基于轨迹微分段模型的快速地图匹配方法

康军, 高晟恺, 来嘉宝

引用本文

康军, 高晟恺, 来嘉宝. 基于轨迹微分段模型的快速地图匹配方法[J]. 计算机科学, 2026, 53(4): 252-259.

KANG Jun, GAO Shengkai, LAI Jiabao. Fast Map Matching Method Based on Trajectory Micro-segment Model [J]. Computer Science, 2026, 53(4): 252-259.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.250900115

基于轨迹微分段模型的快速地图匹配方法

康　军　　高晟恺　　来嘉宝
长安大学信息工程学院　西安 710018

摘　要　地图匹配是智能交通系统中的核心技术之一，旨在将GPS轨迹数据映射至城市路网上，消除定位误差并还原实际行驶路径。随着GPS轨迹数据量的爆炸性增长，传统的基于隐马尔可夫模型(HMM)的地图匹配方法因高计算成本和时序依赖性问题，难以满足实时处理要求。为此，提出了一种基于轨迹微分段模型的快速地图匹配方法(Micro-Segment Fast Matching，MSFM)。该方法基于滑动窗口机制，将轨迹分解为固定长度的微轨迹段，在分布式计算环境中利用向量化计算方法，在兼顾地图匹配准确性的条件下大幅度提升了计算效率。实验结果表明，在给定的分布式集群环境下，MSFM实现了约110000点/秒的地图匹配速度，比基准算法快约7倍，同时保持了95.86%的匹配准确率。MSFM方法通过改进轨迹数据的存储结构，在高效实时处理大规模轨迹数据方面具有显著的性能优势。

关键词：地图匹配；轨迹微分段；隐马尔可夫模型；分布式计算；向量化计算
中图分类号　TP311.13

Fast Map Matching Method Based on Trajectory Micro-segment Model

KANG Jun，GAO Shengkai and LAI Jiabao
School of Information Engineering，Chang'an University，Xi'an 710018，China

Abstract　Map matching is one of the core technologies in intelligent transportation systems，aiming to map GPS trajectory data to urban road networks，eliminate positioning errors，and reconstruct actual driving paths. With the explosive growth of GPS trajectory data volume，traditional HMM(Hidden Markov Model)-based map matching methods struggle to meet real-time processing requirements due to high computational costs and temporal dependency constraints. To address this，this paper proposes a fast map matching method based on a trajectory micro-segment model(Micro-Segment Fast Matching，MSFM). By leveraging a sliding window mechanism，the proposed method decomposes trajectories into fixed-length micro-segments and employs vectorized computation techniques in a distributed computing environment. This approach significantly improves computational efficiency while maintaining map matching accuracy. Experimental results demonstrate that，under a distributed cluster configuration，MSFM achieves a map matching speed of approximately 110000 points per second，which is 7 times faster than baseline algorithms，while retaining a 95.86% matching accuracy. By optimizing the storage structure of trajectory data，the MSFM method exhibits significant performance advantages in real-time processing of large-scale trajectory data with high efficiency.

Keywords　Map matching，Micro-segment，Hidden Markov model，Distributed computing，Vectorized computation

1　引言

随着移动设备与GPS技术的迅速普及，车辆轨迹数据已成为智能交通系统和自动驾驶领域的关键数据来源[1]，支撑着交通流量预测[2]、路径推荐[3]等关键应用。然而，复杂城市环境(如高架桥、隧道及恶劣气候)导致的GPS信号漂移，使得原始轨迹数据往往与实际路网存在较大偏差[4]。因此，地图匹配技术作为减少GPS定位误差、恢复车辆实际行驶路径的重要手段，已成为智能交通领域的核心技术之一。

隐马尔可夫模型(Hidden Markov Model，HMM)[5]因其优秀的时空序列数据建模能力，已成为高精度地图匹配的主流方法[6]。但其较高的计算成本，影响了对大规模轨迹数据的实时处理能力。近年来，分布式计算框架(如Apache Spark[7])为提高地图匹配效率提供了新的解决思路，将地图匹配任务划分为多个子任务进行并行处理，能够显著提升地图匹配效率。

HMM将地图匹配问题建模为隐状态的全局解码问题，具体采用Viterbi算法求解对于一个轨迹段的最佳匹配路段，其中轨迹段指单个移动对象由其停留点划分的时间连续的轨迹点序列。但Viterbi采用递归的方式进行全局解码，即每一时刻的解码结果依赖于之前所有时刻的解码结果，各计算步骤之间存在时序依赖性。这种计算模式使得现有并行地图匹配方法仅能以轨迹段为最小并行计算单元，在一个轨迹段内只能通过逐个轨迹点顺序迭代计算的方式进行处理，无法充分利用现有计算框架中向量化计算的优势，从而限制了地图匹配方法性能的进一步提升。

到稿日期：2025-09-17　返修日期：2025-11-13
通信作者：康军(junkang@chd.edu.cn)

针对上述问题，本文提出了一种基于轨迹微分段模型的快速分布式地图匹配方法（MSFM），基于滑动窗口机制将轨迹段进一步划分为具有固定长度且部分重合的轨迹微分段，即将原来由轨迹点序列构成的轨迹段转换为由轨迹微分段构成的序列，通过轨迹微分段之间的部分重合弱化其间的时序依赖性，从而可使用向量化操作方式一次性对轨迹段的所有轨迹微分段执行 Viterbi 算法，并最终仅将轨迹微分段中某个轨迹点的地图匹配结果作为整个轨迹微分段的地图匹配输出。本文方法通过对轨迹数据的转换，充分利用了目前计算框架中向量化操作的性能优势，进一步大幅度提高了地图匹配方法的执行效率。本文的主要贡献如下：

1）为了在地图匹配过程中进一步降低 I/O 开销，首先设计了一种离线的具有时间约束的路径生成算法，离线生成测试区域路网的最短路径集合。该最短路径集合将作为静态数据集加载内存，在地图匹配时，通过 Hash 表查询方式获取计算转移概率时需要的最短路径数据，以此消除在线最短路径搜索导致的性能瓶颈问题。

2）提出了一种轨迹微分段数据模型。采用滑动窗口机制将轨迹段划分为部分重合的微轨迹段，使用向量化操作一次性对轨迹段的所有微轨迹段执行 Viterbi 算法，并最终仅将微轨迹段中某个轨迹点的地图匹配结果作为整个轨迹微分段的地图匹配输出。

3）通过实验确定了轨迹微分段的最佳长度。实验显示，所提方法在兼顾准确率的条件下，可以大幅度提高地图匹配效率。

本文方法在西安市真实轨迹数据集上进行了测试。在 3 节点的集群环境中，地图匹配速度可达到 110000 点/秒，较传统 FMM 算法[8]提升约 7 倍，同时兼顾较高的地图匹配准确率，表明了所提方法在大规模轨迹数据实时地图匹配中的优势。

2　相关工作

现有的优化地图匹配性能的方法主要可分为 4 类，分别是路网空间索引优化、轨迹精简、最短路径计算加速以及并行化处理。

为了加速候选路段的筛选，部分研究通过构建 R 树[9]、四叉树[10]或者网格索引[11]等方法对路网建立空间索引，但这类方法对提高地图匹配整体性能非常有限。

为简化地图匹配过程的计算，一部分研究者提出了化简轨迹或减少轨迹点数目的策略。例如，文献[12]通过在保留轨迹主要特征（如转折点与曲率变化）的前提下对轨迹进行简化，大幅减少待匹配的 GPS 点数量。需要注意的是，尽管简化计算可以减轻匹配负担，但此类方法往往会损失轨迹的局部细节，从而在一定程度上影响地图匹配的准确率。

传统 HMM 地图匹配方法在计算两个连续轨迹点候选路段之间的转移概率时，需要获得最短路径，而经典的最短路径搜索算法通常具有较高的时间复杂度。在大规模的路网环境下，在线最短路径搜索过程成为地图匹配的主要性能瓶颈。针对这一问题，一些研究采用了加速算法[13]，如 A* 算法[14]或者 ALT 算法[15]。但这些方法均基于原始路网在线搜索，其加速效果仍然有较大的提升空间。文献[16]通过对路网进行预分段和利用邻接关系来快速近似路网距离，并根据 GPS 采样密度动态选择加速或传统计算方式，同时通过地图拼接技术将大规模匹配问题分解为若干子问题，从而显著缩短了最短路径的计算时间。另一方面，FMM 算法[8]通过预计算并缓存一定空间范围内路段对的最短距离，实现了查表式的最短路径查询，但其匹配准确率依赖于预计算表的命中率。此外，文献[17]引入自适应滑动窗口和扩展 Viterbi 算法的二阶 HMM 在线匹配方法，在平衡准确率与计算效率方面取得了进展。

上述方法在提升地图匹配方法性能方面均取得了一定效果，但在处理大规模轨迹数据时，仍然难以满足实时处理的需求。因此，并行化地图匹配方法逐渐成为研究热点。早期一些研究[18]尝试利用 Hadoop[19]框架进行地图匹配，但由于大量磁盘 I/O 操作，其实时性能受到较大制约。近年来，以 Apache Spark 为代表的内存式大数据计算框架在实时轨迹处理领域获得了广泛应用。例如，文献[20]采用四叉树空间索引与 Apache Sedona（原 GeoSpark）[21]技术，实现了面向大规模 GPS 轨迹数据的高效并行化的基于 HMM 的地图匹配方法。然而，尽管并行 HMM 地图匹配方法在实践中展示出其可行性，但由于 HMM 地图匹配过程中单个轨迹段内存在的时序依赖性，使得已有方法仍未能充分利用目前大数据计算框架普遍支持的向量化操作带来的性能提升的优势。并行化 HMM 地图匹配方法仍存在性能提升的空间。

综上，本文提出了一种融合离线最短路径生成、轨迹微分段模型的并行化 HMM 地图匹配方法，实现了面向大规模轨迹数据的快速地图匹配。

3　问题描述

地图匹配可被建模为 HMM 中的全局解码问题，其中 GPS 轨迹点为观测值，而其对应的匹配路段则为隐状态。地图匹配问题就是已知路段集合，根据观测到的 GPS 轨迹点序列，通过 HMM 全局解码算法获得其对应的全局最优的匹配路段序列的过程。下面给出本文中涉及到的主要概念的定义。

定义 1（轨迹点）　轨迹点 $p_i=(ID,x,y,t)$ 表示某 ID 的移动对象在 t 时刻采样的位置坐标 (x,y)。

定义 2（轨迹段）　由同一个移动对象按时间顺序连续产生的轨迹点序列，记为 $T=\langle p_1,p_2,\cdots,p_n\rangle$。

定义 3（轨迹微分段）　通过窗口尺寸为 M、步长为 1 的滑动窗口，从轨迹段中连续提取出的固定长度为 M 的连续轨迹点子序列。对于长度为 n 的轨迹段 T，其可划分为 $n-1$ 个微轨迹段，可分别表示为 $T_1^M=\langle p_1,p_2,\cdots,p_M\rangle,\cdots,T_{n-1}^M=\langle p_{n-M+1},p_{n-M+2},\cdots,p_n\rangle$。

定义 4（路网）　路网是一个有向图 $G=\langle V,E\rangle$，其中 $V=\langle v_1,v_2,\cdots,v_n\rangle$ 是节点（交叉路口）的集合，$E=\langle e_1,e_2,\cdots,e_n\rangle$ 是有向边集。每条边 $e_i=(v_i,v_j)$ 表示一条从节点 v_i 到 v_j 的道路段。

本文提出的快速地图匹配算法是基于轨迹微分段模型对经典 HMM 地图匹配算法[6]的并行化实现，其中地图匹配中观测概率、转移概率的计算均使用了文献[16]中的方法。

4 并行化快速地图匹配方法

4.1 轨迹数据预处理

1)轨迹数据清洗与异常点过滤:计算相邻轨迹点之间的移动速度v_i。

$$v_i=\frac{Haversine(p_i,p_{i-1})}{t_i-t_{i-1}} \tag{1}$$

其中,$Haversine(\cdot)$为地球球面距离的计算式,若速度超过预设阈值,则判定 p_i为异常点并剔除。

随后将原始轨迹点数据$\langle ID,x,y,t\rangle$拓展为含有速度信息的格式$\langle ID,x,y,t,v\rangle$,以便后续处理。

2)停留点检测:采用基于速度阈值与上下文约束的停留点检测算法。其核心规则为:(1)若当前轨迹点速度$v_i<\delta$,则标记为停留点;(2)若$v_i\geqslant\delta$,但与之相邻的前后点速度$v_{i-1}<\delta$且$v_{i+1}<\delta$,则判定该点处于停留区间内,同样标记为停留点。此处的δ为速度阈值,在本文中设置为 10 km/s。

3)轨迹分段:利用检测到的停留点将轨迹进行分段。若相邻点时间间隔大于t',则视为不同轨迹段。此处的时间间隔t'的取值与下一节最短路径搜索时的时间阈值 T 相同。

4)轨迹数据结构化处理:为了在后续处理中进行向量化操作,将清洗后的数据转换为具有轨迹段编号的结构化格式$\langle segID,Point(x,y),t\rangle$。其中,$segID$ 为轨迹段的唯一标识,而 $Point(x,y)$代表轨迹点。

4.2 路网数据预处理

4.2.1 构建 R 树索引

为了加速候选路段的搜索效率,本文采用 R 树空间索引技术对路网数据进行分布式构建与优化。R 树将相邻路段组织为最小外接矩形(Minimum Bounding Rectangle,MBR),形成层次化的空间索引结构,可显著提升范围查询(如轨迹点邻域内的候选路段搜索)效率。相较于四叉树、网格索引等其他空间索引方法,R 树在动态更新与不规则空间分布数据的适配性上更具优势,尤其适用于城市路网中路段长度、方向差异显著的场景。

使用 Apache Sedona 分布式地理信息计算引擎对路网数据以并行化方式构建 R 树索引。由于城市路网数据规模有限(如西安路网仅包含 75639 条边,存储占用不足 100 MB),R 树索引可完整广播至集群所有节点。这样的设计可避免分布式环境下索引跨节点查询的网络传输开销,使得候选路段搜索完全在本地内存中完成。

4.2.2 最短路径生成

为了优化基于 HMM 的地图匹配中的最短路径搜索,本文提出了一种基于有向图的最短路径生成算法。该算法利用在全路网内搜索时间阈值 T 内的可达最短路径,构建压缩路径表以支持在线高效查询。时间阈值 T 的设置有以下两个目的。1)轨迹分段的时间间隔 Δt 与路径生成算法的时间阈值 T 设置为相同值。这是因为相邻两个轨迹点之间的时间间隔不会超过t',从而确保两点相对应候选路段的最短路径必定存在于预计算的最短路径表中。这不仅可以避免因最短路径表大小限制而对匹配准确率产生负面影响,而且当查询未命中时,可以明确判定出目标路段在 200 s 内不可达,也就是该路段对在路网中并不连通,直接返回一个较大的默认值即可。2)限制算法的迭代次数,如果当前路径的累计通行时间超过时间阈值 T,便停止该路径的进一步生成。具体过程如算法 1 所示。

输入的路径集合 P 中包括所有路段及其相邻关系,路径集合的格式为$\langle pID,nID,length,t\rangle$。其中,$pID$ 代表该路段,nID 代表与该路段直接相邻的路段,t 为该路段的通行时间。若该路段有多条相邻路段,那么在路径集合中与之对应多条记录。

随后进行迭代,对于第 k 轮迭代,将当前路径集合 P 与中间结果集合 R'进行运算,生成新的中间结果集,运算规则为 $R'\bowtie_{R'.tID=P.sID}P$。

接下来用一个实际的例子来阐述路径生成算法,路网结构如图 1 所示,图中的每个边都是单向边。以路段e_1为例,初始化的中间结果集中会有两条以路段e_1为起始路段的路径,分别为$e_1\rightarrow e_2$以及$e_1\rightarrow e_4$,并记录两条路径的长度以及累计通行时间。在一次迭代时,这两条路径会与路径集合 P 进行运算,由于e_2与e_3和e_6直接相邻,e_4与e_5和e_7直接相邻,因此会形成 4 条新的路径,$e_1\rightarrow e_2\rightarrow e_3$,$e_1\rightarrow e_2\rightarrow e_6$,$e_1\rightarrow e_4\rightarrow e_5$以及$e_1\rightarrow e_4\rightarrow e_6$,同时计算这 4 条新路径的长度以及累计通行时间,将其存入中间结果集 R'中。

中间结果集合 R'用于存储当前算法生成的中间结果,格式为$\langle sID,tID,length,links,t\rangle$,$sID$ 表示起始路段,tID 表示终止路段,$links$ 表示从起始路段到终止路段的最短路径,$length$ 表示最短路径长度,t 表示累计通行时间,累计通行时间是这条路径所有路段通行时间的和并加上每个路口的通行时间。路口的通行时间设置为一个平均通过时间(10 s),如果该路径包含 5 个路段,那么就会经过 4 个路口,累计通行时间除了 5 个路段的通行时间,还包括 40 s 的路口通行时间。

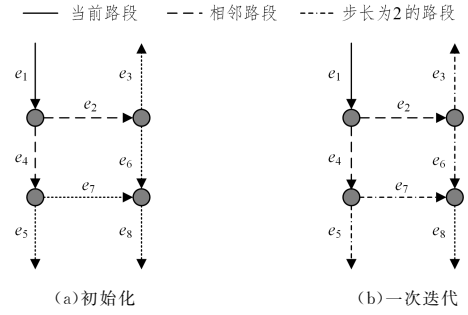


图 1 最短路径生成算法的示例

Fig. 1 Example of shortest path generation algorithm

算法 1 路径生成算法

输入:路径集合 P,时间阈值 T,最大迭代次数 maxIter,中间结果集合 R′

输出:预计算的最短路径集合 R

1. 初始化中间结果集合R′←G
2. While t≤T
   2.1. R′←R′ $\bowtie_{R'.tID=P.sID}$ P
      2.2.1. 计算总通行时间以及路径长度
      2.2.2. 过滤中间集合中的回路,使得 R′.sID! =R′.tID
      2.2.3. 对于相同路径,仅保留长度最短
   2.2. R←R′,若某记录累计通行时间大于时间阈值 T,则在中间集

合中过滤该记录，停止生成

2.3. END

3. R←R′

4. RETURN R

在计算每条新路径的长度和总通行时间后，会过滤回路，回路即为起始路段与终止路段相同的路径。对于相同起始路段和终止路段的情况，算法会形成多条路径，称为 k 最短路径，仅保留长度最小的路径，其余视为冗余路径被丢弃。若某路径的累计时间超过时间阈值 T 时，则直接输入到结果集而不参与后续迭代。预生成的路径集合 R 以哈希表形式存储，可用于在线查询。

4.3　地图匹配阶段

基于轨迹微分段的快速地图匹配方法的数据流如图 2 所示。

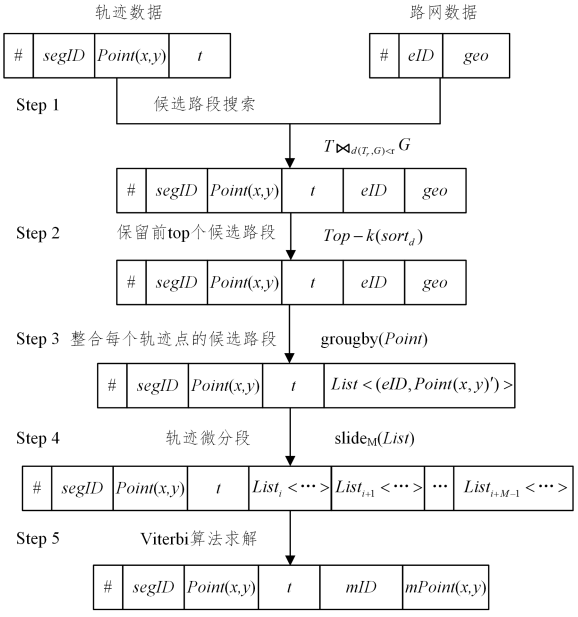


图 2　地图匹配数据流

Fig. 2　Map matching data flow

Step 1　候选路段搜索

首先针对预处理后的轨迹数据集（格式为〈$segID$，$Point(x,y),t$〉）依次计算其候选路段。以轨迹点 p_i 为中心，选取半径为 r 的圆形区域内的候选路段，具体操作如下：

$$T \bowtie_{d(Tr,G)<r} G \tag{2}$$

其中，T 表示预处理后的轨迹数据集；$d(Tr,G)$ 表示数据集中轨迹点与路网的距离，即为投影距离。如果每个轨迹点有多个候选路段，经过上述处理后生成的数据格式为〈$segID$，$Point(x,y),t,eID,geo$〉，其中 eID 表示候选路段的 ID，geo 表示该候选路段的几何信息。如果某个轨迹点有多个候选路段，那么新生成的数据集中会对应多条记录。

Step2　候选路段筛选

接下来针对每个候选路段，利用几何信息 geo 确定轨迹点在其候选路段上的投影点 p_i'，并计算二者之间的大圆距离 d，产生的数据格式为：

〈$segID,Point(x,y),t,eID,Point(x,y)'$〉

随后，根据距离 d 对候选集进行排序，并且仅保留距离最近的 top 个候选路段，以控制计算复杂度。

Step 3　候选路段聚合

由于单个轨迹点可能会有多个候选路段，因此会产生多条记录，需要将其聚合为列表结构，合并为一条记录。为此以轨迹点为分组单位，整合同一分组内的所有候选记录，形成的数据格式为：

〈$segID,Point(x,y),t,List$〈$(eID,Point(x,y)')$〉〉

其中，列表字段包含该点的全部候选路段信息，包括所有的候选路段的 ID 以及投影点。这种聚合方式便于后续对轨迹进行微分段。

Step 4　轨迹微分段

HMM 模型的时序依赖性通常使得前一状态计算结果对当前状态具有决定性影响，导致传统方法必须在轨迹段内串行处理轨迹段，从而限制了并行计算。假设某轨迹中包含 k 个轨迹点，状态空间大小设置为 N，对于轨迹中的每个轨迹点，需要计算每个状态的概率以及考虑前一个轨迹点的状态转移到当前状态的概率，则每个轨迹点的处理时间复杂度为 $O(N^2)$。在计算时，Viterbi 算法的迭代次数与轨迹点的数目相关，那么 k 个轨迹点使用 Viterbi 算法求解的时间复杂度为 $O(kN^2)$。这说明算法的复杂度受轨迹点数目的影响，在实际数据集中每条轨迹的轨迹点数目是不一致的，会面临数据不均衡的问题，导致计算时的节点负载不均衡。

如图 3 所示，使用 Viterbi 算法进行地图匹配时，轨迹段 T_1 是从轨迹点 p_1 开始逐步计算最优路径，并且轨迹段 T_1 与轨迹段 T_2 的轨迹点数目不同。为解决上述问题，本文采用滑动窗口机制对轨迹执行微分段策略，如图 4 所示。以当前轨迹点为中心，向后滑动窗口构造包含 M 个连续点的局部窗口，将整个轨迹分解为多个互相独立的微轨迹段。最终形成的数据格式为：

〈$segID$，$Point(x,y)$，t，$List_i$〈…〉，$List_{i+1}$〈…〉，…，$List_{i+M-1}$〈…〉〉

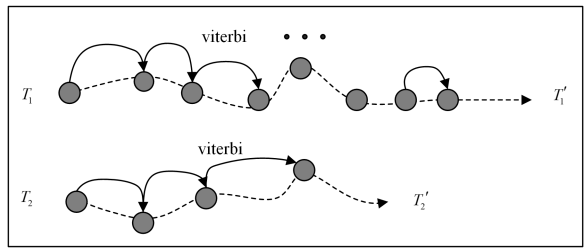


图 3　以轨迹段为单位地图匹配

Fig. 3　Map matching based on trajectory segments

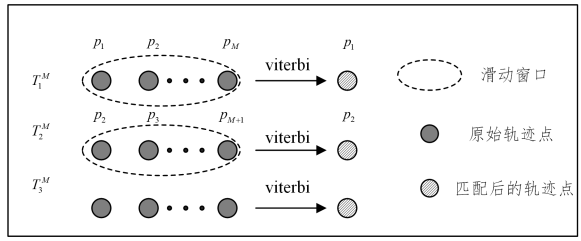


图 4　以轨迹点为单位地图匹配

Fig. 4　Map matching based on trajectory points

该设计的优势在于：1）充分利用 Spark 计算框架中向量

化操作的性能优势，而传统 HMM 方法往往只能以整条轨迹为单位并行匹配，受限于轨迹条数；2）通过滑动窗口对轨迹进行微分段，使得数据集规整，避免数据倾斜带来的性能瓶颈；3）如果某个窗口内轨迹点因噪声匹配出错，则这个错误仅影响当前轨迹点的结果，不会导致后续轨迹点偏离。

Step 5 Viterbi 算法求解

最终在每个窗口内独立应用 Viterbi 算法求解最优匹配路径，以确定窗口内首个轨迹点的匹配结果。针对窗口内的每个轨迹点候选路段状态，基于预生成的路网拓扑最短路径距离构建状态转移概率矩阵，同时计算投影距离对应的观测概率，并采用维特比算法递推求解各候选路段的最大累计联合概率，最终回溯得到窗口内首个轨迹点的匹配结果，输出数据格式为〈$segID$, $Point(x,y)$, t, $mPoint$, mID〉($mPoint$ 为该轨迹点的最优投影点，mID 为匹配路段 ID)。不同于轨迹段作为输入计算后得到整个轨迹段的匹配结果，以轨迹片段为输入，采用 Vitertbi 算法求解得到的是单个轨迹点的匹配结果。

通过上述设计，本文实现了轨迹点级别的并行地图匹配，大幅提升了大规模轨迹数据的处理效率。

5 实验及结果分析

本章通过一系列实验，对 MSFM 方法进行了全面评估。实验内容主要涵盖以下几个方面：实验环境和数据集介绍、关键评估指标、与基准方法(FMM)的对比、参数敏感性分析、算法各模块时间占比以及扩展性验证。

5.1 实验环境和数据集

实验在由 3 台 8 核、32 GB 内存的 ecs.g7.2xlarge 服务器组成的 PC 集群上进行，所有节点均采用 2.7 GHz CPU。轨迹数据选自陕西省西安市出租车数据集，涵盖 11 300 辆出租车在 2016 年 9 月至 11 月的采样轨迹，相邻轨迹点的采样间隔约为 30 s，分段后每条轨迹平均包含 42 个 GPS 点。路网数据覆盖 49 km×44 km 区域，共包含 75 639 条有向边，每个双向道路段以两个相反方向的有向边表示。

为了评估不同采样间隔对模型性能的影响，本文采用非重叠二次采样方法构建实验数据集。以原始 30 s 采样间隔的轨迹数据为基础，通过系统性地抽取子样本，生成了采样间隔分别为 60 s，90 s 和 120 s 的轨迹数据。例如，使用原始的轨迹数据，每隔一个点抽取一个样本即可得到采样间隔为 60 s 的轨迹数据。依此类推，获得采样间隔为 60 s，90 s 以及 120 s 的轨迹数据。

5.2 评估指标

为评估地图匹配效果，本文采用以下指标。1）匹配速度：定义为分布式集群每秒处理的轨迹点数(GPS 点/秒)，反映系统对大规模数据的实时处理能力。2）匹配准确率：选取西安市出租车数据集中 50 条时空分布多样的轨迹(涵盖城区主干道、高架路及交叉路口场景)，人工标注其真实路径作为基准，准确率定义为匹配后轨迹包含的路段数与真实轨迹路段数之比。

准确率 A 定义为：

$$A=\frac{N_{out}}{N_{tr}}$$

其中，N_{tr} 表示真实轨迹所包含的路段数，N_{out} 表示匹配后的轨迹所包含的路段数。

比较以下方法的地图匹配准确率和匹配速度，所使用数据都是经过预处理后的分段轨迹数据集。

1）MSFM：本文方法，将数据进行结构化处理并对轨迹段再进行细分，最终实现轨迹点的并行地图匹配。

2）FMM：通过预计算技术离线计算最短路径并压缩存储，在计算最短路径时查表获取，从而加快地图匹配。

3）D-FMM：由于 FMM 是单机环境，为此在 Spark 集群中实现 FMM 的分布式版本 D-FMM。

5.3 最短路径生成算法实验

在单机环境下统计预计算的最短路径数，如表 1 所列。随着时间阈值 T 的增大，预计算生成的最短路径数量以及相应的算法执行时间均呈现出显著增长的趋势。这是由于更大的时间阈值 T 需要进行更多轮迭代生成路径，使得总体计算量和生成路径数量呈指数级增长，导致算法的执行时间也大幅上升，同时更多的最短路径也需要占用较多的存储空间。为此，选择 200 s 作为最终的时间阈值。

表 1 最短路径数量

Table 1 Number of shortest paths

时间阈值 T/s	最短路径对数量	执行时间/s
60	889 778	143.42
90	1 500 031	257.09
120	2 295 179	464.99
150	3 290 786	745.22
180	4 506 833	1 233.60
200	5 955 324	1 988.50

5.4 地图匹配实验

所有实验均在 Apache Spark 3.4.3 集群环境下运行，选取一天预处理后的轨迹数据进行分布式匹配，经过预处理后共有 13 333 354 个轨迹点。

首先分析 MSFM 的关键参数对匹配效果的影响。实验重点评估候选路段搜索半径 r、保留的候选路段数量 top 以及滑动窗口尺寸 M 对匹配准确率与执行时间的影响。在不同采样间隔下对 MSFM 的各项参数进行了测试，实验结果表明，在不同采样间隔下，各参数对算法性能的影响未表现出明显差异。为方便讨论，后续关于 MSFM 的参数 r 和 top 对算法性能影响的分析，均以 30 s 采样间隔为例进行展开。而滑动窗口尺寸 M 为本文的核心创新参数，对算法的准确率与效率有更复杂的影响。因此，单独对 M 进行分析。

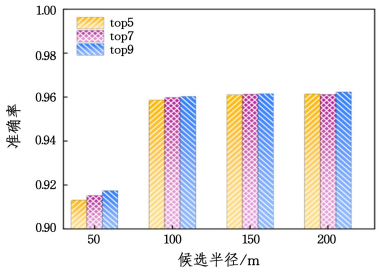
表 2 列出了不同搜索半径所包含的平均候路段数量。当 r 从 50 m 增加至 200 m 时，平均候选路段数从 4.97 膨胀至 27.32，如果不进行过滤，则会导致过多的转移状态，影响整体的计算复杂度。为此，选取距离较小的前 top 条作为最终的候选路段，top 取值从 5 开始是因为搜索半径 $r=50$ m 时，平均候选路段数为 4.97。接下来分析搜索半径对性能的影响，如图 5 所示。以 $top=5$ 举例说明，当搜索半径从 50 m 增加到 100 m 时，准确率提升了 3.2%，但是执行时间增加 68%。并且后续随着搜索半径的增大，准确率仅提升 0.15%，但其执

行时间大幅增加。这表明轨迹点的采样噪声基本不会超过100 m,过大的搜索半径会显著增加计算复杂度,但对准确率的提升作用有限。为此,搜索半径设置为100 m。

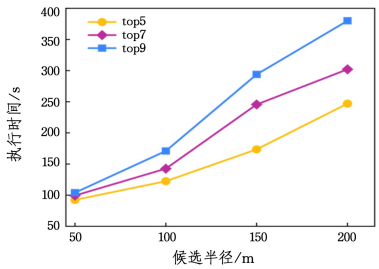
表2　平均候选路段数

Table 2　Average number of candidate road segments

搜索半径 r/m	平均候选路段数
50	4.97
100	11.25
150	18.48
200	27.32



(a)准确率随候选半径的变化



(b)执行时间随候选半径的变化

图5　候选半径和候选路段数目对地图匹配性能的影响

Fig. 5　Influence of candidate radius and number of candidate road segments on map matching performance

保留的候选路段数量 top 通过限制候选路段数量来减少转移状态。以 $r=100$ m 举例,当 $top=5$ 时,准确率达95.86%;进一步增加至 $top=7$ 时,准确率仅提升0.36%,但计算时间增加了16.5%。随后增加保留的候选路段数量也仅能略微提升准确率,这与搜索半径对性能的影响相似。因此,选定 $top=5$ 作为最佳 top。滑动窗口尺寸 M 控制窗口内轨迹点的个数,如图6所示,在不同采样间隔下滑动窗口尺寸对算法准确率与执行时间的影响趋势表现出高度的相似性。在准确率方面,当滑动窗口尺寸 $M=1$ 时,算法仅依赖单个点的投影距离确定匹配结果,导致准确率普遍较低。随着滑动窗口尺寸 M 增大至3时,算法可有效利用窗口内多个轨迹点的信息,使准确率大幅提升。此后继续增大 M 时,准确率的提升趋于平缓,未因采样间隔不同而产生明显差异。在执行时间方面,随着 M 的增大,各采样间隔下的算法执行时间显著增加,这是由于Viterbi算法需要遍历计算更多轨迹点的观测状态以及转移状态。

基于所有间隔下算法性能的变化趋势,本文选定 $r=100$ m,$k=5$,$M=3$ 作为最优参数,后续实验以此参数为基准,以确保在合理的时间范围内实现较高的准确率。

为进一步剖析MSFM算法内部的计算瓶颈,本文统计了各模块在单日轨迹数据匹配过程中的时间占比,如表3所列,候选路段搜索模块占总耗时的52.3%,成为了主要的瓶颈。未来工作将重点探索如何加快候选路段搜索,以进一步提高整体效率。

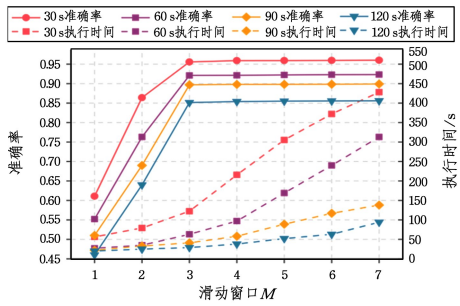


图6　滑动窗口尺寸对地图匹配性能的影响

Fig. 6　Influence of sliding window sizes on map matching performance

表3　各部分时间占比

Table 3　Time proportion of each part

执行部分	时间/s	占比/%
输入	2.57	1.6
广播	6.48	4.0
候选路段搜索	83.14	52.3
轨迹微分段	12.2	7.6
最优路径推断	54.62	34.5
总用时	159.01	100

接下来对比不同算法在不同采样间隔下的性能,测试了处理数据的执行时间以及准确率,所有实验均在统一的最短路径计算文件基础上进行,以确保结果的可比性。表4总结了MSFM,FMM与D-FMM在不同采样间隔下的准确率变化情况,以30 s采样间隔为例,MSFM的准确率相较于FMM降低了2.04%,这是由于FMM能够利用完整的轨迹段信息进行地图匹配,而MSFM仅能依赖微分段内的 M 个轨迹点的信息进行计算,使用了局部的轨迹信息,存在一定的准确率损失。从整体来看,随着采样间隔的增大,HMM模型求解时能够利用的轨迹连续移动信息减少,导致所有算法的准确率受到影响——这是未来前地图匹配研究中需要解决的关键问题。

表4　不同算法在不同采样间隔下的准确率变化

Table 4　Accuracy changes of different algorithms at different sampling intervals

(%)

方法	采样间隔/s			
	30	60	90	120
MSFM	95.86	92.11	89.70	85.14
FMM D-FMM	97.90	95.02	91.96	89.0

图7与表5进一步展示了各算法在不同采样间隔下的匹配速度。根据结果分析,MSFM在效率方面显著优于FMM,处理单日轨迹数据仅需122.40 s,也就是每秒处理能力接近11万条轨迹数据,相比FMM提升约7倍。此外,集群环境下的D-FMM相比单机环境的FMM在匹配速度方面提升近150%,也说明了分布式技术可以极大地提升HMM地图匹配的计算效率。而MSFM的执行行时间仅为D-FMM的1/3,更加表明MSFM充分利用了分布式集群的向量化计算能力。

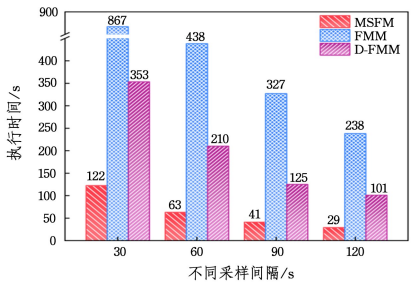


图 7 不同算法在不同采样间隔下的执行时间变化

Fig. 7 Variation of execution time of different algorithms at different sampling intervals

表 5 不同算法在不同采样间隔下的匹配速度变化

Table 5 Changes in matching speed of different algorithms at different sampling intervals

(GPS/s)

方法	采样间隔/s			
	30	60	90	120
MSFM	**108932.6**	**105820.2**	**108401.2**	**114942.7**
FMM	15378.7	15987.2	13591.5	14005.6
D-FMM	37771.5	31746.0	35555.6	33003.3

综合准确率和匹配速度分析，MSFM 通过引入轨迹微分段策略，在匹配准确率略有降低的情况下，实现了计算效率的大幅提升。这说明该策略在面对大规模轨迹数据实时处理需求时，通过有限精度损失换取更高吞吐量的权衡是可以接受的。

最后，通过调整集群计算节点的数量来评估 MSFM 的扩展性。图 8 展示了 MSFM 不同计算节点数对应的匹配速度。当节点从 3 增至 9 时，匹配速度从 11.1 万点/秒提升至 26.6 万点/秒，集群的匹配速度表现出稳定的增长，这表明 MSFM 算法充分利用了 Spark 框架的计算资源。

综合各项实验结果，MSFM 算法在大规模轨迹数据实时匹配中表现出较高的效率和较好的扩展性。尽管在准确率上略逊于 FMM，但这种折衷使得 MSFM 在实时性要求较高的应用场景中更具优势。

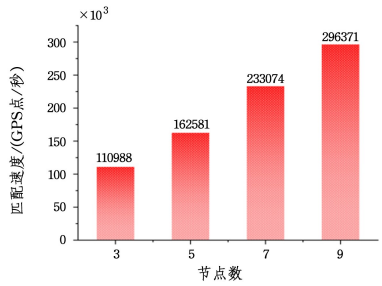


图 8 MSFM 在不同的节点所组成集群下的匹配速度

Fig. 8 Matching speed of MSFM in clusters composed of different nodes

结束语 本文提出了一种基于轨迹微分段模型的快速分布式地图匹配方法，通过滑动窗口机制将轨迹段划分为具有固定长度的微轨迹段，避免了传统 HMM 模型的时序依赖性，充分利用 Spark 框架的向量化技术，实现轨迹点级的分布式并行计算。实验表明，MSFM 在保持较高准确率的前提下，处理速度较 FMM 算法提升约 7 倍，验证了其在实时大规模数据处理中的优势。未来工作不仅要聚焦参数自适应优化和加快候选路段搜索，进一步提升方法的性能，同时要考虑较大采样间隔导致地图匹配准确率下降的问题。

参 考 文 献

[1] ZHENG Y, ZHOU X. Computing with spatial trajectories [M]. Springer Science & Business Media, 2011.

[2] LUO Q, HE S, HAN X, et al. LSTTN: A long-short term transformer-based spatiotemporal neural network for traffic flow forecasting [J]. Knowledge-Based Systems, 2024, 293: 111637.

[3] JUAN Z, ZHANG J, GAO M. A multimodal travel route recommendation system leveraging visual Transformers and self-attention mechanisms [J]. Frontiers in Neurorobotics, 2024, 18: 1439195.

[4] QUDDUS M A, OCHIENG W Y, NOLAND R B. Current map-matching algorithms for transport applications: State-of-the art and future research directions [J]. Transportation Research part C: Emerging Technologies, 2007, 15(5): 312-328.

[5] RABINER L, JUANG B. An introduction to hidden Markov models [J]. IEEE Assp Magazine, 1986, 3(1): 4-16.

[6] NEWSON P, KRUMM J. Hidden Markov map matching through noise and sparseness [C] // Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2009.

[7] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing [C] // Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012.

[8] YANG C, GIDOFALVI G. Fast map matching, an algorithm integrating hidden Markov model with precomputation [J]. International Journal of Geographical Information Science, 2018, 32(3): 547-70.

[9] GUTTMAN A. R-trees: A dynamic index structure for spatial searching [C] // Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data. 1984.

[10] FINKEL R A, BENTLEY J L. Quad trees a data structure for retrieval on composite keys [J]. Acta Knformatica, 1974, 4: 1-9.

[11] BALKIĆ Z, ŠOŠTARIĆ D, HORVAT G. GeoHash and UUID identifier for multi-agent systems [C] // Proceedings of the Agent and Multi-Agent Systems Technologies and Applications: 6th KES International Conference (KES-AMSTA 2012). 2012: 25-27.

[12] ISHIGURO T, SASAI T, FUKUSHIMA S, et al. Leveraging trajectory simplification for efficient map-matching on road network [C] // Proceedings of the 2024 25th IEEE International Conference on Mobile Data Management (MDM). 2024.

[13] KOLLER H, WIDHALM P, DRAGASCHNIG M, et al. Fast hidden Markov model map-matching for sparse and noisy trajectories [C] // Proceedings of the 2015 IEEE 18th International

Conference on Intelligent Transportation Systems. 2015.

[14] HART P E, NILSSON N J, RAPHAEL B. A formal basis for the heuristic determination of minimum cost paths [J]. IEEE Transactions on Systems Science and Cybernetics, 1968, 4(2): 100-107.

[15] GOLDBERG A V, HARRELSON C. Computing the shortest path: A search meets graph theory[C]// Proceedings of the SODA. 2005.

[16] DOGRAMADZI M, KHAN A. Accelerated map matching for GPS trajectories [J]. IEEE Transactions on Intelligent Transportation Systems, 2021, 23(5): 4593-4602.

[17] FU X, ZHANG J, ZHANG Y. An Online Map Matching Algorithm Based on Second-Order Hidden Markov Model [J]. Journal of Advanced Transportation, 2021, 2021(1): 9993860.

[18] HUANG J, QIAO S, YU H, et al. Parallel map matching on massive vehicle gps data using mapreduce[C]// Proceedings of the 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing. 2013.

[19] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113.

[20] ALVES P D, QUOC V N H, ZHENG B, et al. A framework for parallel map-matching at scale using Spark [J]. Distributed and Parallel Databases, 2019, 37: 697-720.

[21] YU J, WU J, SARWAT M. Geospark: A cluster computing framework for processing large-scale spatial data[C]// Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2015.

KANG Jun, born in 1975, Ph. D, associate professor, master's supervisor, is a member of CCF (No. 56262M). His main research interests include big data analysis, spatiotemporal trajectory data mining, and traffic behavior feature analysis and recognition.

（责任编辑：喻藜）