基于二级时空分桶的伴随轨迹查询

王晨旭[1),2)] 汪谨权[1)] 杨 鑫[1)]

[1)](西安交通大学软件学院 西安 710049)

[2)](西安交通大学智能网络与网络安全教育部重点实验室 西安 710049)

摘 要 随着移动传感器设备的普及,人们能够采集到的位置数据越来越多,轨迹数据的规模也越来越庞大.从大规模时空数据中查找与指定轨迹最相似的前 k 条轨迹一直是时空大数据挖掘的重要挑战之一.现有的相似轨迹查询方法大都包括三个阶段:(1)对海量的离线轨迹数据建立索引;(2)基于索引结构从已知轨迹集中查询与指定轨迹相似的候选轨迹;(3)计算指定轨迹与候选轨迹之间的精确相似度并返回相似度最大的前 k 条轨迹.但大多数现有方法对轨迹进行聚类索引时不能有效利用时间和空间信息,导致时间相似度不高的轨迹也会被划分到相同的索引项上,最终影响查询的准确性和效率.此外,现有的时空轨迹相似度计算方法存在大量的无效运算,使得相似轨迹的查询效率整体较低.针对当前伴随轨迹查询方法对时间与空间信息利用不充分的问题,本文提出一种新的二级时空分桶索引结构,首先将每条轨迹数据按照时间滑动窗口划分为若干带有时间槽信息的子轨迹,在时间上对轨迹进行一级索引聚类;在此基础上对在相同时间槽内的子轨迹进行二级空间索引聚类,利用哈希算法将具有连续相同位置点的子轨迹映射到同一时空分桶中.与已有索引方法相比,该方法对不同轨迹在索引时具有更好的区分度,查询时的筛选条件更为严格,有效降低了候选轨迹集的规模.针对现有轨迹相似度计算方法效率低下的问题,提出一种基于时差约束的轨迹相似度计算方法.利用轨迹之间的时差排除大量不必要的位置比较运算,将轨迹相似度的计算复杂度控制在线性级别,大大提高了计算效率,同时为过滤伴随轨迹查询过程中的无效计算,对基于时差约束的轨迹相似度计算方法进行变体得到一种上下界过滤方法,最大限度地避免了无效计算.最后,在 4 个真实的大规模轨迹数据集上对所提方法进行实验验证,实验结果表明所提方法的轨迹查找效率是已知最好方法的 9~20 倍,证明了算法的有效性.

关键词 二级时空索引;轨迹相似度计算;伴随轨迹查询

中图法分类号 TP392 **DOI 号** 10.11897/SP.J.1016.2024.00131

Accompanying Trajectory Query Based on Secondary Temporal and Spatial Indexing

WANG Chen-Xu[1),2)] WANG Jin-Quan[1)] YANG Xin[1)]

[1)](*School of Software Engineering, Xi'an Jiaotong University, Xi'an* 710049)

[2)](*Ministry of Education Key Lab of Intelligent Network and Network Security, Xi'an Jiaotong University, Xi'an* 710049)

Abstract With the proliferation of mobile sensor devices, an increasing amount of position data is being collected, and the scale of trajectory data also grows rapidly. It has always been a challenge to find the top-k trajectories most similar to a given query trajectory from massive spatial-temporal data. Existing trajectory-query methods comprises three main stages: (1) offline trajectory indexing on a massive dataset, (2) searching for candidate trajectories similar to the query trajectory based on the indexed structure, and (3) computing the precise similarity between the query trajectory and candidate trajectories, and returning the top-k most similar trajectories. However, most existing methods fail to effectively exploit the temporal and spatial information during trajectory clustering indexing, Trajectories with low temporal similarity are mistakenly partitioned into the

收稿日期:2022-12-01;在线发布日期:2023-07-12.本课题得到国家自然科学基金(62272379)、陕西省自然科学基金(2021JM-018)、国家重点研发计划(2021YFB1715600)、中央高校基本科研业务费(xzy012023068)资助.**王晨旭**(通信作者),博士,副教授,中国计算机学会(CCF)会员,主要研究方向为大数据挖掘、人工智能、网络安全等.E-mail:cxwang@mail.xjtu.edu.cn.**汪谨权**,硕士,主要研究方向为伴随轨迹查询、轨迹索引结构内存优化等.**杨 鑫**,硕士研究生,主要研究方向为海量轨迹处理、协同运动模式挖掘等.

same index, degrading query accuracy and efficiency. Furthermore, existing spatial-temporal trajectory similarity-calculation methods involve numerous unnecessary operations, resulting in low query efficiency. To address these issues, this paper proposes a novel secondary trajectory index structure. Firstly, a trajectory is divided into sub-trajectories by a sliding time window. Then, sub-trajectories are clustered for first-level indexing based on their time slots. Then, sub-trajectories within the same time slot are grouped into second-level spatial clusters. Sub-trajectories with continuous identical locations are partitioned into the same spatial-temporal bucket by a hash algorithm. Compared to existing indexing methods, this approach provides better distinction between different trajectories during indexing, resulting in more strict filtering conditions in the query process, reducing the size of the candidate trajectory set. To tackle the computational efficiency problem of existing trajectory similarity calculation methods, we propose a trajectory similarity calculation method based on the constraints of time difference, eliminating many unnecessary location comparisons and thus reducing the computational complexity of trajectory similarity calculation. Additionally, we devise a new variant of the similarity calculation approach based on upper and lower bound filtering. This scheme further reduces unnecessary computations during the search of accompanying trajectories. Finally, we conduct experiments to evaluate the effectiveness of the proposed method based on four large-scale real trajectory datasets. The experimental results demonstrate that the proposed method achieves 9 to 20 times faster trajectory retrieval efficiency than state-of-the-art approaches, confirming the effectiveness of the algorithms.

Keywords secondary indexes; trajectory similarity computation accompanying; accompanying trajectory query

1 引 言

随着位置采集设备的普及和在线地图技术的不断发展,移动位置数据的采集越来越普遍,时空轨迹大数据挖掘受到越来越多的关注[1]. 从海量数据中查找相似轨迹是时空轨迹大数据挖掘的重要基础研究之一,可以为众多应用提供技术支撑[2-3]. 伴随轨迹查询是从包含大量时空轨迹数据的集合中查找与指定轨迹在时间和空间上均相似的前 k 条轨迹,在交通规划[4-5]、服务推荐[6-7]、跨网站用户识别[8-10]、疫情防控[11]、治安管理[12-13]等多个领域具有重要应用. 例如,当已知一名病毒传播阳性患者的行程轨迹时,快速找到该患者的密切接触者并进行隔离管控是疫情防控的关键任务,在伴随轨迹查询任务较多时,算法的查找效率将极大地影响疫情朔源任务的时效性. 再如,当已知某犯罪团伙中一名或多名犯罪分子的行程轨迹时,通过伴随轨迹搜索能够快速地找到与其长时间共同行动的其他犯罪同伙,可以实现对罪犯的有效辨别和及时抓捕.

伴随轨迹查询问题的定义如下:给定查询轨迹 Q,在给定的时间和空间阈值范围内,从已知轨迹集合 $\mathcal{P}$ 中搜索与 Q 相似度最大的前 k 条轨迹,其中 Q 由用户输入,k 为可调参数. 图 1 为伴随轨迹搜索的

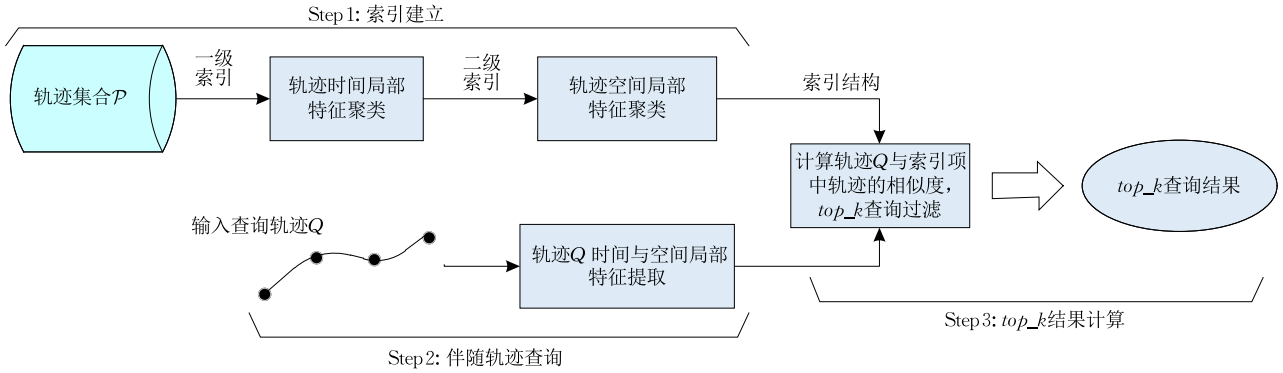


图 1 伴随轨迹查询的一般流程

一般流程，通常包括 3 个阶段：索引建立、候选集查询和伴随轨迹相似度计算，索引建立阶段利用轨迹的时间和空间局部特征对集合中的轨迹进行聚类，建立一个轨迹索引结构；在候选集查询阶段，通过输入轨迹 Q 的时间和空间局部特征从索引结构中快速查找与其较为相似的候选轨迹集；最后计算轨迹 Q 与候选集中各轨迹之间的相似度，并根据计算的相似度结果进行排序过滤得到前 k 个最相似轨迹作为查询结果输出.

现有的时空相似轨迹查询方法在建立索引时，往往会忽略轨迹的时间信息，或者仅仅对轨迹的开始时间与结束时间建立索引，导致不能在时间上对轨迹进行有效的聚类，使得一些在时间上没有关系或时间相似性较低的轨迹被映射到同一分桶中，极大地影响查询的效率和准确性. 此外，在轨迹的空间信息处理上，已有方法大多利用轨迹的单个位置点或无交叠的轨迹段来构建索引，前者导致轨迹搜索的候选集冗余，严重影响查询效率；而后者导致分段处的轨迹点之间的相关性信息丢失，影响查询的准确性. 且算法性能受轨迹片段的大小影响较大，当轨迹片段划分较小时将极大地降低查询效率，而当轨迹片段较大时将导致不能对轨迹进行有效聚类，导致查询准确率较低. 现有的大多数相似轨迹查找方法无法满足伴随轨迹查询的要求，主要因为当前大多数轨迹相似度计算方法未考虑轨迹之间的时间相似度，仅衡量轨迹在空间上的相似度[14-17]，导致查询结果不精确，不能满足疫情防控、拼车服务、罪犯抓捕等对时间相似性要求较高的需求.

近年来，已有不少工作同时利用时间和空间信息来计算轨迹相似性[18-20]，这些方法大多分别计算轨迹的时间相似度和空间相似度，然后利用加权组合的方式综合得到相似度结果，此类方法会产生轨迹不匹配的问题. 部分方法对该问题进行了改进[21]，将轨迹划分为长度为 2 的轨迹段，利用轨迹段和位置点之间的相似性计算轨迹的相似度，但该方法在计算过程中需要对轨迹中的位置点进行全量遍历，位置点之间存在大量无效计算.

针对以上问题，本文提出一种高效的伴随轨迹搜索方法(TSlot). TSlot 采用一种基于二级时空分桶的轨迹索引结构，首先将轨迹在时间上进行一级索引聚类，将每条轨迹数据按照时间滑动窗口划分为若干带有时间槽信息的子轨迹，在此基础上对同一时间槽内的子轨迹在空间上进行二级索引聚类，利用哈希算法将具有连续相同位置点的子轨迹映射到同一个时空分桶中. 该索引结构同时考虑了时间与空间信息，在查询时筛选条件更为严格，得到的候选轨迹集更加精准，查询效率更高. 与现有方法利用整条轨迹的时间段信息建立索引的方式不同，TSlot 利用滑动窗口机制，将轨迹划分为若干条子轨迹，子轨迹与子轨迹之间根据滑动步长的设定会包含一定数量的重复位置点，有效解决了轨迹分段处的信息丢失问题. 且 TSlot 允许灵活设置滑动窗口的宽度和滑动步长，与现有索引方法利用单个位置点或不包含重复位置点的轨迹片段进行空间索引的方式相比，TSlot 在进行空间聚类时轨迹之间的区分度更大，并可通过滑动窗口的宽度进行调节，宽度越大区分度越高，且不会丢失轨迹中连续位置点之间的关联信息. 在轨迹相似度计算阶段，本文提出了一种基于时差约束的空间轨迹相似度计算方法，通过位置点之间的时间相似度避免不必要的位置点比较计算，极大地提高了轨迹相似度的计算效率，且该方法在计算长轨迹之间的相似度时能保持稳定的计算效率和准确度.

最后，本文基于 4 个公开的真实数据对 TSlot 的有效性进行了实验验证，与当前最先进方法相比，不仅计算结果的准确性有较大的提升，计算效率达到已知最好算法的 9 到 20 倍.

本文的主要贡献包括：

(1) 提出了一种查询条件更为严格的二级时空索引结构，对候选结果的选取更为严格，在保证结果准确性的前提下有效降低了候选结果集的规模，极大地提高了伴随轨迹的查询效率.

(2) 提出了一种基于时差约束的空间轨迹相似度计算方法，该算法的时间计算复杂度为线性级别，使得轨迹相似度的计算效率有了量级的提高.

(3) 在多个公开的真实数据集上进行了大量的实验验证，有效证明了所提方法的准确性和时间效率.

2　相关工作

2.1　轨迹索引

轨迹索引是指对已采集的轨迹数据按照一定的规则进行聚类，从而在轨迹查询时通过索引结构有效降低搜索的范围. 高效的索引能够避免大量的轨迹间相似度计算，提高伴随轨迹的搜索效率. 现有的轨迹索引主要包括基于轨迹数据的索引和基于道路网络的索引. 基于轨迹数据的索引结构主要利用轨迹

的时间或空间信息对轨迹进行聚类，如 Sim_st[19-20]、FNR[22]、MON[23]等方法采用树形结构对轨迹的时间信息进行索引，把轨迹按照时间信息划分到相应的时间段中，但当轨迹数量庞大或轨迹之间的时间交叉较多时，会严重降低查询的效率. BFGPI[1]、SST[24]等方法基于轨迹的位置点构建索引，首先将位置点映射到网格中，然后对相同网格下的轨迹进行聚类，这类方法在建立索引时未考虑轨迹的时间信息，因此不适合伴随轨迹搜索这种对时间要求较高的需求场景. tSTAT[25]、DYFT[26]等方法利用局部敏感哈希(LSH)技术将轨迹映射为带有原始数据信息的向量，对向量进行切段后构建索引，CEAL[27]使用树结构利用子轨迹的空间信息对轨迹进行聚类，EPAA[28]基于子序列的方差和平均差提出子序列相似性的高效计算方法，这类方法在构建索引结构时也未利用轨迹的时间信息，且在切段时会丢失轨迹的部分空间信息. 基于道路网络的索引结构将十字路口、道路终点这类特殊空间位置抽象为节点，可通行道路由节点之间的边表示，从而构成道路网络，然后将轨迹的位置点映射到道路网络中，TEN-QueryIP[29]基于网络中的节点，查询与该节点距离小于指定阈值的节点，TL-Query[30]主要工作内容是快速计算两个节点之间的最短路径的数目，这类方法应用到伴随轨迹查询时都是基于单个位置点构建索引结构且忽略了时间上的约束，这些方法高度依赖道路网络的构建，且需要定期更新道路网络，无法应用于自由移动的对象.

与已有方法相比，本文提出了一种基于二级时空分桶的轨迹索引结构. 使用子轨迹的开始时间建立时间索引，解决了现有方法的索引结构模糊问题；利用带有重复信息的子轨迹建立空间索引，更加充分地利用轨迹的空间信息，提高索引结构的有效性.

2.2 轨迹相似度计算

轨迹相似度的高效计算是时空轨迹大数据挖掘的基础，现有轨迹相似度计算研究主要分为两个方向：传统的针对计算效率和结果准确率的算法优化和基于深度学习的框架. 传统意义上的轨迹相似度计算方法又可以分为两大类. 第一类仅关注轨迹在空间上的相似度，如 Frechet[31]、Hausdorff[21]和 DTW[32]，这些方法在计算相似度时只考虑轨迹之间的空间信息，无法满足疫情防控、治安管理等对时间要求较高的场景需求. 第二类方法在计算轨迹相似度时会同时考虑轨迹之间的时间相似度和空间相似度，代表性工作包括 Sim_st[18-20]、MPS[24]和 EDR[33]等算法，Sim_st[18-20]等算法分别计算轨迹的时间相似度与空间相似度，然后通过加权和将二者进行融合. MPS[24]将轨迹相似度缩小到轨迹段和轨迹点之间的相似性计算，通过计算轨迹点之间的相似度得到最终的轨迹相似度，有效结合了时间和空间信息. EDR[33]采用动态规划方法，允许一些位置点重复计算以实现最相似子轨迹之间的对齐，并通过匹配阈值的方式增强计算的鲁棒性. 但此类方法在计算相似度时阈值不易确定，甚至影响结果的准确率. 基于深度学习的方法通过训练将轨迹转化为向量表示，通过计算向量相似度得到一个高效的轨迹相似度计算方法，如 seq2seq[34]、Aries[35]、NEUTRAJ[36]等，然而这类基于深度学习的方法模型往往在训练时需要消耗大量的时间，且随着传统方法研究的不断进步，传统方式的轨迹相似度计算方法在计算效率和准确率方面仍有一定的优势.

与现有方法不同，本文提出的相似度计算方法采用时间优先策略，既有效地结合了时间相似度与空间相似度，又避免了轨迹位置点之间的全量计算，在保证有效性的前提下，使得计算效率得到了量级的提升.

3 二级时空分桶索引结构构建

3.1 问题描述

定义 1. 轨迹是对象在移动过程中产生的一系列有序的位置点，并按照位置点的时间先后顺序排列. 本文用 $T=(p_1,p_2,\cdots,p_{|T|})$ 表示一条轨迹，其中 $p_i=(time_i,lon_i,lat_i)$，且当 $i>j$ 时，$time_i>time_j$，$time_i$ 为对象在位置点 p_i 的采样时刻，由标准时间戳表示，lon_i 和 lat_i 分别为位置点的经度和纬度，$|T|$ 为轨迹 T 包含位置点的数量. 由于不同对象在移动速度和时长上存在差异，不同轨迹中位置点的数量 $|T|$ 不尽相同. 本文假设轨迹中的位置点 $(time,lon,lat)$ 是由位置采集装置按照固定的采样频率 θ 收集到的对象移动位置信息，这一假设在目前大多数的位置采集设备上均能实现. 对于 θ 不固定的轨迹数据，需要采用轨迹预测算法对漏采位置点进行补全后再采用本文算法进行伴随轨迹查询，对于这类轨迹数据的处理将在后续工作中展开研究.

定义 2. 给定查询轨迹 Q 和轨迹集合 $\mathcal{P}$，伴随轨迹的 top_k 查询是指从 $\mathcal{P}$ 中找出包含 k 条轨迹的结果集 $\mathcal{R}$，使得 $Sim(Q,P)>Sim(Q,P')$，$P\in\mathcal{R}$ &

$P' \in \mathcal{P}-\mathcal{R}$，其中 $Sim(Q,P)$ 表示轨迹 P 和轨迹 Q 之间的相似度，其具体计算方式见 4.2 节.

当从海量轨迹集合中查找与指定轨迹相似度最大的前 k 个轨迹时，计算指定轨迹与所有轨迹之间的相似度计算效率较低，为实现相似轨迹的快速查询，可以对集合中的轨迹数据按照一定的规则建立索引，将相似度大于一定阈值的轨迹划分到同一个分桶中，在查询时先按照相同的规则查询轨迹所对应的分桶（此时分桶中的所有轨迹与查询轨迹的相似度均大于设定的阈值），仅需计算待查轨迹与分桶中候选轨迹之间的相似度然后排序即可，从而实现对轨迹数据的快速查询. 索引的构建直接影响查询的效率和准确性. 本文提出一种二级时空分桶的索引结构，同时考虑了轨迹的时间和空间信息，在轨迹查询时筛选条件更为严格，分桶中的候选轨迹数量相对较少，因此查询效率更高.

3.2　数据预处理

首先对轨迹数据进行预处理. 为了方便建立时间索引，本文以天为周期对轨迹数据进行处理. 将一天平均划分为 TN 个时长相同的时间片，每个片段称为一个时间槽，通过时间槽对轨迹建立时间索引. 时间槽跨度 ST 表示时间片长度，例如当 $ST=5\,\text{min}$ 时，得到的时间槽个数为 $TN=\frac{24*60\,\text{min}}{ST}=288$ 个，即一天被划分为 288 个时间槽. 时间槽长度可以根据采样频率和查询精度的需要来设定. 此外，本文算法也可以方便地以周或其他时间周期为单位对轨迹数据进行处理.

为了方便建立空间索引，本文将原始的二维经纬度位置点压缩为一维表示. 将地图划分为一系列大小相同的正方形，对每个正方形赋予唯一的字符编号，同一个正方形内的位置点由该编号唯一表示，正方形的边长可以灵活设定，也称网格压缩精度. 图 2 为对原始轨迹处理过程的示意图，首先地图被划分为 6×7 的网格，间槽跨度为 $ST=1$ 小时，对 T_1 到 T_4 等 4 条原始轨迹中的位置点处理后可以得到如图中所示的四条轨迹表示，并对轨迹中具有伴随关系的子轨迹使用相同颜色进行标识.

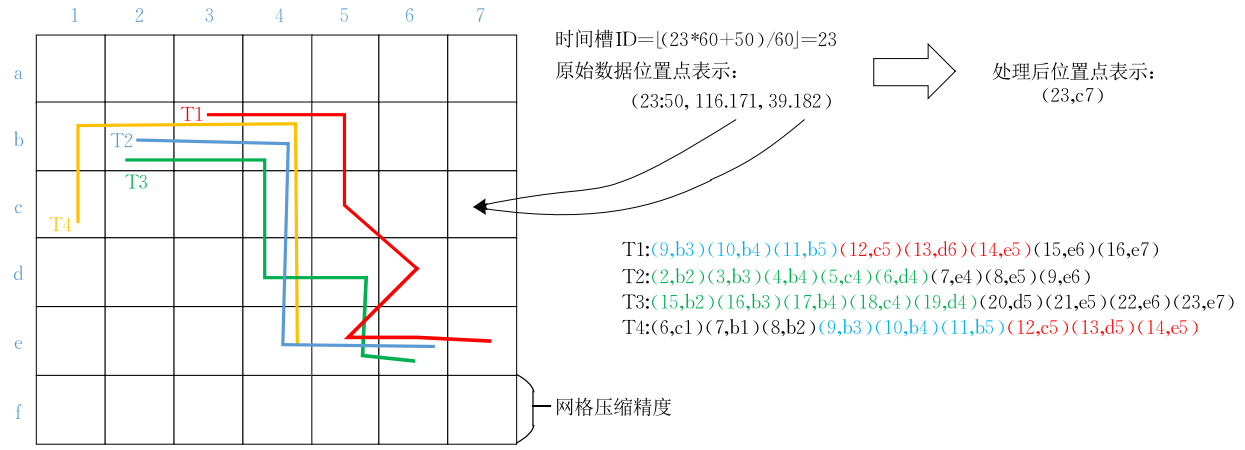


图 2　轨迹数据预处理示意图

3.3　子轨迹索引

在建立轨迹索引时，利用轨迹的局部特征映射整体特征是一种简单且高效的做法，但如何提取轨迹数据的局部特征一直是个难题. 与现有方法使用单个位置点、道路网络中的边、独立的轨迹段等方式不同，本文采用滑动窗口机制，解决了单个位置点和道路网络中边的局部特征不明显的问题，同时避免了轨迹段方法存在的空间关联信息丢失问题.

给定一条预处理后的轨迹 $T=(p_1,p_2,\cdots,p_{|T|})$，其中 $|T|\leq TN$ 为轨迹序列的长度，设滑动窗口的宽度为 w. 本文提出基于滑动窗口的轨迹索引结构的初衷是为了避免轨迹数据的丢失，因此本文将滑动步长设置为 1，即一个滑动窗口包含了 w 个位置点数据. 由此可知，对轨迹 T 处理后可以得到 $|T|-w+1$ 个长度为 w 的子轨迹，设每个子轨迹起始位置点的时间为 s，则每条子轨迹表示了对象在 $(s,s+w*ST)$ 时间范围内的移动轨迹，子轨迹对应的时间槽为 $g=\frac{s}{ST}$，将该子轨迹及其对应的完整原始轨迹 ID 划入相应的时间槽中，从而构建一级时间索引.

在此基础上，将同一个时间槽内的子轨迹继续在空间上进行聚类建立二级空间分桶索引，分桶标识由子轨迹序列的字符串 $subTra$ 的哈希值表示，分桶中存放原始轨迹对应的 ID. 子轨迹序列的字符串 $subTra$ 为轨迹中位置序列对应网格的字符编码拼接得到. 分桶哈希值的计算公式为

$$hash(subTra)=\sum_{i=0}^{n-1}s[i]*31^{n-1-i},$$

其中 n 为子轨迹字符串 $subTra$ 的长度，$s[i]$为子轨迹 $subTra$ 的第 i 个字符编码.

由此可知，同一个二级空间分桶中的所有轨迹至少有部分子轨迹是完全相同的，因此，在轨迹查询时需要将分桶中的所有轨迹加入到候选结果集中进一步筛选. 当时间槽的跨度大于子轨迹的持续时间 $w*\theta$ 时(θ 为位置采样率)，同一轨迹 ID 的多条子轨迹会出现同一个时间槽中，则该轨迹将同时出现在同一个时间槽下多个二级空间分桶中. 通过设置参数 w 的值可以调节索引结构的聚类效果，w 的长度决定了每个二级空间分桶中的轨迹最短伴随的时长. 与使用单个位置点进行索引相比，本文方法过滤掉了仅有位置点相交而无伴随的轨迹，同时使用滑动窗口避免了由于轨迹段切分导致的轨迹信息丢失问题.

算法 1 为建立轨迹索引的具体过程. 向量 $\boldsymbol{L}$ 表示时间槽，每个时间槽内存放若干个二级空间分桶. 主要过程如下：对数据集中每一条轨迹，定义起始下标 $i=0$ 表示轨迹中当前位置点的位置，根据起始和结束位置利用滑动窗口获取窗口内的子轨迹(行 2). 对所有的子轨迹，首先获取子轨迹的开始时间和滑动窗口范围内的移动位置信息(行 3、行 4)；然后根据子轨迹的开始时间找到对应的时间槽(行 5)，在时间槽中匹配二级空间分桶，如果匹配则将轨迹 ID 存放到二级空间分桶中，否则将创建一个新的分桶，然后将轨迹 ID 存放到该分桶中(行 6 到行 10)，窗口向后滑动(行 11)，完成索引结构的创建(行 12).

算法 1. 轨迹索引建立.

输入：轨迹集合$\mathcal{P}$

输出：轨迹索引结构 $\boldsymbol{L}$ //$\boldsymbol{L}$ 为 hashmap 的数组

1. Initialize $\boldsymbol{L}[j]\leftarrow Map\langle subTras, Set\langle T_{Id}\rangle\rangle$//$j=1,\cdots,TN$
2. for $T\in\mathcal{P}$ do $subTras\leftarrow GetSubTras(T,i,i+w)$
3. index=start time slot of $subTras$
4. $h=hash(subTras)$
5. $S=\boldsymbol{L}$[index]
6. if $h\notin S.Keys$ then
7. Initialize Set $C\leftarrow\varnothing$
8. S.add$\langle h,C\rangle$
9. end if
10. S.get(h).add(T_{Id})
11. i++
12. end for

图 3 为对图 2 中的轨迹构建的索引结构示意图. 以轨迹 T_1 为例，当 $ST=1$h，滑动窗口长度 $w=3$ 时，轨迹 T_1 被滑动窗口划分为 6 条子轨迹，每条子轨迹被映射到不同的时间槽中，用子轨迹的哈希值作为 Key 值建立二级空间分桶，该分桶保存了在同一时间槽且有相同子轨迹的所有原始轨迹的 ID，完成轨迹的聚类索引.

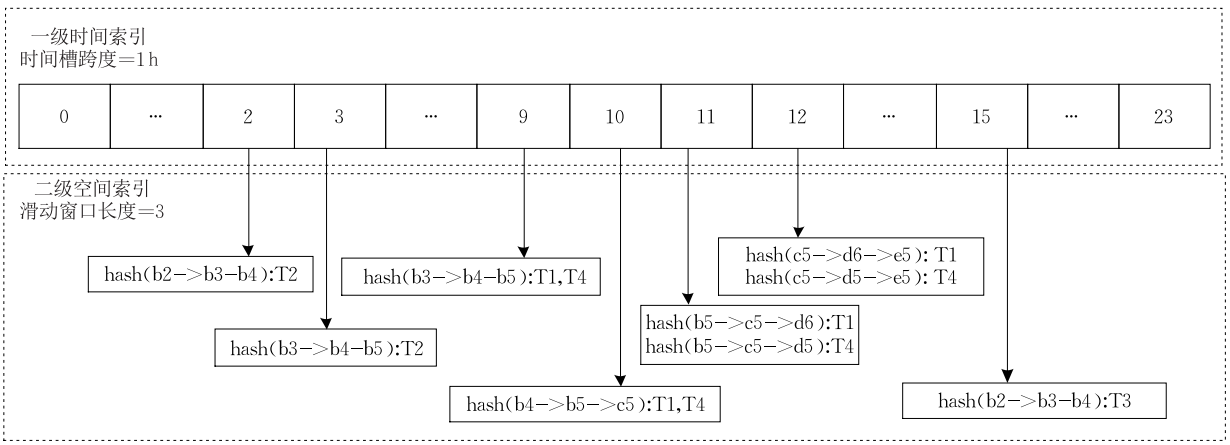


图 3 索引结构示意图

4 伴随轨迹查询

4.1 候选结果集获取

给定查询轨迹 Q，首先对 Q 进行预处理，将其映射到地图网格中，然后划分为多个带有时间戳的子轨迹，分别查找每个子轨迹对应的时空分桶，共有 $N=(|Q|-w+1)$ 个分桶，设第 i 个分桶中的轨迹集合为$\mathcal{B}_i$，则候选结果集为

$$\mathcal{C}_Q=\bigcup_{i=1}^{N}\mathcal{B}_i \tag{1}$$

其中$\mathcal{C}_Q$为候选结果集.

4.2 基于时差约束的空间轨迹相似度计算

给定轨迹 Q 和候选集中的任一条轨迹 $P\in\mathcal{C}_Q$，首先将轨迹 P 和 Q 中的所有位置点按照时间划分到对应的时间槽中. 当属于不同轨迹的两个位置点被划分到同一时间槽或相邻时间槽时，认为这两个位置点互为邻居. 对于 Q 中的每个位置点，获取其在 P 中的所有邻居位置点，记该集合为 $\mathcal{N}_i$，则点与轨迹 P 的相似度可由式(2)计算得到：

$$subS_i=\begin{cases}\max\limits_{p_j\in\mathcal{N}_i}\{\lambda(e^{-dist\_s(q_i,p_j)}+1)+(1-\lambda)(e^{-dist\_t(q_i,p_j)}+1)\}, & |\mathcal{N}_i|>0\\ 0, & \text{其他}\end{cases}\tag{2}$$

$$dist\_s(q_i,p_j)=\begin{cases}dist(q_i,p_j), & |q_i.time-p_j.time|\le\varepsilon \text{ and } dist(q_i,p_j)\le\eta\\ 0 & \text{其他}\end{cases}\tag{3}$$

$$dist\_t(q_i,p_j)=\begin{cases}|q_i.time-p_j.time|, & |q_i.time-p_j.time|\le\varepsilon\\ 0, & \text{其他}\end{cases}\tag{4}$$

当 $|q_i.time-p_j.time|\le\varepsilon$ 时，认为两个对象在位置点 q_i 和 p_j 在时间上是伴随的. λ 表示计算位置点相似度时位置点空间相似度的重要性，即 $1-\lambda$ 表示时间相似度的重要性，本文中 λ 值默认为 0.5. $dist(q_i,p_j)$ 为位置点在空间中的距离，本文采用欧式距离计算两个经纬度坐标点之间的距离，η 为空间上的阈值，ε 为时间上的阈值，由用户按照需求和位置点的采集频率自由设定. 例如，当用户认为在同一个位置出现的时差不超过 60 s 时就认为在时间上是伴随的，即可将 ε 设置为 60 s. 同理，当用户认为两个位置点之间的距离不大于 600 m 时即在空间上存在伴随关系，可将 η 设置为 600 m.

对轨迹 Q 中所有位置点的相似度求和得到 $sim\_st\_q=\sum\limits_{q_i\in Q}subS_i$. 同理可以计算得到轨迹 P 中所有位置点与 Q 的相似度 sim_st_p，则轨迹 P 和 Q 之间的相似度 $Sim(P,Q)$ 由如下方式计算得到：

$$Sim(P,Q)=\frac{1}{2}\left(\frac{sim\_st\_p}{|P|}+\frac{sim\_st\_q}{|Q|}\right)\tag{5}$$

其中 $|P|$ 为轨迹 P 长度，$|Q|$ 为轨 Q 的长度. 在查询伴随关系时，只有当两个位置点在时间上满足一定阈值条件时，才产生伴随关系，此时才有计算两个位置点空间相似度的必要. 本文所提基于时差约束的轨迹相似度计算方法利用时间伴随条件对位置点进行初步筛选，将轨迹中的位置点划分到对应的时间槽中，如此，在计算轨迹 Q 中的点 q_i 与轨迹 P 的相似度时，无需遍历 P 中的所有位置点，大大降低了计算时间复杂度.

4.3 结果集过滤

在基于时差约束的空间轨迹相似度计算方法的基础上，本文还设计了一种基于上下界的过滤方法(udf)进一步加速轨迹相似度的计算. 根据式(2)可知，无须计算 q_i 和 p_j 之间实际的空间和时间距离，即可直接计算它们之间相似度的上界 Sim_{up}：

$$subS_{\text{up}}=\begin{cases}(e^0+1)*\lambda+(e^0+1)*(1-\lambda), & \mathcal{N}_i\text{不为空}\\ 0, & \text{其他}\end{cases}\tag{6}$$

$$Sim_{\text{up}}(P,Q)=\frac{1}{2}\left(\frac{count_P}{|P|}subS_{\text{up}}+\frac{count_Q}{|Q|}subS_{\text{up}}\right)\tag{7}$$

式(7)中 $count_P$ 和 $count_Q$ 分别为轨迹 P 和 Q 中最近邻集合 N_i 不为空的位置点的数量. 式(6)表示两个位置点的相似度为 100%，在时间上与空间上都十分契合时的相似度. 与式(2)相比，式(6)无须计算两个位置点之间的相似性，可直接得到两条轨迹的相似度上界. 在实际搜索过程中，可以记录已查找到的轨迹与目标轨迹相似度的最小值，对于相似度上界小于该最小值的轨迹可以直接跳过，大大减少了不必要的运算.

对于获取到的候选结果集，将候选结果用一个堆数据结构 $\mathcal{H}$ 来维护，堆中的轨迹按照与查询轨迹 Q 之间的相似度大小进行排序且限定堆中候选结果的数量为 k，同时维护一个最小相似度 $S_{\min}$ 表示堆中轨迹与 Q 的相似度最小值. 在对候选轨迹进行相似度排序过程中，当堆中轨迹的数量小于 k 时，计算轨迹 Q 与当前候选轨迹 P 的相似度并将 P 加入堆中，用 $S_{\min}$ 表示堆中所有轨迹与轨迹 Q 的相似度的最小值. 当堆中轨迹数量为 k 时，首先计算轨迹 P 和 Q 之间的相似度上界 $S_{\text{up}}(P,Q)$，如果 $S_{\text{up}}(P,Q)<S_{\min}$，则表示轨迹 P 与 Q 之间的相似度肯定不满足要求，可以安全地将轨迹 P 抛弃. 反之，则计算轨迹 P 与 Q 之间的实际相似度，并将轨迹 P 插入到堆中，同时删除堆中相似度最小的轨迹并更新 $S_{\min}$ 值. 最后堆中的 k 条轨迹即为伴随轨迹的查询结果.

4.4 计算时间复杂度分析

为了计算轨迹之间的时空相似度，TSlot 首先将两条轨迹 P 和 Q 中的位置点划分到时间槽中，其

时间复杂度为 $O(|P|+|Q|)$. 划分到时间槽后,分别遍历轨迹 P 和 Q 中的位置点,并获取相应的邻居位置点,计算时间复杂度与邻居位置点的数量密切相关. 对于候选轨迹 P,最坏的情况是每个位置 p_i 的邻居位置点包含了轨迹 Q 中所有的位置点,其计算复杂度为 $O(|P||Q|)$,此时基于时差约束的空间相似度的计算时间复杂度为 $O(2|P||Q|)$. 假设对于每个位置点 p_i,其邻居位置点的平均数量为 h,计算轨迹 P 到 Q 的时间复杂度为 $O(|P|h)$,此时位置点之间相似度的时间复杂度为 $O(|P|h+|Q|h)$,h 的大小与设定的时间阈值相关,h 为 1 时时间复杂度最低. 综合位置点划分到时间槽的时间复杂度可知,基于时差约束的空间相似度计算时间复杂度最坏情况下为 $O(2|P||Q|+|P|+|Q|)$,最好情况下为 $O(2(|P|+|Q|))$. Sim_st、EwDP 等方法的时间复杂度均为 $O(|P||Q|)$,且 MPS 的时间复杂度为 $O(2|P||Q|)$,且经过实验验证,本文提出的轨迹相似度的计算效率最高,实验结果见第 5 节.

4.5 索引结构复杂度分析

给定轨迹数据集 $\mathcal{P}$,对于集合中的每一条轨迹 T,使用长度为 w 的滑动窗口将其分为 $|T|-w+1$ 条子轨迹,导致每条子轨迹中的位置点在内存中保存了 w 次. 因此索引结构的空间复杂度为 $O(w|\mathcal{P}||T|)$. 而基于位置点和轨迹段切分方法的索引结构的空间复杂度为 $O(|\mathcal{P}||T|)$,因此本文提到的索引结构在内存空间上不占优势,内存优化也是未来后续工作的目标.

5 实验分析

5.1 实验数据

本文在四个公开真实轨迹数据集 DiDi、Proto、NewYork、Geolife 上对所提方法的有效性和计算效率进行评估. DiDi 数据集是滴滴出行发布的中国陕西西安地区用户在滴滴出行平台的订单出行信息,数据包含了 2016 年 11 月内的订单行程信息,位置点采样间隔时间为 3 到 5 s. 每条轨迹为一个完整订单,用户信息均经过匿名化处理. Proto 数据集为在葡萄牙波尔图市运行的所有 442 辆出租车一年的轨迹(从 2013 年 7 月 1 日至 2014 年 6 月 30 日),New York 数据集为长时间跟踪 10354 辆出租车所得的出行轨迹,New York 数据集中的轨迹持续时间与 Proto 相比更长,位置点的采样间隔时间也更长. Geolife 数据集由 182 名用户在五年多的时间内在 Geolife 项目中收集,包含了用户的户外活动,包括回家、上班等日常活动和购物、观光、骑自行车等娱乐活动轨迹数据. 相关数据集的统计信息如表 1 所示.

表 1 实验数据集

数据集	轨迹数量	移动对象数量	采样率/(s/次)	最小长度	最大长度	平均长度	平均持续时长/s
DiDi	1.2M	/	3~5	6	8481	931	3724
Proto	1.7M	442	15	6	3881	50	750
NewYork	10052	10052	49	7	154688	1757	86093
Geolife	18669	182	1~5	2	92644	1332	9675

本文算法由基于 jdk1.8 的 Java 代码实现,运行环境为安装了 Ubuntu 18.04 操作系统、Intel(R) Xeon(R) Gold 6266C CPU@3.00GHz、1024GB 内存的服务器. 本文算法所采用的默认参数如表 2 所示.

表 2 实验参数设置表

符号	参数描述	默认值
ST	时间槽跨度	3 倍采样率
w	滑动窗口宽度	2
d	网格压缩精度	200 m
k	top_k 查询时返回结果数量	20
ε	轨迹相似度计算时间阈值	300 s
η	轨迹相似度计算空间阈值	400 m

5.2 轨迹相似度计算方法的有效性验证

为了验证 TSlot 在轨迹相似度计算方面的有效性和时间效率,与 Sim_st[18-20]、MPS[21]、DTW[32]、EDwP[16]等 4 种算法进行对比. MPS 将包含 n 个位置点的轨迹划分为 $n-1$ 个轨迹段,利用位置点的时间信息和空间信息计算轨迹段之间的相似性,进而衡量两个轨迹之间的相似度. Sim_st[18]分别计算轨迹之间的时间相似度和空间相似度后,使用加权和将二者结合起来作为时空相似度. EDwP[16]和 DTW[32]都使用动态规划方法利用位置点的序列信息来计算轨迹之间的相似度. Aries[35]和 NEUTRAJ[36]方法均为基于深度学习的方法,通过设定其损失函数将轨迹转化为向量,通过学习到的模型计算向量之间的相似度表示轨迹之间的相似度,由于基于深度学习的方法在计算相似度之前需要训练得到模型,因此本文实验中只对比有效率,在验证相似度计算效率时不与基于深度学习的方法进行对比. MPS

方法利用轨迹段相似度来衡量轨迹之间的相似度，计算轨迹段相似度时同时考虑起点和终端的时间和空间相似度，用加权和来计算综合相似性，本文中采用 MPS 原文中的设置，权值为 0.5. Sim_st 方法分别计算轨迹的时间相似度和空间相似度[18]，用加权和来计算综合相似性，本文同样采用原文中设置，权值为 0.5. DTW 和 EwDP 方法采用动态规划的方法，无需考虑超参问题. 本文中 TSlot 方法中的超参 λ 设置为 0.5. 所有超参均表示时间相似度和空间相似度在进行计算时重要程度相同.

基于深度学习的轨迹相似性计算方法在实验时往往用传统的相似度计算方法为每条轨迹得到最相似的 k 条伴随轨迹，然后验证测试集轨迹是否包含在这 k 条伴随轨迹中，因此用 $Hitting\ Ratio@k$，$Recall\ k@R$ 做为衡量指标. 然而传统方法计算轨迹相似度方法，对于每条轨迹往往难以获得其真实的前 k 条相似度最高的轨迹，因此有效性指标往往使用命中率 $Hitting\ Ratio@1$ 做为衡量指标. 实验时，从轨迹数据集中随机挑选 10000 条长度大于 100 的轨迹数据集记做，然后从中随机抽取 1000 条轨迹，按照指定的采样概率抽取轨迹中的位置点，分别获取这 1000 条轨迹的子采样轨迹集合 $\mathcal{G}'$，即对于 $\mathcal{G}$ 中的每条轨迹，子采样轨迹为随机地抽取一定比例的位置点构成的一条新轨迹，并认为中的原始轨迹与其子采样轨迹相似度最高，例如当采样概率为 0.05 时，子采样轨迹的长度与原始轨迹长度之比为 0.05.

在计算命中率时，对于 $T\in\mathcal{G}'$，当

$$Sim(T,Q)>\underset{Q'\in\mathcal{G}\&Q'\neq Q}{Sim(T,Q')}$$

(T 为 Q 的子采样轨迹)时，认为命中，命中率定义为

$$hitR=\frac{hitCount}{traCount},$$

式中，$hitCount$ 为命中的次数，$traCount$ 为测试轨迹的数量，即 1000. 以上过程重复 10 次，以 10 次实验结果的平均值作为最终结果.

表 3 所示的实验结果表明，本文提出的相似度计算方法在 4 个数据集上均取得了最好的结果，其中采样概率表示采样后轨迹位置点数量与采样前数量之比，衡量指标为命中率. 当轨迹位置点的采样频率越高时，轨迹相似度的命中率越高，这是因为采样频率越高，数据所表示的轨迹越详细，位置点之间

表 3　各方法轨迹查询命中率结果　　(单位：%)

方法		采样概率							
		0.39	0.01	0.03	0.05	0.07	0.15	0.23	0.31
Proto	TSlot	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	MPS	89.6	93.1	98.3	99.7	**100.0**	**100.0**	**100.0**	**100.0**
	Sim_st	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	DTW	23.8	25.6	29.4	68.0	91.6	96.4	99.6	99.7
	EwDP	26.7	38.5	49.3	79.6	89.5	98.1	**100.0**	**100.0**
	NEUTRAJ	21.8	30.2	45.2	66.8	74.1	88.8	94.9	98.1
	Aries	25.4	34.9	47.5	73.1	82.5	93.8	97.8	99.8
New York	TSlot	**97.5**	**98.3**	**98.9**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	MPS	87.6	89.1	94.3	96.7	99.4	**100.0**	**100.0**	**100.0**
	Sim_st	93.6	94.5	95.8	97.9	98.6	99.1	**100.0**	**100.0**
	DTW	20.7	25.0	26.4	59.7	86.6	91.8	96.3	99.1
	EwDP	14.9	23.5	46.2	59.7	79.4	91.3	94.6	98.5
	NEUTRAJ	9.6	14.8	28.3	39.9	52.7	79.3	82.3	88.5
	Aries	12.6	19.7	35.9	46.3	67.3	86.8	89.2	93.5
DiDi	TSlot	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	MPS	99.5	99.8	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	Sim_st	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	DTW	29.8	46.1	66.3	79.5	96.4	98.5	**100.0**	**100.0**
	EwDP	35.8	49.6	71.6	84.4	97.3	99.1	**100.0**	**100.0**
	NEUTRAJ	30.1	35.8	52.5	76.9	80.4	82.2	89.0	91.5
	Aries	33.9	47.8	69.3	81.2	89.6	94.5	97.2	99.0
Geolife	TSlot	87.1	94.4	96.1	99.3	**100.0**	**100.0**	**100.0**	**100.0**
	MPS	78.3	84.3	87.3	90.5	94.1	95.9	98.6	**100.0**
	Sim_st	82.4	85.4	89.8	92.3	96.8	99.3	**100.0**	**100.0**
	DTW	31.2	35.3	37.2	49.1	64.8	71.5	78.9	83.4
	EwDP	42.8	45.4	53.2	69.7	75.5	86.4	89.2	93.8
	NEUTRAJ	30.2	34.8	29.8	41.6	59.9	63.6	78.8	84.1
	Aries	38.5	41.5	49.2	58.4	67.4	75.4	87.8	90.8

出现误差的概率越小，计算方法的命中率也随之升高. Sim_st[18]方法暴力地对每个位置点进行匹配，但是在匹配的时候分离了时间与空间相似度，当位置点的采样率较低时计算结果容易产生较大误差. MPS[21]方法虽然将时间与空间相似度进行了有效结合，但是在计算过程中利用轨迹段之间的相似度来计算轨迹相似度，与位置点之间的相似度相比不够精确. DTW[32]和 EDwP[16]在计算时没有利用位置点之间的时间信息，而是对轨迹中的位置点的前后顺序做了一个简单的约束，因此效果较差. Arie[35]和 NEUTRAJ[36]方法通过深度学习的方法训练出一个与已有相似度计算方法具有相同效果的模型. 表 3 的结果表明深度模型的有效性往往低于已有的相似度计算方法，由于基于深度学习的相似度计算方法是通过训练学习一个与已有的相似度计算方法相似的模型，学习到的模型与已有的相似度方法相比在有效性方面往往无法取得优势. NEUTRAJ 和 Aries 训练时基于 EwDP 相似度计算方法.

为了分析算法的计算效率，随机抽取 100 条轨迹，并与 1000 条轨迹进行相似度计算，针对不同长度的轨迹，统计所有轨迹相似度的计算耗时，结果如图 4 所示. 实验结果表明，现有轨迹相似度计算方法的计算效率都与轨迹的长度密切相关，短轨迹与长轨迹之间的计算时间差别巨大，而本文提出的方法计算效率稳定，在轨迹长度变大的情况下计算时长增加并不明显，计算效率最高，比其他方法能够提升 2～703 倍.

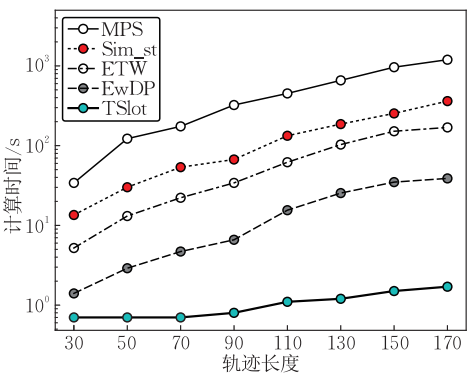


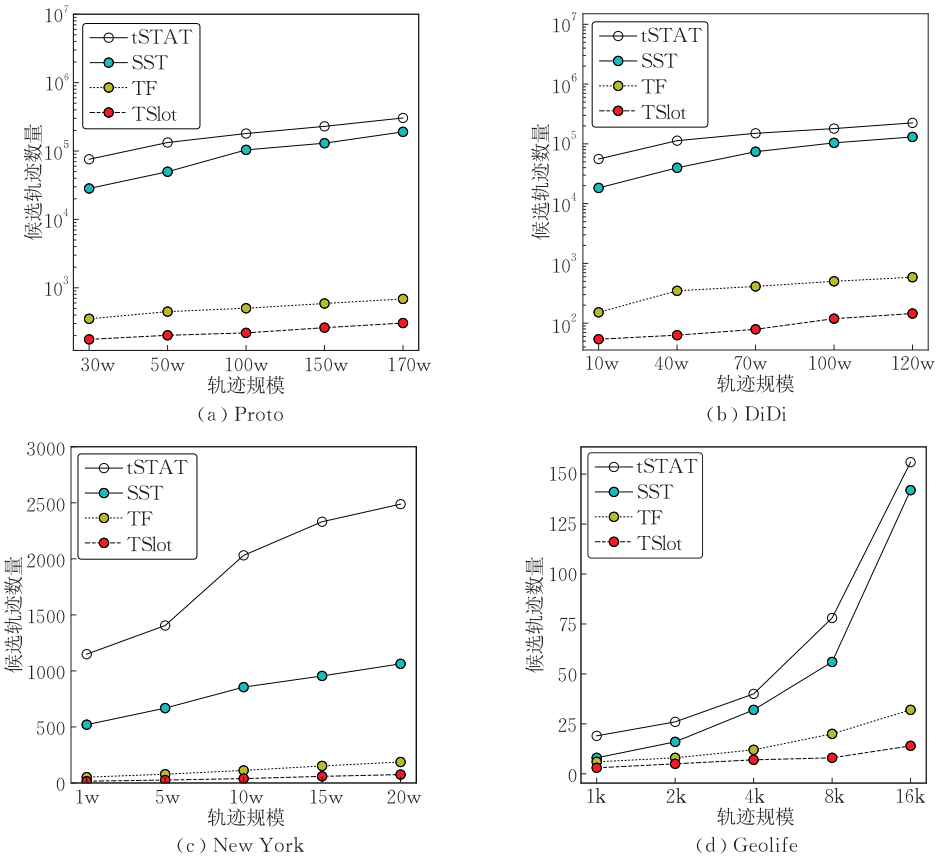
图 4 各方法计算效率对比

5.3 轨迹索引结构

构建轨迹索引结构的目的是将跟子轨迹具有伴随关系的轨迹聚类到一起，在进行伴随轨迹查询时通过相同的索引方法从索引结构获取到候选结果集合，因此通过查询时间来衡量轨迹索引结构，通过命中率来衡量轨迹相似度计算方法的有效性. 为了验证建立时间索引的必要性，将 TSlot 与没有时间约束的索引结构 SST[24]、tSTAT[25]方法和有时间索引的 TF[37]方法进行实验对比. SST 使用网格映射方法，将轨迹的单个位置点作为索引项，使得位置点相似的轨迹都保存在同一个索引项中. tSTAT 将轨迹划分为相同长度的若干个轨迹段，用轨迹段作为索引项. TF 利用轨迹的整个发生时间段建立一级时间索引，对每个一级时间索引项下的轨迹使用每个位置的时间信息建立二级时间索引，然后使用与 SST 相同的基于网格的三级空间索引. SST 方法和 tSTAT 方法均将轨迹位置点的二维经纬度采用网格的方式进行压缩，本实验将网格精度设置为 200*200. 与 SST 方法采用单个位置点的方式不同，tSTAT 将轨迹划分为若干长度为 h 的轨迹段，本实验设置 h 为 2. 对于 TSlot 方法，设置滑动窗口宽度为 2，时间槽跨度设置为采样率的三倍，如 Proto 数据集，时间槽跨度设置为 45 s. 对于 TF 方法中的超参 wt，λ，k_0，w，本文分别设置为 0.5，0.5，120，45.

本文采用查询所得的候选集中轨迹的数量来对比不同索引方法之间的差异，结果如图 5 所示. 从图中可以看出，虽然 tSTAT、SST 方法所得的候选轨迹数量较于原始数据规模相比压缩到了十分一左右，但在加入时间约束后在 Proto 数据集上候选轨迹的数量降低了 270～1500 倍. SST 方法利用轨迹的单个位置点进行空间索引，不考虑时间信息，两条轨迹只要有一个位置点位于同一个索引下，在搜索时就会被加入到候选集，索引结构对轨迹的区分度较低. 类似地，tSTAT 也未考虑时间信息，与 SST 不同的是其使用轨迹段信息来构建空间索引，对轨迹的区分度比 SST 高，但索引效果与 TSlot 和 TF 相比依旧存在较大差距. TF 构建索引时虽然利用轨迹的时间信息建立了两层索引结构，但是一级时间索引的边界较为模糊，二级时间索引和三级空间索引又都是基于轨迹的位置点信息，对轨迹信息利用不充分，查询时候选集中会包含时间和空间相似度较低的轨迹. 而 TSlot 利用子轨迹的开始时间建立一级时间索引，对时间相似度进行了更细粒度的过滤，在二级空间索引时利用子轨迹建立索引，子轨迹的长度天然地包含了时间约束信息，使得相似度较低的轨迹不会出现在同一个索引项中.

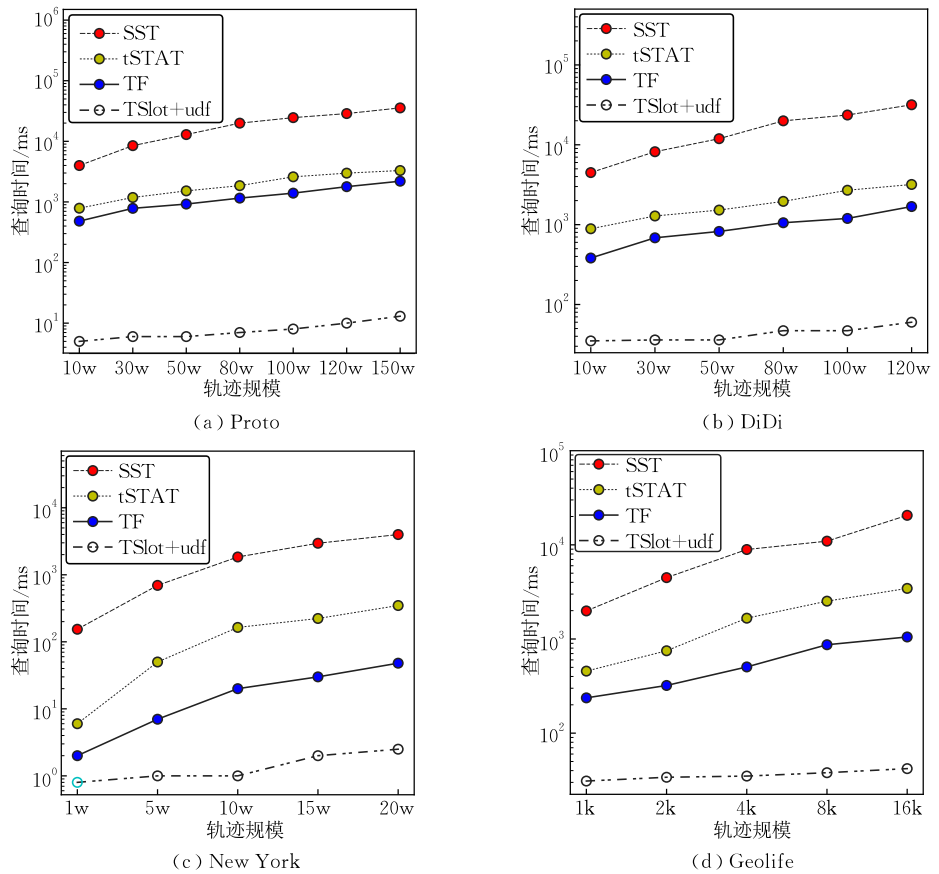
为了进一步验证本文所提索引结构的有效性，与 TF 方法进行了 *top*_20 查询时间的实验对比，以验正 TSlot 的查询效率. 图 6 所示的实验结果表明，



(a) Proto　(b) DiDi

(c) New York　(d) Geolife

图 5　候选结果对比



(a) Proto　(b) DiDi

(c) New York　(d) Geolife

图 6　伴随轨迹查询效率对比

TSlot 的查询性能与 TF 相比提高了 9～20 倍. SST 与 tSTAT 方法由于在时间上都没有约束导致候选结果数量巨大,因此查询效率较低,其中 SST 方法采用 MPS(Matching Point-point based Similarity)方法[21]计算轨迹相似度,索引结构 tSTAT 与 EDR 轨迹相似度方法结合. 与有时间约束的 TF 相比,TSlot 在轨迹相似度计算和索引结构上进行了双重优化,在索引阶段 TSlot 方法对时间做了更准确的索引,查询效率更高,且在计算相似度时 TSlot 采用了上下界方法过滤掉了大量不必要的轨迹间相似性计算,因此查询效率较高.

查询时间与轨迹的长度也密切相关,不同长度的轨迹所需查询时间差别较大. 为了验证不同查询轨迹长度下 TSlot 的查询时间性能,本文使用不同长度的查询轨迹进行实验,观察长短轨迹的查询效率. 图 7 所示的实验结果表明,查询轨迹越长,查询效率越低. 原因是当查询轨迹较长时,查询得到的索引项越多,导致候选结果增多,查询变慢. 此外,轨迹查询时间在不同的网格映射精度和时间槽跨度条件下表现出不同的结果.

为了分析滑动窗口宽度、时间槽跨度和网格压缩精度参数的影响,图 8～图 10 为三个参数对 *top*_20

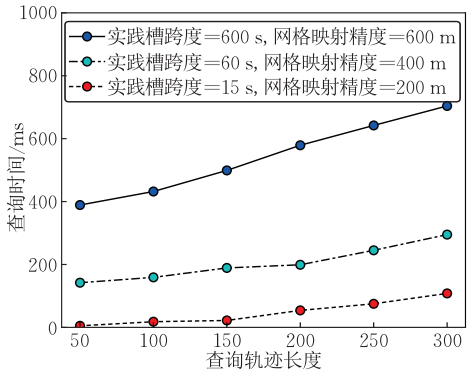


图 7 轨迹长度查询

的查询效率的影响. 实验结果表明,当网格压缩精度和时间槽跨度不变时,滑动窗口宽度越大查询效率越高. 这是因为滑动窗口越大,在时间槽内对空间索引聚类时轨迹区分度越大,同一空间索引下的轨迹越少,且搜索时查询的空间索引也更少,使得候选结果集规模较小.

在滑动窗口宽度与网格压缩精度不变的条件下,时间槽的跨度越小,查询效率越高. 这是因为当时间槽跨度较小时,在时间上进行一级索引时对时间的要求越严格,轨迹区分度越大,划分到同一时间槽的轨迹数量越小,候选结果的数量也就越小,提高了查询效率.

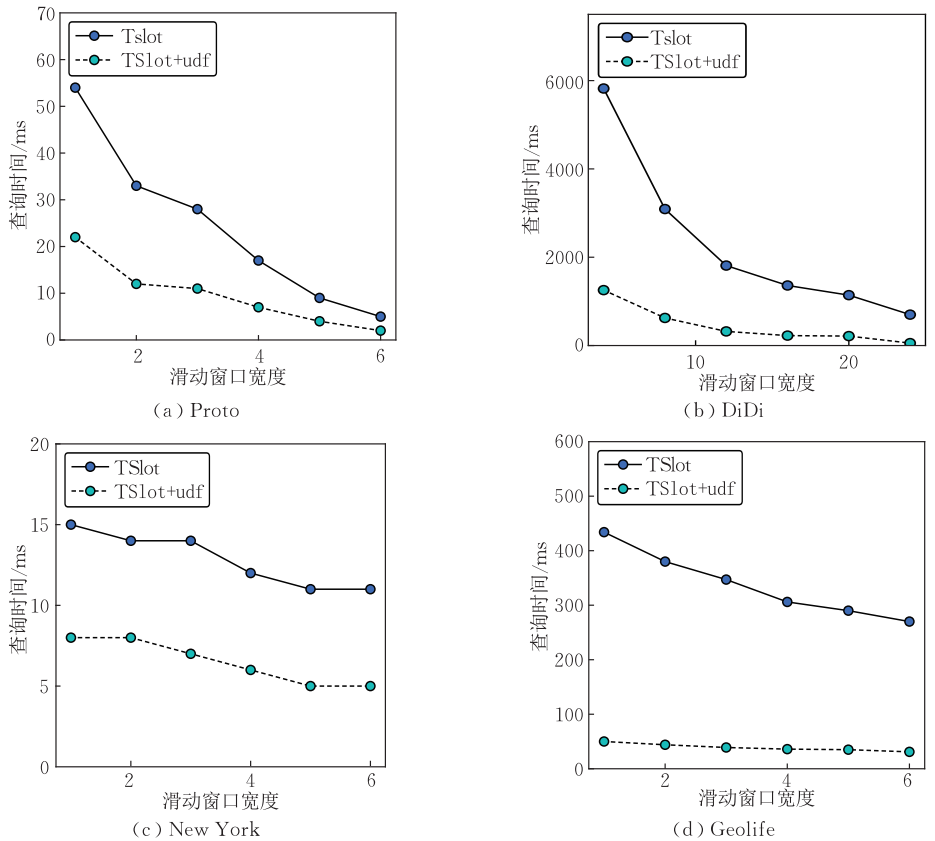


图 8 滑动窗口宽度参数影响查询效率实验

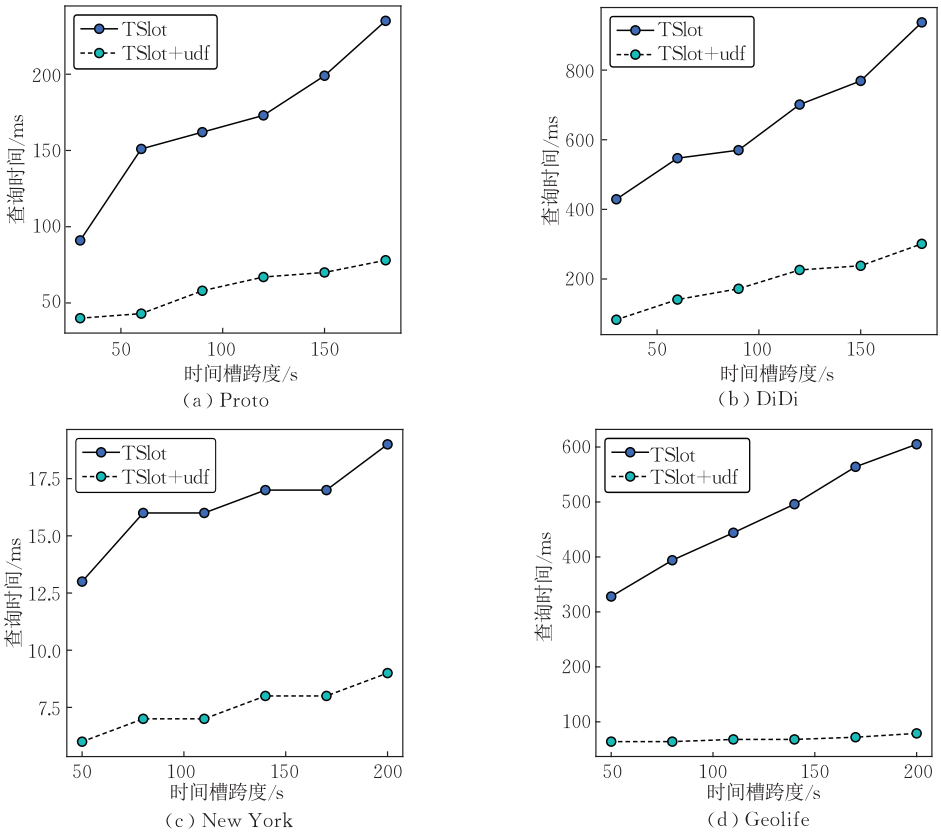


图 9　时间槽参数影响查询效率实验

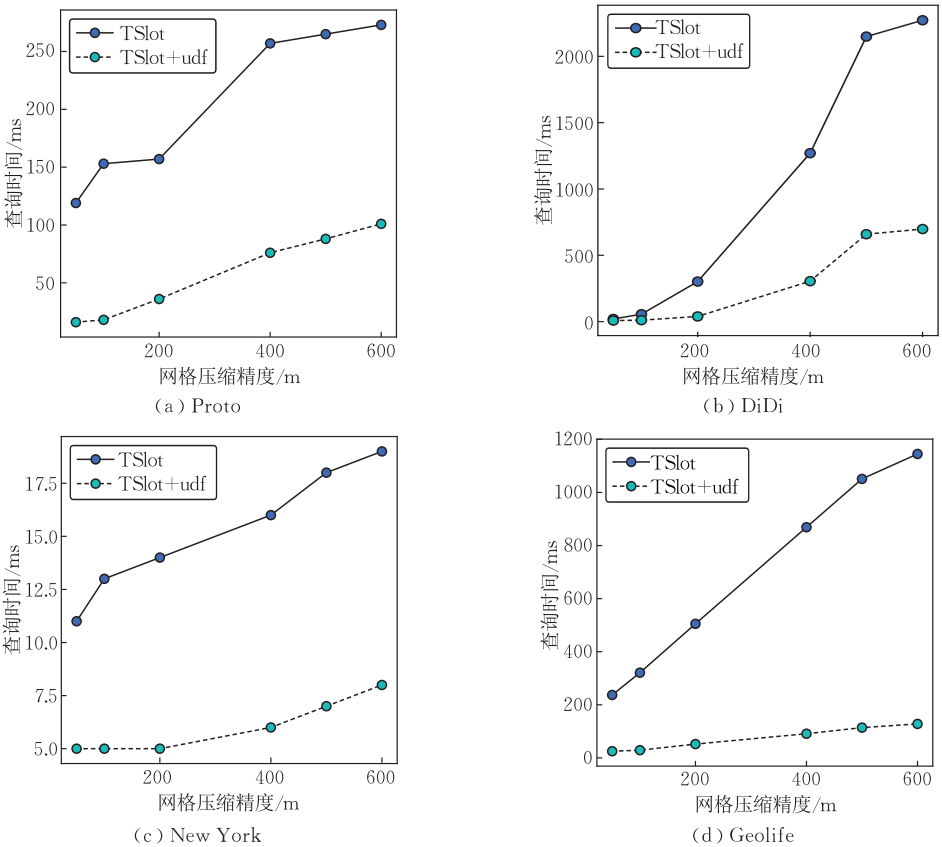


图 10　网格压缩精度参数影响查询效率实验

同理,在滑动窗口宽度和时间槽的跨度参数相同的条件下,网格压缩精度越大,压缩到同一个网格中的位置点数量越多,在进行二级空间索引时轨迹的区分度就会变小,划分到同一个分桶中的轨迹数量会增多,查询候选轨迹集合规模变大,最终导致查询效率变低.

图 11 为不同 k 值条件下的查询效率.实验结果表明,k 值越小时 TSlot 的查询效率越高.这是由于 k 值增大时堆中的轨迹与查询轨迹的相似度最小值越小,在过滤候选轨迹时轨迹之间计算相似度计算更加频繁,查询时间越长.

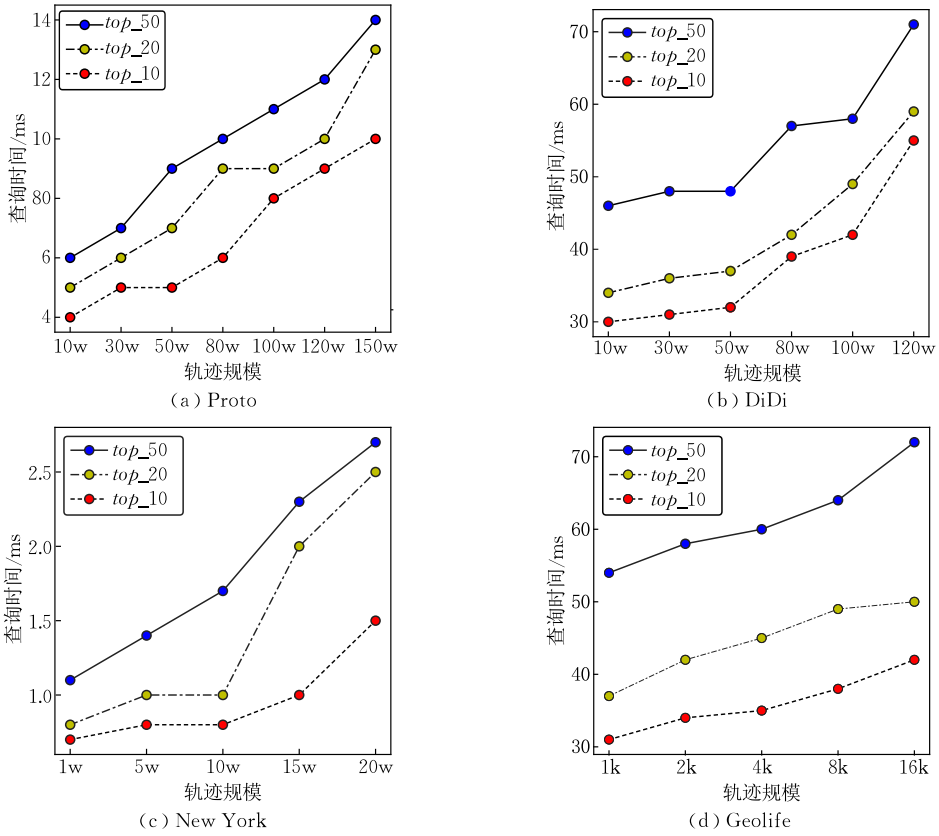


图 11 *top_k* 值查询效率对比

与已有方法相比,TSlot 在查询效率上有了巨大的提升,但是这种提升是牺牲了部分内存得到的.TSlot 将轨迹分段的方法更改为带有重复信息的子轨迹,巨大地提升了索引的区分度,但是在同一时间槽内会包含更多的轨迹段,索引结构中的索引项数量也随之增加,因此,与已有方法相比 TSlot 有更高的内存需求.图 12 为 TSlot 在不同滑动窗口宽度情况下与 SST、tSTAT、TF 等方法的内存消耗对比,实验结果表明 TSlot 方法在多个数据集上的内存消耗都高于对比方法,且当滑动窗口宽度增大时,同一时间槽内的轨迹段数量增加,TSlot 内存消耗也不断增大.在当今计算机硬件高速发展的时代,内存问题已不再是一个难题,TSlot 的内存消耗也在可以接受的水平内.

由于 TSlot 方法采用滑动步长为 1 的滑动窗口机制构建轨迹索引结构,索引结构的索引项较多导致内存消耗较大.滑动窗口截取的子轨迹的位置点具有先后依赖关系,满足前缀树数据结构的特征.因此,可以尝试使用前缀树进行优化,后续研究工作也是在保证索引结构查询有效性的前提下降低其内存消耗.

6 结 论

本文围绕伴随轨迹的高效查询展开研究,为了在索引时对轨迹进行有效的聚类,提出了一种二级时空分桶索引结构,利用滑动窗口将原始轨迹划分为等长的子轨迹,使用子轨迹的时间信息建立一级时间索引,并在此基础上利用滑动窗口实现轨迹的二级空间索引,有效减少了候选集的规模,极大地提升了伴随轨迹的查询效率.此外,本文提出了一种新的轨迹时空相似度计算方法,利用时间约束优先于空间约束的思想避免大量无效计算,在保证有效性的同时将计算时间复杂度降低到线性级别.最后通过大量的实验验证了本文提出方法的有效性.

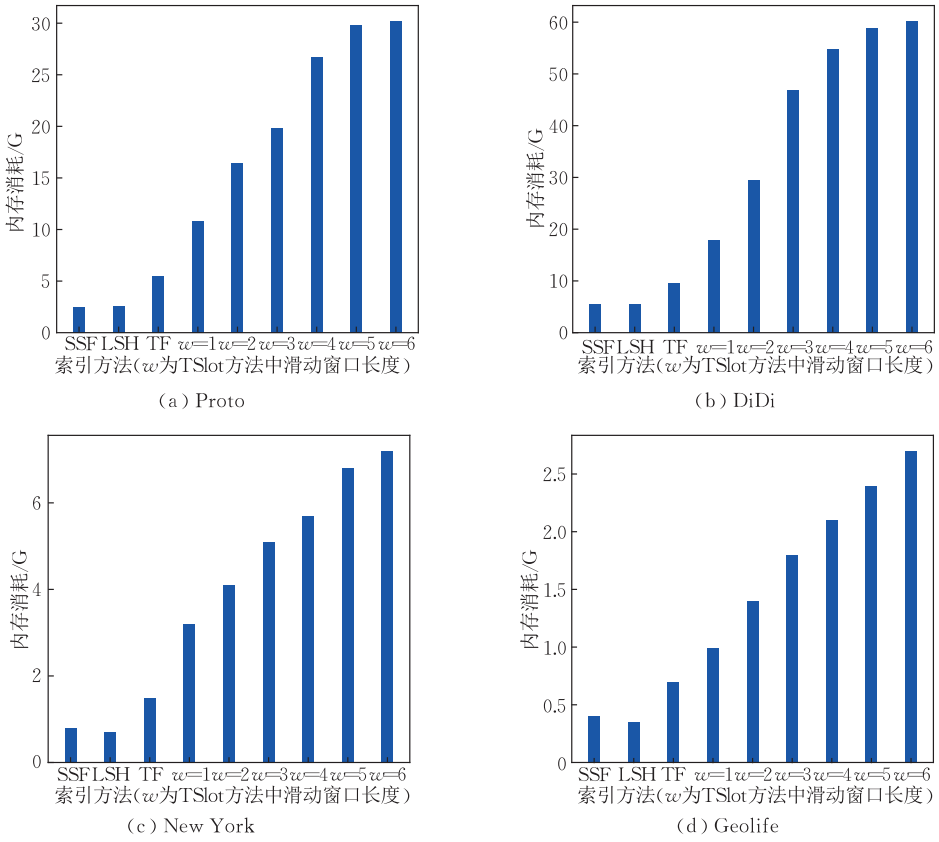


图 12　内存消耗对比实验

参　考　文　献

[1] Zhao K, Musolesi M, Hui P, et al. Explaining the power-law distribution of human mobility through transportation modality decomposition. Scientific Reports, 2015, 5(1): 9136

[2] Mariescu-Istodor R, Fränti P. Context-aware similarity of GPS trajectories. Journal of Location Based Services, 2020, 14(4): 231-251

[3] Su H, Liu S, Zheng B, et al. A survey of trajectory distance measures and performance evaluation. The VLDB Journal, 2020, 29: 3-32

[4] Tang X, Gong B, Yu Y, et al. Joint modeling of dense and incomplete trajectories for citywide traffic volume inference//Proceedings of the World Wide Web Conference. San Francisco, USA, 2019: 1806-1817

[5] Lin Z, Zhang G, He Z, et al. Vehicle trajectory recovery on road network based on traffic camera video data//Proceedings of the 29th International Conference on Advances in Geographic Information Systems. Beijing, China, 2021: 389-398

[6] Zheng Y, Zhang L, Xie X, Ma W Y. Mining interesting locations and travel sequences from GPS trajectories//Proceedings of the 18th International Conference on World Wide Web. Madrid, Spain, 2009: 791-800

[7] Park M H, Hong J H, Cho S B. Location-based recommendation system using Bayesian user's preference model in mobile devices//Proceedings of the 4th International Conference on Ubiquitous Intelligence and Computing(UIC 2007). Hong Kong, China, 2007: 1130-1139

[8] Li Y, Su Z, Yang J, Gao C. Exploiting similarities of user friendship networks across social networks for user identification. Information Sciences, 2020, 506: 78-98

[9] Chen W, Yin H, Wang W, et al. Effective and efficient user account linkage across location based social networks//Proceedings of the 2018 IEEE 34th International Conference on Data Engineering(ICDE). Paris, France, 2018: 1085-1096

[10] Han X, Wang L, Xu L, Zhang S. Social media account linkage using user-generated geo-location data//Proceedings of the 2016 IEEE Conference on Intelligence and Security Informatics (ISI). Tucson, USA, 2016: 157-162

[11] Li R, Wang R, Liu J, et al. Distributed spatio-temporal k nearest neighbors join//Proceedings of the 29th International Conference on Advances in Geographic Information Systems. Beijing, China, 2021: 435-445

[12] Hipp J R, Kim Y A. Measuring crime concentration across cities of varying sizes: Complications based on the spatial and temporal scale employed. Journal of Quantitative Criminology, 2017, 33: 595-632

[13] Andresen M A, Curman A S, Linning S J. The trajectories of crime at places: Understanding the patterns of disaggregated

crime types. Journal of Quantitative Criminology, 2017, 33: 427-449

[14] Toohey K, Duckham M. Trajectory similarity measures. SIGSPATIAL Special, 2015, 7(1): 43-50

[15] Das G, Gunopulos D, Mannila H. Finding similar time series //Proceedings of the European Symposium on Principles of Data Mining and Knowledge Discovery. Berlin, Germany: Springer, 1997: 88-100

[16] Ranu S, Deepak P, Telang A D, et al. Indexing and matching trajectories under inconsistent sampling rates//Proceedings of the 2015 IEEE 31st International Conference on Data Engineering. Seoul South, Korea, 2015: 999-1010

[17] Ta N, Li G, Xie Y, et al. Signature-based trajectory similarity join. IEEE Transactions on Knowledge and Data Engineering, 2017, 29(4): 870-883

[18] Chang J W, Bista R, Kim Y C, Kim Y K. Spatio-temporal similarity measure algorithm for moving objects on spatial networks//Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2007). Kuala Lumpur, Malaysia, 2007: 1165-1178

[19] Shang S, Chen L, Wei Z, et al. Trajectory similarity join in spatial networks. Proceedings of the VLDB Endowment, 2017, 10(11): 1178-1189

[20] Shang S, Chen L, Wei Z, et al. Parallel trajectory similarity joins in spatial networks. The VLDB Journal, 2018, 27(3): 395-420

[21] Dubuisson M P, Jain A K. A modified Hausdorff distance for object matching//Proceedings of the 12th International Conference on Pattern Recognition. Jerusalem, Israel, 1994, 1: 566-568

[22] Yuan H, Li G. Distributed in-memory trajectory similarity search and join on road network//Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE). Macao, China, 2019: 1262-1273

[23] de Almeida V T, Güting R H. Indexing the trajectories of moving objects in networks. GeoInformatica, 2005, 9(1): 33-60

[24] Zhao P, Rao W, Zhang C, et al. SST: Synchronized spatial-temporal trajectory similarity search. GeoInformatica, 2020, 24: 777-800

[25] Kanda S, Takeuchi K, Fujii K, Tabei Y. Succinct trit-array trie for scalable trajectory similarity search//Proceedings of the 28th International Conference on Advances in Geographic Information Systems. Seattle, USA, 2020: 518-529

[26] Kanda S, Tabei Y. DyFT: A dynamic similarity search method on integer sketches. Knowledge and Information Systems, 2021, 63(11): 2815-2840

[27] Pliakis N, Tiakas E, Manolopoulos Y. Indexing and progressive top-*k* similarity retrieval of trajectories. World Wide Web, 2021, 24: 51-83

[28] Chao Z, Gao H, An Y, Li J. The inherent time complexity and an efficient algorithm for subsequence matching problem. Proceedings of the VLDB Endowment, 2022, 15(7): 1453-1465

[29] Ouyang D, Wen D, Qin L, et al. Progressive top-*k* nearest neighbors search in large road networks//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. Portland, USA, 2020: 1781-1795

[30] Qiu Y X, Wen D, Qin L, et al. Efficient shortest path counting on large road networks. Proceedings of the VLDB Endowment, 2022, 15(10): 2098-2110

[31] Mark D B, Otfried C, Marc V K, Mark O. Computational Geometry Algorithms and Applications. New York, USA: Spinger, 2008

[32] Yi B K, Jagadish H V, Faloutsos C. Efficient retrieval of similar time sequences under time warping//Proceedings of the 14th International Conference on Data Engineering. Orlando, USA, 1998: 201-208

[33] Chen L, Özsu M T, Oria V. Robust and fast similarity search for moving object trajectories//Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. Baltimore, Maryland, 2005: 491-502

[34] Li X, Zhao K, Cong G, et al. Deep representation learning for trajectory similarity computation//Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE). Paris, France, 2018: 617-628

[35] Feng C, Pan Z, Fang J, et al. Aries: Accurate metric-based representation learning for fast top-*k* trajectory similarity query //Proceedings of the 31st ACM International Conference on Information & Knowledge Management. Atlanta, USA, 2022: 499-508

[36] Yao D, Cong G, Zhang C, Bi J. Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach//Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE). Macao, China, 2019: 1358-1369

[37] Luo C, Dan T, Li Y, et al. Why-not questions about spatial temporal top-*k* trajectory similarity search. Knowledge-Based Systems, 2021, 231: 107407

WANG Chen-Xu, Ph. D., associate professor. His current research interests include big data mining, artificial intelligence, and network security.

WANG Jin-Quan, M. S. His current research interests include adjoint track queries and track index structure memory optimization.

YANG Xin, M. S. candidate. His current research interests include mass trajectory processing and co-movement pattern mining.

Background

The topic of this paper is about efficient query of accompanying trajectories. Due to its essential application in various fields such as traffic planning, service recommendation, epidemic prevention and control, cross-website user identification, and public security management, there have been many relevant studies on this problem.

This paper presents a novel secondary trajectory indexing structure. Trajectories are first divided into sub-tracks based on the time of a sliding window. Then, sub-tracks within the same time slot are indexed based on the spatial information. Sub-tracks with the same position sequences are clustered into the same bucket with a hashing algorithm. Our indexing method has better distinction of different trajectories in the indexing stage and significantly reduces the number of trajectories in the candidate set for similar trajectory searching. To solve the problem of low computational efficiency of trajectory similarity, this paper proposes a new trajectory similarity computation scheme based on the constraints of time difference. The method has a linear computational complexity. Experimental results based on three real-world datasets show that the efficiency of the proposed method is 9～20 times higher than that of the state-of-the-art approaches, verifying the effectiveness of the proposed method.

This research was supported by the National Natural Science Foundation of China (62272379), the Natural Science Foundation of Shaanxi Province (2021JM-018), the National Key Research and Development Program (2021YFB1715600), and the Fundamental Research Funds for the Central Universities (xzy012023068).