \frac{\sum_{e \in ps} EH(e)}{|ps|} \tag{4.1}$$

二、合理化算法：CPEP 修正算法高效性

1. 高效的 WAF 更新

由下式已知 WAF 的更新计算时间复杂度为常数，极大地提升了运行效率。

$$WAF_{new} = \frac{ALH \times |ps| + EH(e_{new})}{|ps'|} \tag{4.2}$$

其中 $ps' = ps \cup e_{new}$

2. 累计长度直方图 (ALH) 的构建

但仅仅引入加权平均频数还不够，由于考虑边的真实长度使得轨迹长度是原来的几个数量级，原始的累计长度直方图构建方法不再能在可以接受的时间内输出结果，于是利用累计长度直方图的定义式设计出算法4.4。

分析两个算法时间，容易分析得到改进前的 ALH 构建算法时间复杂度为 $O((\max_{T \in S}|T| - \min_{T \in S}|T|)^2 + \max_{T \in S}|T| + \min_{T \in S}|T|)$ 而改进后的时间复杂度变为 $O(\max_{T \in S}|T|)$，改进后的算法极大地提升了程序运行效率，使得在合理时长内得到结果成为可能，也就得到了最后的考虑了边真实长度的 CPEP 修正算法4.5。

算法 4.3 简单类推得到的 refinement in CPEP 算法

Input: 聚类 S，待选图心路径 ps，记录聚类 S 中按降序排列的边频数数组 (*Frequency Sorted Array, FSA*)，以 ps 为 S 的图心路经得到的目标值 *OV(ps)*

Output: 图心路径 ps 未来延拓的目标值下界 *LB(ps)*

```
space = max_{T∈S} |T| − |ps|, LB(ps) = OV(ps);
for i = 0 : FSA.size() do
    item = FSA[i], ps' = ps ∪ {item};
    delta = EH(item) − ALH[|ps'|] + ALH[|ps|];
    volume = min{|item|, space} ;
    space = space-volume;
    if delta>0 or space <=0 then
        break;
    end
    OV = OV + delta, ps = ps';
end
```

算法 4.4 改进前后的 ALH 构建算法 (左边是改变前算法，右边是改变后算法)

Input: 聚类 S

Output: 针对聚类 S 的累计长度直方图 ALH

```
for i = 0 : min_{T∈S} |T| do
    ALH[i]=0;
end
for m = min_{T∈S} |T| + 1 : max_{T∈S} |T| do
    ALH[m] = ALH[m-1];
    for n = min_{T∈S} |T| : m − 1 do
        ALH[m]+=LH.count(n);
    end
end
```

Input: 聚类 S, 其中所有轨迹的长度按照升序记录在轨迹长度有序数组 (*Length Sorted Array*, *LSA*) 中

Output: 针对聚类 S 的累计长度直方图 (ALH)

```
for i = 0 : min_{T∈S} |T| do
    ALH[i]=0;
end
for i = 1 : LSA.size() do
    prior = LSA[i-1], present = LSA[i];
    for m = prior + 1 : present do
        ALH[m]=ALH[m-1]+i;
    end
end
```

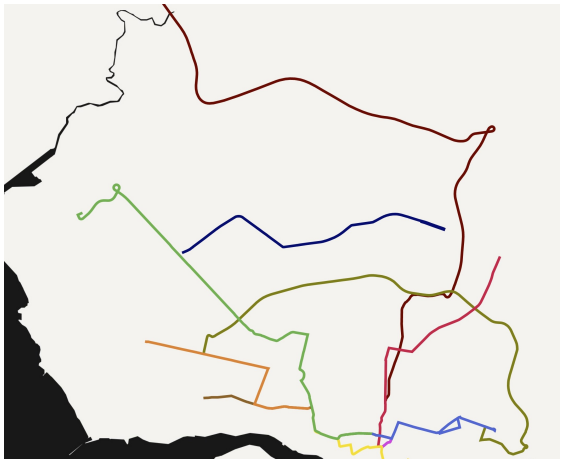
算法 4.5 My refinement in CPEP

```
Input: 道路网络 G
Output: min Objective Value (minOV)，图心路径 μ
1  PQ = ∅，Buffer = ∅，minOV = Double.MAX_VALUE，Iteration = 0;
2  foreach edge e ∈ EH do
3      ComputeAWF(e);
4      PQ.push(e, AWF(e));
5  end
6  while PQ.IsNotEmpty() and Iteration < MAX_ITERATION do
7      (ps, AWF(ps)) = PQ.poll();
8      foreach neighbor edge e of ps do
9          if EH.contains(e) and e∉ ps then
10             ps = ps ∪ e;
11             ComputeOV(ps);
12             ComputeAWF(ps);
13             if OV_ps < minOV then
14                 minOV = OV(ps), μ = ps;
15             end
16             construeToken(ps);
17             if Buffer.contains(Token(ps)) then
18                 if AWF(ps) < Buffer.get(Token(ps)) then
19                     AWF(ps) = Buffer.get(Token(ps));
20                 end
21             end
22             else
23                 Buffer.add(ps, AWF(ps));
24             end
25             PQ.add(ps, AWF(ps));
26         end
27     end
28     Iteration = Iteration + 1;
29 end
```

第五章　实验结果

图5.1是 k = 10 时，一次迭代得到的图心路径可视化。

图 5.1　一个图心路径可视化结果



我们通过实验来测试基于 EBD 度量的聚类算法的有效性和高效性，通过观测运行时间来验证利用反向索引、算法并行化对算法高效性的提升；通过记录被 “省” 去的距离计算次数与实际距离计算次数来验证利用三角不等式削减距离计算次数的作用；通过记录目标函数值来以及利用图搜索对算法有效性的提升。同时我们将验证使用了加权平均频数 (WAF) 的改良算法，以及其他更加合理的考虑了轨迹真实长度的算法的有效性以及高效性。在没有特别说明的情况下, 以下实验均是在考虑边的真实长度的前提下进行。

测试数据集. 本实验采用了波尔图市 2013 年 7 月 1 日到 2014 年 6 月 30 日共 442 辆出租车的轨迹数据，量级在百万级别。

程序实现环境. 所有实验均在使用 Intel(R) Core(TM) i7-9750H CPU 的 PC 上实现，可用内存为 15.7GB，使用 Eclipse 集成开发环境，Java 版本为 14.0.1，JVM 堆大小设置为 16GB。这里值得说明的是，虽然程序实际上记录的是运行时间而不是准确的 CPU 时间，但是我们只强调用时的量级，可以认为与 CPU 耗时在量级上相差不大，故作图时均用 CPU 耗时。并且实验时电脑内存使用环境尽量保持一致，可以用于不同方法的比较。

衡量指标. 本文主要通过运行时间以及目标函数值的大小衡量算法的高效性与有效性，具体的，根据各个过程的目的进行划分，直方图更新的时间被算入修正过程运行时间，而图心路径之间 $k \times k$ 的距离方阵计算时间则被算入分配时间。

用被削减的距离计算的次数与实际进行的计算次数来衡量利用三角不等式的加速技巧的有效性。

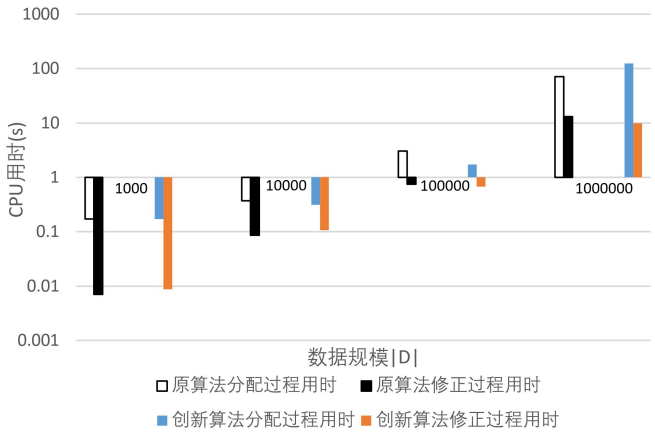
实现技巧. 边直方图和长度直方图都用多重集合 (Multiset) 来实现，并行化算法中利用 ConcurrentHashMap 记录移出以及移入的轨迹序号集合以及到新旧图心路径的距离，其中移入移出轨迹序号集合用 SynchronizedList 实现。

实验准备. 为了研究初始值对算法表现的影响，事先生成了两种初始化方法：随机初始化算法和 k-means++ 算法生成的 50 组初始值，每组初始值有 100 个，对于 k<50，就取前 k 个值作为初始值。由于 k-means++ 算法的运行时长与轨迹数据规模相关，50 万规模的数据集再现有设备上需要运行数十小时，只生成了以 1000，1000,0，1000,00 条轨迹作为总数据集的初始值。

测试参数. $|D| = \{100, 1000, 10,000, \underline{100,000}, 1,000,000\}$，$k = \{10, 20, \underline{30}, 40, 50\}$，随机初始化 与 k-means++ 初始化，使用普通修正算法与 CPEP 修正算法，使用并行化算法与非并行化算法，带下划线的参数是对比其他参数时该参数的默认值，下文中没有特别说明的情况下都是使用默认参数，为了避免个别难以收敛的情况，实验中将最大迭代次数设置为了 20 次，20 次还没有停止迭代时就取第二十次迭代后的结果作为最终迭代结果。

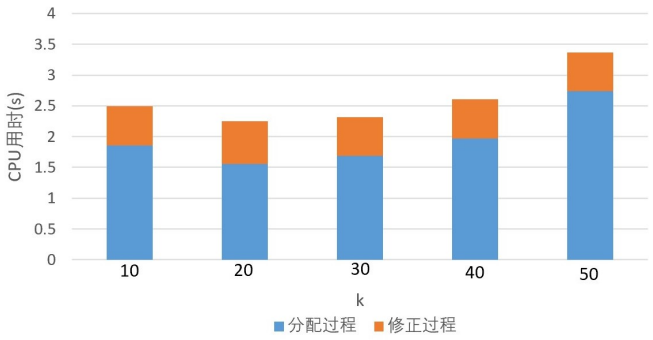
创新算法与原算法效率对比. 对比了并行化并考虑边真实长度的算法与非并行化且不考虑边真实长度的原算法的分配过程和普通修正过程 (以已有轨迹为图心路径) 在不同轨迹数据规模 |D| 下的运行时间，由 [6] 中进行的一系列对比，可以作为创新合理化算法高效性的一个辅证，结果见图5.2。由图可知，创新算法与原算法计算数量级相当，运行时间相近，合理化算法效率是可观的。并且随着数据规模变大，用时逐渐边长，在整个过程中分配过程用时占主导地位。

图 5.2 创新基线算法与原基线算法效率对比



k 的大小的分析. 对比不同 k 取值时，分配过程与普通修正过程的用时变化。见图5.3。可知随着 k 的增大，分配与普通修正用时也缓慢增加，相关系数由斜率容易判断出小于 1，显然相关度不及数据规模的影响。

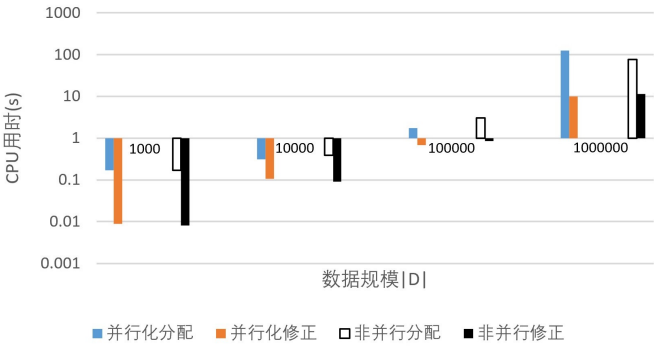
图 5.3　不同 k 对分配与修正过程用时影响



加速技巧的分析. 控制变量进行实验，探究反向索引和并行化两个技巧对算法效率的提升效果。通过对比使用反向索引与不使用反向索引 (见图5.5，5.6)，使用并行化与不使用并行化 ((见图5.4)，在各个轨迹规模的分配与修正过程运行时间来进行分析。相比较之下并行化技巧提速效果没有反向索引明显且受运行环境影响较大，但从理论分析，在相同的环境下还是对程序速度有一定提升作用。

反向索引技巧在数据规模越大的情况下加速效果越明显，这是容易理解的，因为此时需要的 EBD 距离计算更多，使用反向索引简化计算节省的时间也就越多。

图 5.4　并行化技巧提速效果



分配过程的分析. 统计在不同数据规模下，利用削减距离计算次数的技巧而减少的距离计算次数与总距离计算次数 (削减距离计算次数加实际距离计算次数)，见图5.7，由百分比直方图可见利用三角不等式可以节省大部分的距离计算，从而有效地提升算法效率。

并记录在默认参数下，使用普通修正与 CPEP 修正算法时每次迭代改变了所

图 5.5　反向索引技巧提速效果

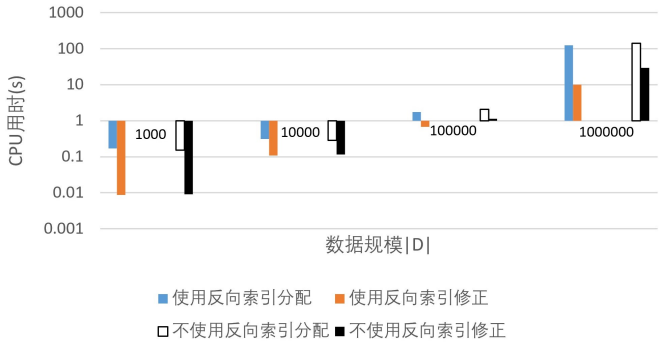


图 5.6　反向索引技巧在原算法中提速效果

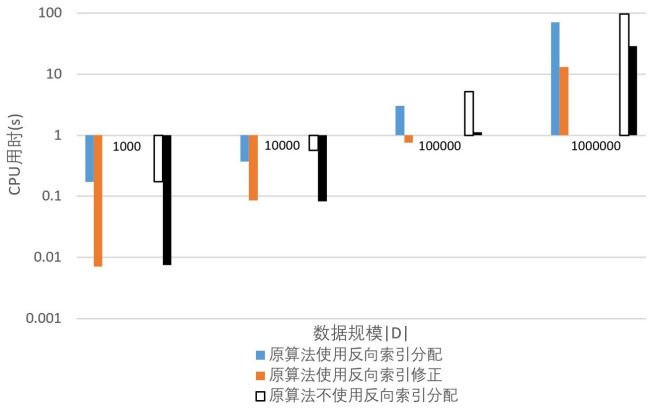
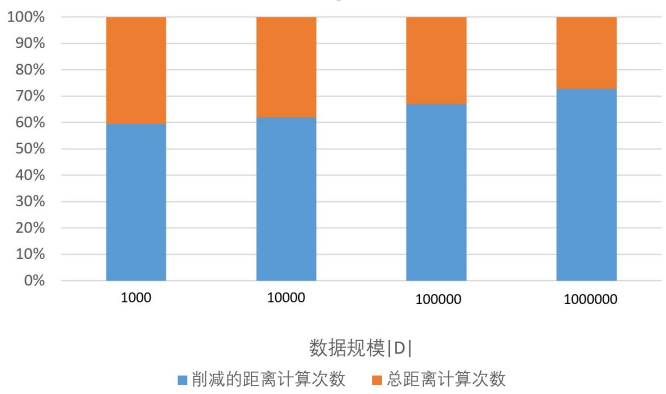


图 5.7　削减距离计算次数占总计算次数比例



属聚类的轨迹数目，验证合理化后两种算法的收敛性，见图5.8，5.9，由图也可知使用 CPEP 修正算法平均迭代次数多于采用普通修正的算法。

CPEP 修正过程的分析. 统计每次迭代中 CPEP 修正过程达到最终最优图心路径延拓迭代次数，这是 CPEP 算法中需要设定的唯一参数，见图5.10，可知数据规模为 1000,00 时，设置最大 4000 次的延拓次数是合理的，数据规模与迭代次数的关系可以在未来的工作中进一步研究。

普通修正过程与 CPEP 修正过程的对比分析. 统计使用普通修正过程和

图 5.8　使用普通修正的算法收敛性验证

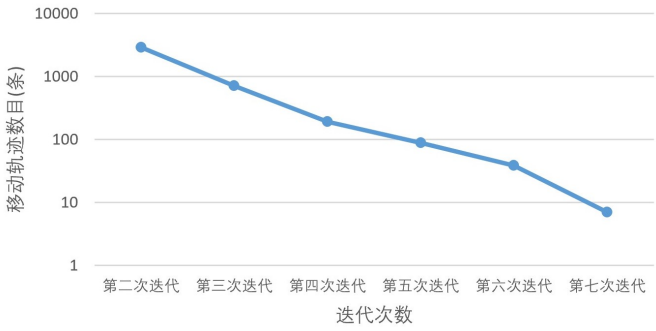


图 5.9　使用 CPEP 修正算法的算法收敛性验证

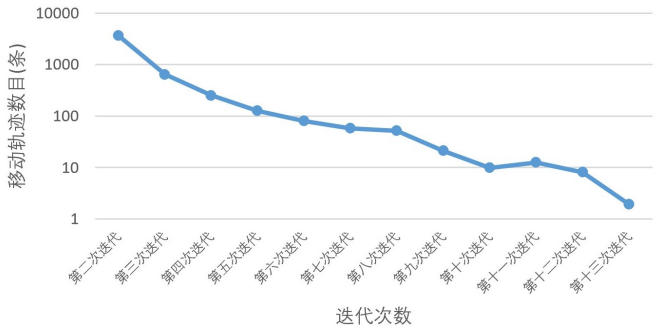
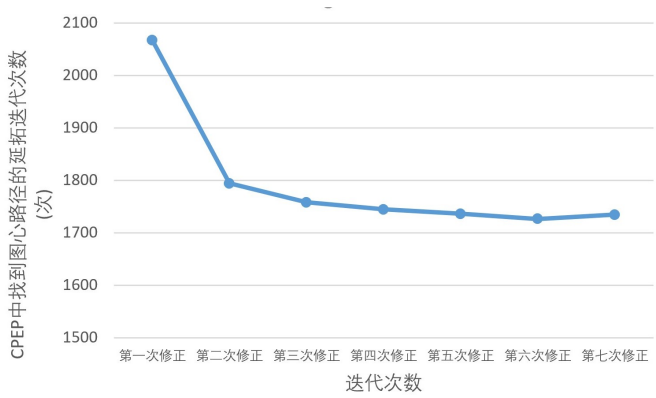


图 5.10　CPEP 修正中延拓迭代次数



CPEP 修正过程时的两种算法返回最终的 k 条图心路经所需分配和修正时间(图5.16) 以及最后的目标值大小 (图5.15)。

由图5.15可知 CPEP 修正算法能够有效进一步优化目标函数，但是实验过程中也发现，CPEP 内的 4000 的迭代次数并不适用于所有数据规模，比如 1000,00 及以下规模使用 4000 效果较好，而 1000,000 量级的轨迹的迭代结果则出现了更多的波动，值得未来进一步研究。

但由图5.16可以看出 CPEP 修正算法与普通修正算法最大的不同在于修正过程的用时，CPEP 修正算法虽然优化了目标函数，用时虽然同属百秒量级但是也

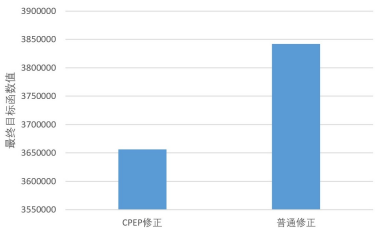


图 5.11　|D| = 1000

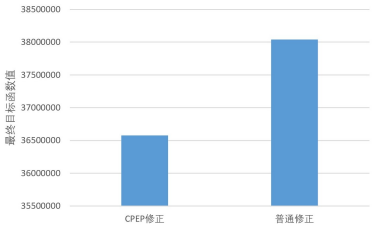


图 5.12　|D| = 10000

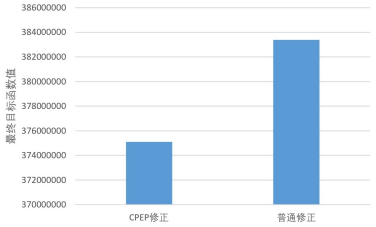


图 5.13　|D| = 100000

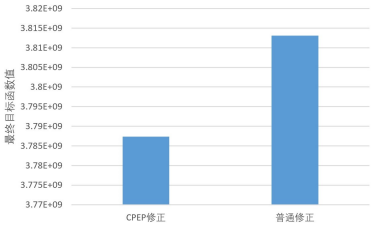
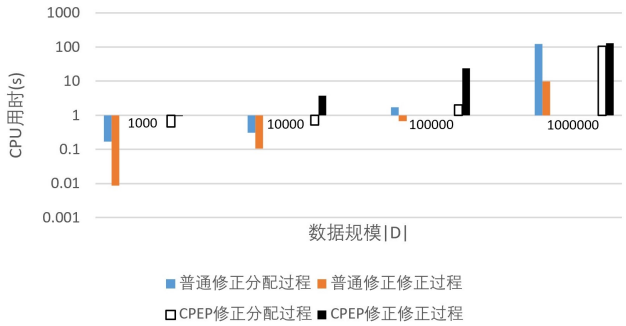


图 5.14　|D| = 1000000

图 5.15　不同数据规模下，两种修正算法得到的最终目标值

是普通修正算法的数倍，这是考虑边真实长度带来的结果，也是算法的一个待提高之处。

图 5.16　采用普通修正与 CPEP 修正的运行用时对比



初始化方法分析. 对比使用 k-means++ 算法生成的初始值与随机生成初始值时算法的分配以及修正时间，以及最后的目标函数值。

由图5.17与图5.18可知，以 k-means++ 算法初始化，平均而言的确可以使算法收敛到更优结果，并且运行时间差别不大。这里将 CPEP 修正算法也加入比较，说明在优化目标函数方面使用 CPEP 修正算法比使用 k-means++ 初始化方法效果更佳。

读取图文件并建立反向索引的用时. 文件读取与反向索引建立的用时只与数据规模相关和图中的连接度 (connectivity) 相关，而不是和数据规模成简单正比关系，见图5.22。

图 5.17　不同初始化方法与运行时间

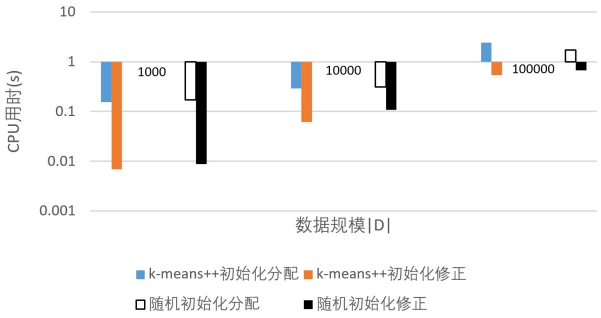


图 5.18　不同初始化方法与最终目标函数值

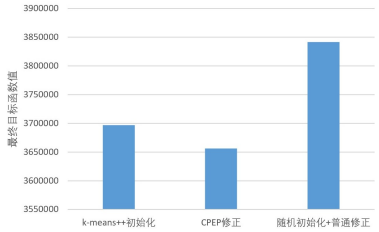


图 5.19　|D| = 1000

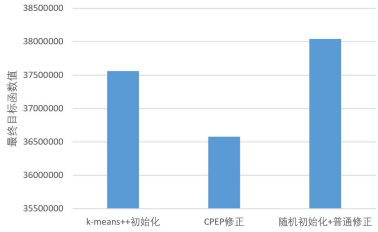


图 5.20　|D| = 10000

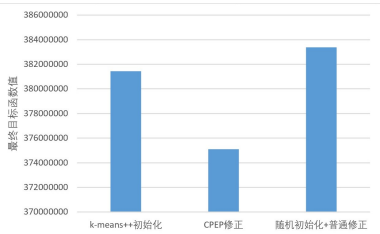
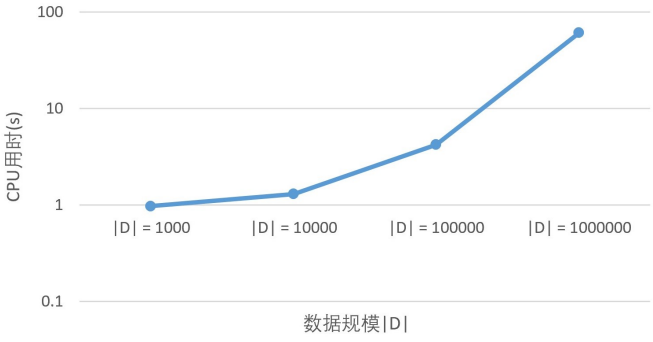


图 5.21　|D| = 100000

图 5.22　读取图文件并建立反向索引用时



第六章　总结与展望

本文成功复现了基于 EBD 度量，采用反向索引与三角不等式削减距离计算次数的聚类算法，用实验证明了这些技巧的有效性，并在其基础上改良，得到了合理化的可行算法，用实验证明了新算法的可行性与高效。但是在饰演的过程中，也有一些尚未解决的问题可供将来研究。

- 前文谈到过简化的聚类目标值计算公式的适用范围，但是只进行了定性的描述，而没有定量的分析，可以进一步分析要使用简化的聚类目标值公式具体对数据集有怎样的要求。
- 因为在本文实验中，并行化算法带来的提速效果并不明朗，所以可以对并行化算法对算法的性能影响进行更细致的分析。
- CPEP 修正算法内，搜索迭代的次数与数据规模，图的连接度的关系，本文中的设定是依靠实验测试。
- 进一步优化本文中的 CPEP 修正算法耗时。

参 考 文 献

[1] https://perma.cc/BPM2-QNW4.

[2] Yu Zheng. Trajectory Data Mining: An Overview. ACM Transactions on Intelligent Systems and Technology[J], 2015, 6(3)

[3] Lee, Jae-Gil, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework[C]. Proceedings of the 2007 ACM SIGMOD international conference on Management of data, 2007:593-604

[4] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise[C]. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996: 226–231

[5] S. Wang, Z. Bao, J. S. Culpepper, Z. Xie, Q. Liu and X. Qin. Torch: A search engine for trajectory data[C]. The 41st International ACM SIGIR Conference, 2018: 535-544

[6] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, Xiaolin Qin. Fast Large-Scale Trajectory Clustering[C]. Proceedings of the VLDB Endowment, 2019: 29-42

[7] https://www.openstreetmap.org/#map=10/51.4749/6.3896

[8] https://www.graphhopper.com/

[9] Paul Newson, and John Krumm. Hidden Markov map matching through noise and sparseness[J]. Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, 2009: 336-343.

[10] Arthur, David , and S. Vassilvitskii. K-Means++: The Advantages of Careful Seeding[C]. Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 2007: 1027-1035

[11] Baeza-Yates, Ricardo, Salinger, Alejandro. Fast Intersection Algorithms for Sorted Sequences[J]. Algorithm and Applications, Lecture Notes in Computer Science, 2010, 6060: 45-61.