DOI: 10.16725/j.1673-808X.202365

引用本文: 陈亮, 张敬伟, 陈劲方, 等. 基于高并发轨迹流的通用伴随模式挖掘框架[J]. 桂林电子科技大学学报, 2026, 46(3): 230-240.

Chen Liang, Zhang Jingwei, Chen Jinfang, et al. General co-occurrence pattern mining framework based on high-concurrency trajectory streams[J]. Journal of Guilin University of Electronic Technology, 2026, 46(3): 230-240.

基于高并发轨迹流的通用伴随模式挖掘框架

陈　亮[1]，张敬伟[1]，陈劲方[1]，黄昊天[1]，刘宇翔[2]

（1. 桂林电子科技大学 广西可信软件重点实验室, 广西 桂林　541004；
2. 桂林电子科技大学 计算机与信息安全学院, 广西 桂林　541004）

摘　要： 移动设备的普及和精准定位技术的发展, 让轨迹数据更易于采集, 为基于轨迹的用户行为分析和智能服务带来新的机遇。时空轨迹伴随模式挖掘在城市交通优化、旅游路线规划等领域具有重要应用价值, 但其涉及的大规模相似性计算带来了巨大挑战。为了应对上述挑战, 提出了一个基于高并发轨迹流的通用伴随模式挖掘框架, 其包含基于索引树的群组发现和模式挖掘 2 个部分。在基于索引树的群组发现部分, 设计了一个索引优化的方法挖掘空间近邻性群组; 在模式挖掘部分, 通过基于车辆编号的分区算法实现高效的模式挖掘。通过在真实数据集上进行实验, 对本框架进行了评估。实验结果表明, 在保持相同的伴随模式挖掘能力同时, 将各组通用伴随模式的时间消耗降低了一个数量级。

关键词： 轨迹流分析; 通用伴随模式; 空间近邻组挖掘; 索引树

中图分类号： TP311　　**文献标志码：** A　　**文章编号：** 1673-808X(2026)03-0230-11

General co-occurrence pattern mining framework based on high-concurrency trajectory streams

CHEN Liang[1], ZHANG Jingwei[1], CHEN Jinfang[1], HUANG Haotian[1], LIU Yuxiang[2]

（1. Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China;
2. School of Computer and Information Security, Guilin University of Electronic Technology, Guilin 541004, China）

Abstract: With the widespread use of mobile devices and the advances in positioning technologies, trajectory data have become increasingly accessible, enabling applications in user behavior analysis and intelligent services based on trajectory data. Spatiotemporal trajectory pattern mining plays an important role in applications such as urban traffic optimization and tourist route planning, but large-scale similarity computation remains a major challenge. To address this issue, a spatiotemporal trajectory pattern mining framework for high-concurrency trajectory streams is proposed. The framework consists of two components: index-tree group discovery and pattern mining. In the first component, an index optimization strategy is developed to identify spatially neighboring groups efficiently. In the pattern mining stage, a vehicle ID-based partition algorithm is employed to improve mining efficiency. The proposed framework is validated on real-world datasets. Experimental results show that the proposed method reduces computational time by an order of magnitude while maintaining comparable pattern mining performance.

Key words: trajectory stream mining; co-movement pattern; spatial neighbor group mining; index tree

近年来, 随着电子监控、无线传感器以及卫星定位技术的大规模应用, 获得时空轨迹数据的成本大幅降低, 导致数据规模爆炸式增长, 如成都等大城市一天内可生成超过 5 000 万条出租车轨迹数据记录,

收稿日期: 2023-03-14

基金项目: 国家自然科学基金(62267002, 62167002); 广西重点研发计划(桂科 AB22080047); 广西自然科学基金(2020GXNSFAA159117)

通信作者: 张敬伟(1977—), 男, 教授, 博士, 研究方向为大数据分析、推荐系统、海量数据存储与查询优化技术。E-mail: gtzjw@hotmail.com

而滴滴出行等商业平台每日新增轨迹数据量可达 10^6 TiB[1]。随着轨迹数据规模不断增长, 及时处理轨迹流数据的重要性不断提升。与轨迹离线数据[2]相比, 轨迹流数据具有数据并发量大、数据实时性需求高、数据质量参差不齐等特点[3]。因此, 需要设计高效的流式算法和实时质量控制策略来支持快速决策。

轨迹流数据的快速增长促进了大量轨迹相关应用的研究发展, 其中最热门的研究方向之一是共同运动实体发现, 即轨迹伴随模式挖掘。轨迹伴随指的是一组在一段时间内一起行进的移动实体。例如, 在高速公路上行驶的一组车辆或在公共场所移动的一群人。过去已经提出了多种伴随模式, 这些模式根据持续时间、移动实体对象组的形状等因素而有所不同。其中代表性模式包括成群模式、护航模式、蜂群模式和排组模式等。然而, 这些伴随模式需要输入多个参数, 而参数约束的变化可能导致一些移动物体聚类丢失(不同的参数策略可能会对实验结果产生很大影响), 所以伴随挖掘质量对参数选择十分敏感。为解决这个问题, Fan 等[4]提出了一个统一定义, 在这个定义中, 共同运动模型包含一组对象 O, 其时间序列 T 满足以下 5 个约束条件: 空间接近性, 即集合 O 中的实体彼此在空间上靠近; 重要性 M, 用于约束 O 的最小数量; 持续时间 K, 用于约束时间序列 T 的最小长度; 连续性 L, 用于约束时间序列 T 中每个连续段的最小长度; 连接性 G, 用于约束时间序列 T 中相邻段之间的间隔长度。寻找伴随实体在拥堵管理、传染病防控等许多应用领域[5]都具有重要意义。由于大量地理位置定位的联网车载设备的部署[6], 支持高并发的流式轨迹重要性日益凸显。道路拥堵管理和轨迹压缩等功能需要及时处理[7]。例如, 交通管理者希望通过识别同一路线上的车辆集群来监测道路拥堵情况, 因此能够及时检测新模式非常重要。现有离线批处理算法缺乏时效性和增量性, 不适用于对流数据的伴随模式检测。因为在离线处理时, 所有数据在处理开始都是可用的。而在流处理领域中, 数据持续实时抵达, 模式检测需要满足低延迟响应、动态更新模型、保障数据实时性等多项要求, 以此实现精准、实时的模式检测。此外, 轨迹追踪数据集具有大规模、高维度和密集的特点, 单机处理难以胜任。搜索空间近邻群组对于伴随模式挖掘至关重要, 但这种搜索过程计算开销极高。

为了解决伴随模式挖掘中高计算开销的问题, 采用分布式流处理框架 Flink[8]。Flink 的流数据处理模型和内存计算机制可以保证数据模型实时更新。设计了一个自适应分区算法来平衡负载, 充分利用分布式计算优势, 提高算法的性能和效率。此外, 提出基于 VP-Tree 的校准聚类算法(calibrated clustering algorithm, 简称 CCA), 有效解决轨迹伴随模式在空间近邻搜索阶段计算开销大的问题, 加快伴随模式发现速度。相比传统的密度聚类算法, CCA 为分布式轨迹流伴随模式挖掘框架提供了更快的群组挖掘计算。实验结果表明, 相较于密度聚类算法, CCA 在空间近邻组挖掘中的计算效率提升了一个数量级。

1　问题定义

定义 1　轨迹流。

实时轨迹流时间序列 S 由一系列时间切片组成, $S=\{s_1, s_2, \cdots, s_i, \cdots\}$, 其中, s_i 为时间戳 i 处的时间切片。

定义 2　时间切片。

轨迹流的时间切片 s_i 是一个轨迹对象的集合, $s_i=\{o_1, o_2, \cdots, o_n\}$, 其中 $o_n=\langle n, x_n, y_n\rangle$。

实时轨迹流是持续不断到达的轨迹点, 意味着总长度未知。而轨迹伴随模式挖掘的序列对象长度固定。因此, 引入一个轨迹快照概念。

定义 3　轨迹快照。

轨迹快照 $\eta=\{s_1, s_2, \cdots, s_l\}$ 是由一段长度为 l 的连续时间序列轨迹对象构成的集合。

实时轨迹流是一个不间断到达的无界数据, 运动物体的轨迹记录可能会以不同步的方式到达中央服务器, 这会干扰下游处理。为了解决这个问题, 采用时间序列离散化方法[9]将时间转化为由整数索引的小区间, 再将每条轨迹记录的时间戳映射到相应的间隔索引。假设开始时间为00: 00: 00, 将时间划分为持续时间 $\Delta t=10$ s 的间隔, 那么时间序列 $\langle$00: 00: 01,00: 00: 12,00: 00: 22,00: 00: 33$\rangle$被映射为$\langle 0,1,2,3\rangle$。称这种序列为离散时间序列, 每个离散时间戳称为时间步长 d_t。

定义 4　时间序列。

时间序列 S 表示一段连续的时间, 其中,

$$|S[m: n]|=n-m+1。$$

定义 5　时间集合。

$T=\langle S_m, S_{m+1}, \cdots, S_n\rangle$ 是一个离散的时间序列集合, 其中, 对于 $\forall x\in S_i^{\mathrm{T}}, y\in S_j^{\mathrm{T}}, i<j$, 都有 $x<y$。

定义 6　L 连续。

若一个时间序列 $|S_i|\geqslant L$, 则称时间序列 S_i^{T} 满足 L 连续。

定义 7　G 连接。

对于一个时间集合 $T=\{S_i, S_{i+1}, \cdots, S_j\}$, 若 $\forall k\in[i, j)$, 都有 $S_{k+1}[0]-S_k[|S_k|-1]\leqslant G$, 则称时间集合 T 满

足 G 连接。

定义 8 通用伴随模式(co-movement pattern, 简称 CP)。

给定一个快照集合$\eta_i=\{s_i, s_{i+1}, \cdots, s_{i+l-1}\}$, 有对象集合 $O=\{o_1, o_2, \cdots, o_n\}$, 对应近邻簇 $C=\{c_1, c_2, \cdots, c_n\}$, $(o_i, t_i, i)\in c_i$。通用伴随模式 CP$(M, K, L, G)=\{O_s, S_s\}$, 其中$O_s \subseteq O, T=\langle S_1, S_2, \cdots, S_n\rangle$, 满足: 1)对于$\forall t_k \in S_s$, 都有$O_s^{t_k} \subseteq c_s$; 2)$O_s \geqslant M$; 3)对于$\forall S_i \subseteq T$, S_i都符合 L 连续; 4)$\sum_i^n |S_i| \geqslant K$; 5)$T$ 符合 G 连接。

图 1 为通用伴随模式挖掘示例, 以该图为例理解 CP。给定 $M=3$, $K=4$, $L=2$, $G=2$,快照$\eta=\{s_1, s_2, s_3, s_4\}$。有对象集合 $O=\{o_1, o_2, o_3, o_4, o_5\}$, 每个时间片的近邻簇 $C=\{c_1, c_2, c_3, c_4, c_5\}$。其中 $c_1=\{o_1, o_2, o_4\}$, $c_2=\{o_1, o_2, o_3, o_5\}$, $c_3=\{o_1, o_2, o_3, o_5\}$, $c_4=\{o_1, o_2, o_3, o_4\}$, $c_5=\{o_1, o_2, o_3, o_4\}$。在 $t=1$ 时, 有 $\{o_1, o_2, o_4: \langle 0,1\rangle\}$; $t=2$ 时, 有新增$\{o_1, o_2, o_3, o_5: \langle 1,2\rangle\}$, $\{o_1, o_2, o_5: \langle 1,2\rangle\}$, $\{o_2, o_3, o_5: \langle 1,2\rangle\}$, $\{o_1, o_2, o_3: \langle 1,2\rangle\}$, $\{\{o_1, o_3, o_5: \langle 1,2\rangle\}$; $t=3$ 时, 更新有$\{o_1, o_3, o_4: \langle 0,1,3\rangle\}\{o_1, o_2, o_3: \langle 1,2,3\rangle\}$; 当$t=4$ 时, 检测有$\{o_1, o_2, o_4: \langle 0,1,3,4\rangle\}$, $\{o_1, o_2, o_3: \langle 1,2,3,4\rangle\}$。最后输出 2 个伴随模式: CP$_1=\{O_1, T_1\}$, $O_1=\{o_1, o_2, o_3\}$, $T_1=\langle S\rangle$, $S=\langle 1,2,3\rangle$; CP$_2=\{O_2, T_2\}$, $O_2=\{o_1, o_2, o_4\}$, $T_2=\langle S_0, S_0\rangle$, $S_0=\langle 0,1\rangle$, $S_1=\langle 3,4\rangle$。

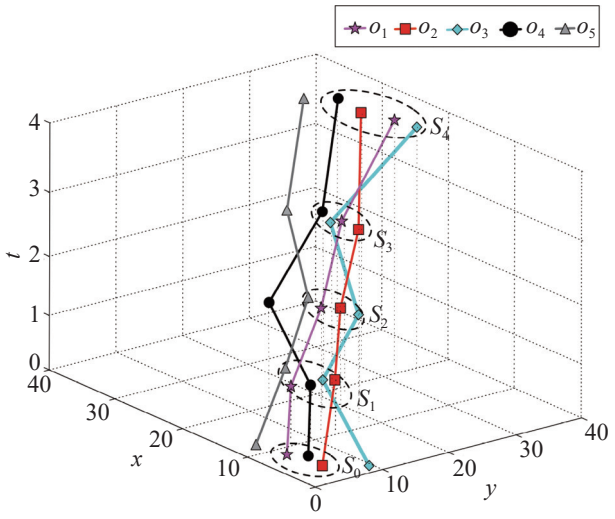


图 1 通用伴随模式挖掘示例

2 分布式校准轨迹流挖掘框架

在通用伴随模式中, 空间接近性的挖掘是非常重要且时间占比最多的步骤。其目的是将数据点划分为不同组, 以挖掘出数据中的模式。像 DBSCAN[10] 这样的密度聚类算法常被用来发现具有空间接近性的群体, 但当数据集较大时, 算法的执行时间可能会变得非常长。为了提升算法效率和整体性能, 文献[11-12] 将这些算法部署在分布式环境中, 并将数据集划分为几个区域, 不同的节点分别处理每个区域内的数据, 通过并行计算加速挖掘过程。

但在实际应用场景中, 随着越来越多的定位设备部署, 轨迹数据并发性增加, 这可能会引起分布式系统中某些节点过载, 从而导致响应时间变慢, 无法及时处理请求。即使使用基于分布式的聚类挖掘算法, 实时响应也面临重大挑战。密度聚类算法的主要时间开销用于遍历各数据对象并搜索其相似邻域。

因此, 提出了一个基于索引的聚类挖掘框架(FCMPHTS)来提高搜索效率。首先, 将数据点组织成基于空间分割的 VP-Tree[13] 结构, 这是一种树状索引, 能够迅速地识别最近的邻居聚类。然而, VP-Tree 对噪声点和异常值很敏感, 因为使用距离度量来分割空间, 噪声点和异常值可能会扭曲轨迹点之间的距离分布, 导致划分不准确。为了解决该问题, 将轨迹流的原始轨迹点与 OpenStreetMap[12-20] 中的道路网络进行匹配, 从而更准确地计算轨迹点之间的距离, 并排除不合理的轨迹点。校准后的点随后被索引为 VP-Tree。通过优化索引结构, 实现空间距离的快速计算, 降低了密度聚类算法的计算成本。

此外, 该索引结构可通过同时创建多个使用实例来提高分布式算法的响应速度, 从而降低通信开销和处理延迟。

图 2 为 FCMPHTS 整体框架, 包含数据分发、数据校准、群组挖掘和模式挖掘 4 个阶段。FCMPHTS 以轨迹流作为输入, 并将流处理转换为使用 Flink 中的滚动窗口进行快照批处理。尽管同时处理多个快照, 但是这并不意味着该方法是批处理方法。因模式挖掘需要长度的约束条件, 这使得快照处理是必要的。

为满足分布式需求, 需将轨迹数据分区。数据分发阶段, 根据轨迹数据的地理位置将数据划分为多个分区, 将相同分区内的数据分发到同一个节点。数据校准阶段, 将轨迹点校准的操作分发给多个节点执行。群组发现阶段, 在每个节点建立校准轨迹的地理空间索引结构[21-22], 并将所有空间数据存储索引节点视为具有空间相似性的一组节点, 因此当并发性增加时, 只需增加节点数量, 即可满足快速响应的需求。模式挖掘阶段, 聚合所有节点的索引簇, 进行 PPAM 模式挖掘。这种分布式处理方式能有效提高处理速度和响应能力, 使轨迹数据挖掘能更好地满足实时性需求。

算法 1 FCMPHTS。

输入: 轨迹数据流集合 S, 校准索引路网 VP-Tree, 通用伴随约束条件 (M, K, L, G)。

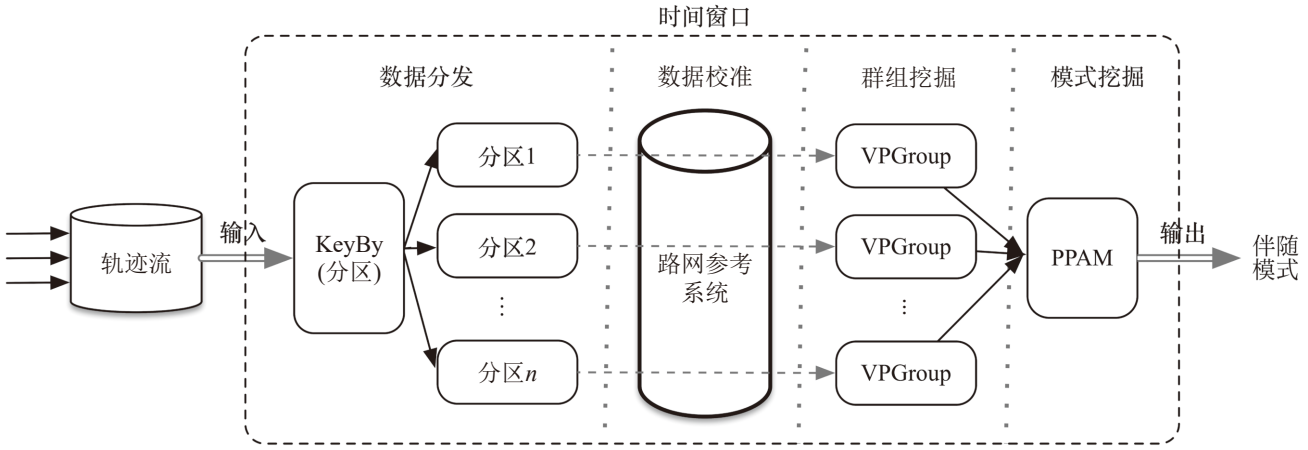


图 2　FCMPHTS 整体框架

输出: 伴随模式 R。

1. for S 中每个时间片 s do//监测不断到达的轨迹数据
2. create 滚动时间窗口 //创建长度为 l 时间窗口
3. 预处理阶段, 数据分区
4. 得到每个网格的边界 Region$_i$, 将每个网格中的轨迹点分发到不同节点
5. Keyby(Region$_i$)
6. 输出分区后的数据集合, 并发送到不同节点
7. 第一个 Map, 轨迹点校准
8. for each s_i of η do //对分区的每个轨迹点进行校准
9. C_{s_i} = CCA (s_i, VP-Tree)
10. 将C_{s_i}加入 Cali
11. end for
12. 输出校准索引轨迹流 Cali
13. 第二个 Map, 获取
14. for each index and calibrates in Cali
15. 将 (index, calibrates) 加入 $P_i(o)$
16. end for
17. 输出分区的空间接近性组集合 $P_i(o)$
18. 第一个 Reduce, 组合并
19. PPAM($P_i(o)$, (M, K, L, G))
20. 输出伴随模式 R
21. end for 。

3　数据分区及分布式组挖掘

3.1　数据分区

在处理大规模轨迹数据时, 传统单机环境的计算能力和数据处理能力难以应对庞大的数据。为解决这个问题, 分布式环境提供了更强的计算和处理能力。然而, 使用分布式环境需要将轨迹数据进行分割。这会带来一些挑战, 例如, 如何高效地划分轨迹数据, 如何有效地平衡计算负载等。针对这些挑战, 提出了一个解决机制: 基于轨迹数据的地理分布特征对轨迹数据进行分割。具体来说, 根据轨迹数据的地理位置信息, 将轨迹数据划分成多个子集, 每个子集包含地理位置信息相近的轨迹数据; 然后, 根据网格边界点的分布, 进行自适应扩展, 再将其分配给不同的节点。

定义 9　轨迹分区。

一个位置 o_i=(x_i, y_i) 所属网格单元的值可计算为

$$K_{\text{grid}} = \left\lfloor \left(x_i - \min_x\right)/l_{\text{g}} \right\rfloor + \left\lfloor (y_i - \min_y)/l_{\text{g}} \right\rfloor \times (1/r_{\text{lg}}),$$

其中: l_{g}为网格单元宽度比; r_{lg} 为单位网格宽度比; K_{grid} 为主索引。

轨迹分区为每个位置计算一个键, 具有相同键的位置将被分配到同一个子任务。根据 Flink 的运行方式, 建立网格索引后, 分区相互独立, 即不能访问其他分区中的位置。然而, 在空间接近性挖掘过程中, 一个网格的边界点需要访问周围分区的边界点, 因此需要扩展分区。

图 3 为轨迹自适应分区过程。尽管 o_4 不在网格单元 R_5 中, 但是在对 o_3 执行最近的邻居查询时, R_5 中的 o_3 将访问 R_4 的元素。为解决这个问题, 将每个位置复制成多个 GridObjects, 并将它们分发到相关网格单元。若以统一的长度作为扩展边界的标准, 则会产生不必要的索引构建成本, 以及在校准过程中的搜索成本。若RC(p_i,d_{r})以 R_5 的区域中心 p_i 为中心, 半径 d_{r} 作为扩充边界圆, 并以RC(p_i,d_{r})作为扩充范围, 除了将必要的 o_4、o_5 扩充到 R_5, 还将多余的 o_6 纳入分区中, 则会产生不必要的遍历。自适应扩展分区通过如下思想来改善分区计算的性能: 若能联合搜索边界

微元 $B(o, d_{\text{cut}})$, 其中 $o \in R_i$, 将微元整合为一个整体作为扩展分区, 则在网格 R_i 中可减少不必要的遍历。

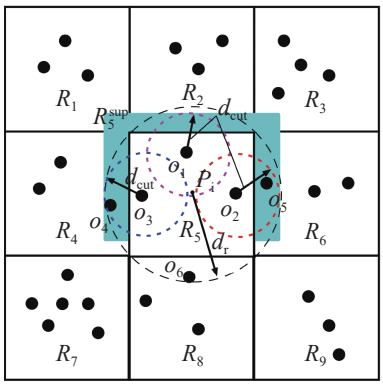


图 3 数据分区策略

定义 10 扩展分区。

对于一个网格集合 R_i 区域邻近的单元集合 R_i', 网格单元宽度 l_g, R_i 的边界坐标 $R_i = (\min_x, \min_y, \max_x, \max_y)$, 以及距离阈值 d_{cut}。对于 $\forall o \in R_i'$, 若

$$x \in [\min(\min_x R_i - d_{\text{cut}}, \min_x),\ \max(\max_x R_i + d_{\text{cut}}, \max_x)] \wedge$$
$$y \in [\min(\min_y R_i - d_{\text{cut}}, \min_y),\ \max(\max_y R_i + d_{\text{cut}}, \max_y)],$$

则 $o \in R_i^{\text{sup}}$。

图 3 中深色点边矩形突出了区域 R_5 的支持区域(表示为 R_5^{sup})。该支持区域由边界点 o_1、o_2 和 o_3 组成。R_5 内的本地邻居(如 o_4 和 o_5)被这个支持区域所增强。这样可以减少不必要的冗余数据被划入。当使用固定的网格边缘扩展技术时, 所有分区都会被分割成大小和形状相同的网格, 即使某些分区内数据量很少, 也会导致过多的冗余数据被划入分区。例如, 在图 3 中以 R_5 的中心点为圆心, 以固定长度为半径构建的支持区域会将 o_6 划分进来, 额外的冗余数据点划分会影响挖掘算法的效率和准确性。

3.2 基于 VP-Tree 的空间群组挖掘

通过一个框架来实现第一个约束性条件(紧密性), 该框架通过空间划分索引结构和近邻搜索校准算法来替换传统的基于密度的聚类方法 DBSCAN[10], 以实现基于空间的聚类校准。

3.2.1 基于 VP-Tree 的现实路网组织方法

当处理轨迹数据时, VP-Tree 是一种高效的数据结构, 可以将轨迹数据映射到一个多维空间中, 从而使得轨迹之间的相似度计算可通过空间距离的计算来完成。VP-Tree 的优势在于可以快速地进行轨迹查找和排序, 且支持多种距离度量。因此, CCA 算法中使用 VP-Tree 来加速轨迹相似度的计算, 以提高算法的效率和性能。VP-Tree 通过对每个数据点选择一个虚拟点 VP(Vantage Point)作为中心点, 计算轨迹数据点与虚拟点间的距离, 将距离进行排序, 然后根据距离大小将数据点分配到节点, 并递归地建立子树。最终, VP-Tree 的叶节点代表数据点在空间上的邻近关系, 使得轨迹数据的相似度计算更加高效。相比于传统的线性搜索方法, VP-Tree 具有更快的查询速度和更低的计算成本, 可以大大加快轨迹相似度计算的速度。

定义 11 内部节点。

VP-Tree 的内部节点定义为 $V_i = (P_i, M_i, R_i, L_i)$, $P_i = (c_i, A_i)$表示虚拟点, 其中: c_i为现实校准点, 是经过校准点c_i的道路角度方向的集合; M_i为该节点子树中所有点(属于该节点的子树)与校准点c_i间距离的中位数; R_i和L_i分别为指向该节点的左右子节点的指针。左(右)子节点包含与c_i距离小于等于(大于等于)M_i的点。

3.2.2 空间索引校准

通过 VP-Tree 结构, 可快速定位轨迹近邻点, 从而提高校准的效率和准确性。然而, 传统 VP-Tree 的近邻搜索采用欧氏距离作为测量数据点相似性的手段, 但这不足以准确评估轨迹数据的相似性。为此, 在距离指标中加入角度信息作为加权因素, 以改善对轨迹数据相似性的评估性能。使用考虑角度信息的距离度量而非纯粹的空间度量, 可以减少匹配误差, 从而提高校准精度。因此, 近邻搜索和校准使用 2 个独立的距离度量。近邻搜索使用欧氏距离, 而校准使用相干性测量算法。

定义 12 移动方向。

对于一条轨迹的连续 2 个时间片 s_i、s_{i+1} 相邻的 2 个位置 (x_i, y_i) 和 (x_{i+1}, y_{i+1}), 它们的轨迹向量与正北方向的夹角可表示为

$$\theta_i = (\arctan2(y_{i+1} - y_i, x_{i+1} - x_i) - \frac{\pi}{2}) \times \frac{180}{\pi},$$

其中: arctan2 表示反正切函数; θ_i表示第 i 个时间片中轨迹向量与正北方向的夹角; x_i和y_i分别表示轨迹中第 i 个位置的横坐标和纵坐标。

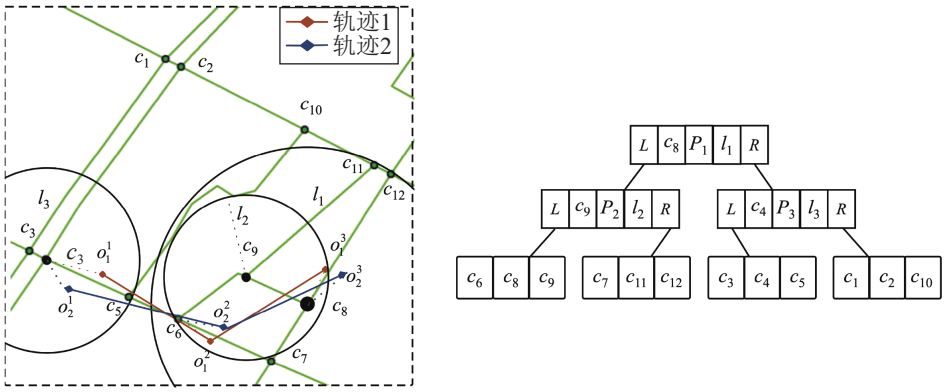
定义 13 相干性度量。

$\text{coh}(P_i, o) = \exp(-(\frac{\text{dist}(P_i, o)}{\delta}) \times |\sin \min_{i=1}^{n} |\theta_o - A_i^{P_i}\|^2)$。$\text{dist}(P_i, o)$为虚拟点 VP 与 o 之间的欧氏距离, α、δ为谐调参数。角度相似度是虚拟点 VP 的角度集合与原始轨迹点 o 的移动方向差异角度的最小值, 范围从 0 到 π。在方程中, $\exp(-(\text{dist}(P_i, o)/\delta))$ 部分是对距离的一致性评分, $\text{dist}(P_i, o)$上升时呈指数级下降。作为

RBF 核函数的α、δ谐调参数已经在相关研究[14]中进行了讨论, 实验中使用推荐的参数配置, 将α设为 5, δ设为 200, 作为相干性度量的谐调参数。

图 4 为 CCA 使用的 VP-Tree 索引结构示例, 在该示例中, 使用了 2 条轨迹点c_4和 $\{o_2^1, o_2^2, o_2^3\}$的 3 个时间片, 标准路网的路口点被用来构建 VP-Tree, 如图 4(b)所示。比较o_1^1与 P_1的中心点c_8的欧氏距离与阈值 l_1的大小, 因为比l_1大, 所以o_1^1被划分到P_3, 再与中心点c_4进行比较, 最终将o_1^1轨迹点校准为c_4。

算法 2 为 CCA。首先初始化C_{s_i}为空集。遍历数据集S_i的所有点。在当前遍历的对象上, 使用 VP-Tree 的近邻搜索算法, 获得一组近邻点的集合 neighbours, 近邻搜索使用的是欧氏距离。轨迹校准使用考虑角度的距离度量来校准 p 到最近的校准路网点 calibration_point。为了获取索引信息, 从叶子节点开始向上回溯, 找到最近的索引 align_Index。将校准点信息和索引信息作为一对加入C_{S_i}中。当所有数据点都被处理完后, 输出C_{s_i}。



(a) VP-Tree　　(b) VP-Tree索引

图 4　空间群组挖掘 VP-Tree 索引结构

算法 2　CCA。

输入: 快照η的第 i 个时间片数据集 s_i, 校准索引路网 vptree。

输出: 校准簇集合C_{s_i}。

1. C_{s_i}=[ ]
2. for each p in s_i do
3. 　//使用 VP-Tree 的近邻搜索算法搜索最邻近组
4. 　neighbour_group←vptree_neighbor_search (vptree, p)
5. 　//使用考虑角度的距离度量来校准到最近的校准点
6. 　calibration_point ← find_nearest_calibration_point(p, nearest_group)
7. 　// 从叶子节点开始向上回溯, 找到最近的索引
8. 　align_Index ← search_align_Index(vptree, p)
9. 　C_{s_i} ← (calibration_point, align_index)
10. end for
11. 输出C_{s_i}。

CCA 遍历当前快照中的所有对象, 其时间复杂度取决于区域查询次数。CCA 的时间复杂度为 $O(\log n)$, 算法通过加入角度维度使结果更符合轨迹的移动方向, 并剪枝无用对象, 以节省时间和空间, 提高后续聚类合并和模式挖掘的效率。

4　分布式伴随模式挖掘

在大规模空间密度轨迹簇的挖掘中, 寻找满足条件的伴随模式是一项费时任务。通过分布式簇挖掘方法可生成成百上千的簇, 因此仅通过遍历所有簇的所有轨迹点并对其进行挖掘是不可行的, 选择轨迹对象索引作为分区依据, 采用合理的分区策略来降低每个分区的数据输入, 以提高挖掘效率。

首先, 对于一个持续性约束的时间序列, 对具有相同分支节点序列的轨迹流进行划分, 划分为若干个子任务; 然后, 在每个子任务中, 为每个轨迹创建一个以其 ID 为键的子任务, 并将轨迹流继续进行划分。为了避免重复, 定义时刻 t 包含所有 ID 大于 o 的同一簇轨迹的集合为 $P_t(o)$, 并将所有时刻的 $P_t(o)$ 划分到关于 o.id 的子任务中, 以此实现并行枚举模式。

算法 3 为伴随模式挖掘 PPAM 算法。先初始化一个空列表 H, 再循环遍历所有合适的点集 O, 并将其插入 H 中, 然后, 循环所有 H 中的每个候选模式 h 和下一个分区, 检查该分区是否包含候选模式中的点集, 并检查时间戳是否符合约束条件, 即是否是连续时间戳或者符合 L 和 G 约束的非连续时间戳。若满

足这些条件, 则将该分区的时间戳添加到候选模式 h 的时间戳集合 $h.T$ 中, 同时检查是否满足输出条件。若满足, 则输出该模式; 若不满足, 则进入循环并跳转至下一分区继续执行。若无分区满足该条件, 则将候选模式 h 从列表中删除。

算法 3 PPAM 算法。

输入: 分区 $P_i(o)$, 约束条件 (M, K, L, G)。

输出: 伴随模式。

1. $H \leftarrow \varnothing$
2. for 每个时间戳 t in $P_i(o)$ do
3. for each point set O with $|O \geqslant M|$ do
4. H.insert($\langle O, \{t\}\rangle$)
5. end for
6. end for
7. for each candidate pattern h in H do//循环所有候选模式 h 和下一个分区 $P_i(o)$
8. for each partition $P_i(o)$ with $(t+1) \leqslant i \leqslant (t+\eta-1)$ do
9. if $h.O \subseteq P_i(o) \wedge (i - \max(h.T) = 1)$ or $(h.O \subseteq P_i(o) \wedge |h.l| \geqslant L \wedge (i - \max(h.T) \leqslant G))$ then
10. $h.T \leftarrow h.T \cup \{P_i(o).t\}$ //更新模式的时间戳
11. if $|h.T| \geqslant K \wedge |h.l| \geqslant L$ then
12. output h
13. break
14. end if
15. else
16. H.remove(h)
17. end if
18. end for
19. end for 。

图 5 为 PPAM 算法挖掘过程, 表 1 为图 5 的所有分区, PPAM 为每个分区都创建了一个子任务。在时间片 S_1, 有轨迹簇 $\{(o_1,o_2,o_3),(o_5,o_6,o_7)\}$, PPAM 将该时间片分配给 7 个子任务进行计算, 子任务 1 为 $P_1(o_1)=\{o_2,o_3\}$, 子任务 2 为 $P_1(o_2)=\{o_3\}$, 子任务 3 为 $P_1(o_3)=\varnothing$, 子任务 4 为 $P_1(o_4)=\varnothing$, 子任务 5 为 $P_1(o_5)=\{o_6,o_7\}$, 子任务 6 为 $P_1(o_6)=\{o_7\}$, 子任务 7 为 $P_1(o_7)=\varnothing$。基于重要性约束 M, 将 $|P_i(o_i)|<(M-1)$ 过滤掉, 减少不必要的遍历, 例如在表 1 中, 对于 M=3, 在时间片 S_1 上将子任务 2、3、4、6、7 的簇丢弃掉。以 $P_1(o_1)$ 的候选模式 $h=\{o_2,o_3\}$, K=4, L=2, G=1 为例来描述 PPAM 的模式列举和模式验证过程, 对于下一个分区 $P_2(o_1)=\{o_2,o_3,o_4\}$, 由于 h 是 $P_2(o_1)$ 的子集, 且是连续的时间戳, 更新 $h.l$ 和 $h.T$, 分别为 2, {1, 2}。对于下一个分区, h 并非 $P_3(o_1)$ 的子集, 但此时 $h.g<G \wedge h.l \geqslant L$。遍历 S_4 的分区, 此时 h 是 $P_4(o_1)$ 的子集, 且符合 L 和 G 约束, 因此重置 $h.g$ 和 $h.l$, 并更新 $h.T$={1,2,4}。最后遍历到快照的最后一个分区, 由于是连续时间戳, 更新 $h.T$={1,2,4,5}, 同时 $|h.T| \geqslant K$, 输出候选模式 h。

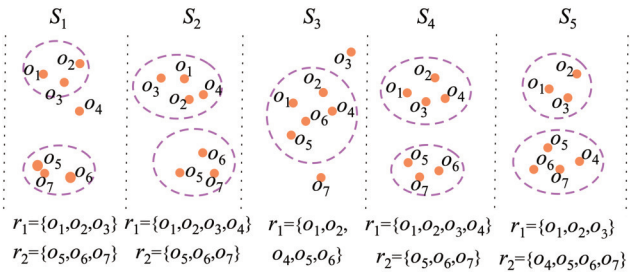


图 5 PPAM 算法挖掘过程

表 1 PPAM 算法的模式列举

分区	S_1	S_2	S_3	S_4	S_5
o_1	{o_2, o_3}	{o_2, o_3, o_4}	{o_2, o_4, o_5, o_6}	{o_2, o_3, o_4}	{o_2, o_3}
o_2	{o_3}	{o_3, o_4}	{o_4, o_5, o_6}	{o_3, o_4}	{o_3}
o_3	∅	{o_4}	∅	{o_4}	∅
o_4	∅	∅	{o_5, o_6}	∅	{o_5, o_6, o_7}
o_5	{o_6, o_7}	{o_6, o_7}	{o_6}	{o_6, o_7}	{o_6, o_7}
o_6	{o_7}	{o_7}	∅	{o_7}	{o_7}
o_7	∅	∅	∅	∅	∅

5 实验

5.1 实验数据

使用两组真实场景的数据集: Taxi[15-16] 和 GeoLife[17-18], 数据集信息如表 2 所示。

1)Taxi 是一个真实的轨迹数据集, 包含了 2014 年 8 月到 2014 年 9 月期间成都市出租车行驶记录, 涵盖了车辆的 GPS 坐标、时间戳、速度等信息。这些数据记录了每辆出租车在城市中的移动轨迹。

2)GeoLife 数据集记录了 182 名用户 3 年以上的轨迹数据。该数据集的 GPS 轨迹由一系列时间戳点表示, 每个点都包含纬度、经度等信息。这些轨迹由不同的 GPS 记录仪和 GPS 手机记录, 并具有各种采样率。在这个数据集中, 91.5% 的轨迹采用密集采样方式记录, 采样时间间隔介于 1~5 s, 空间间隔为 5~10 m。

5.2 实验环境

本实验在一个由 4 台 Dell 服务器组成的 Flink[19] 集群上运行, 该集群包含 17 个虚拟节点。每台服务

表 2　轨迹数据集信息

属性	GeoLife	Taxi
轨迹对象	18 670	13 605
轨迹点	44 189 853	27 363 286
平均时间片中对象	2 557	12 668
数据量/GiB	1.29	1.87

表 3　参数及默认值

参数	参数值
r_{lg} /%	0.4，0.8，1.6，3.2，6.4
d_{cut}	0.02，0.04，0.06，0.08
M	7，10，13，16，19
K	120，150，180，210，240
L	15，20，25，30，35
G	5，10，15，20，25

器都配备了 128 GiB RAM 和 56 个 CPU 内核(Intel Xeon Gold 5117 CPU @2.0 GHz), 每个节点都具有 10 个 CPU 内核、32 GiB RAM 和 200 GiB 磁盘容量。该集群运行的操作系统版本为 CentOS 7.0, Java 虚拟机采用 JDK 1.8, 流计算 Flink 版本为 1.7.2, 并且搭建在基于 Hadoop 2.7.3 的 Yarn 上。所有代码都使用 Scala 语言编写, 编辑器为 IDEA 2020.1。相关的依赖关系使用 Maven 4.0.0 配置, 并打包上传至 Flink 集群。

5.3　数据预处理

在预处理阶段, 对移动物体原始 ID 重新编码, 生成从 1 开始的连续序号。为了提高后续数据挖掘和应用的准确性, 对 GeoLife 数据集每 5 s 采样一次, 对 Taxi 数据集每 30 s 采样一次, 并通过线性插值填充缺失的数据。为了进一步完善数据, 删除了多余的数据, 只保留了固定频率以上的数据。计算相邻 2 个时间点之间每个 ID 的方向向量与正北方向的角度(0°~360°)。然后, 对数据集进行离线地理分区, 包括每个数据点的用户 ID、纬度、经度、时间戳、角度以及所在的地理分区信息。这种分区能够更好地分析和处理运动轨迹, 以便进行后续的数据挖掘和应用。

5.4　实验对比与分析

5.4.1　FCMPHTS 参数设置

在实验中, 对影响性能的几个因素进行了研究。表 3 列出了参数设置, 其中, r_{lg} 和 d_{cut} 被设置为整个数据集的最大距离的百分比。在每组实验中, 改变一个参数, 同时将其他参数固定为默认值。网格宽度的默认值为 0.8%, 距离阈值设为 0.04%, 通用模式挖掘参数 M、K、L、G 的默认值依次为 13、180、25、15。

为了满足实时系统快速响应的要求, 必须在下一个快照到达前进行伴随模式的挖掘。采用一种分布式方法来测试其效率, 并与基线方法进行比较。GeoLife 数据集包含 17 280 个时间片, 其中分布了 44 189 853 个轨迹点, 平均每个快照包含约 2 557 个轨迹点。相比之下, Taxi 数据集包含 2 160 个时间片, 其中分布了 27 363 286 个轨迹点, 平均每个快照包含约 12 668 个轨迹点。因此, GeoLife 数据集的稀疏程度要比 Taxi 数据集更高, 但其响应时间却更短。

5.4.2　FCMPHTS 与 Baseline 性能对比

1)群组挖掘性能。

在实验开始前, 对 FCMPHTS 算法进行了采样点数据测试, 以检验其挖掘数量是否与 Baseline[20] 方法相同, 进而验证 FCMPHTS 计算结果的准确性。

图 6 反映了 2 个关键因素的影响, 即距离阈值 d_{cut} 和网格宽度 r_{lg} 对 FCMPHTS 框架的群组挖掘的影响。

图 6(a)、(b) 描述了将 d_{cut} 从 0.02% 增加到 0.08% 情况下的平均延迟。与预期一致, CCA 在实时性上的表现优于 RJC, CCA 挖掘算法的平均延迟比 GeoLife 中的 RJC 延迟更大。这是因为 CCA 挖掘算法必须要有构建固定索引的额外时间成本。相反, 在 Taxi 数据集中的分组挖掘的平均延迟要比 Baseline 方法小一个数据级别。图 6(c)、(d) 为将 r_{lg} 从 0.4% 扩大到 6.4% 时的平均延迟。CCA 和 RJC 的聚类性能随 r_{lg} 的变化呈现先提升后下降的趋势。这是因为, 当 r_{lg} 太小时, 多个分区会有额外的管理开销, 而当 r_{lg} 太大时, 每个分区需要处理的数据量增加, 导致处理延迟上升, 最终降低聚类性能。

2)模式挖掘性能。

图 7 为 M、K、L 和 G 在 Taxi 和 GeoLife 数据集上对挖掘性能的影响变化。FCMPHTS 在模式挖掘中的延迟性能比基准方法更好。

随着 M、K 或 L 的增长, 由于会生成较少的有效候选项, 平均延迟降低。然而, 随着 G 的持续增长, 平均延迟会增加, 这是因为此时会生成更多有效模式, 从而提升了计算开销。

CCA 算法将原始轨迹点校准到道路点上, 因此生成的群组簇数量少于基准算法。聚类数量减少后, 模式枚举和模式验证阶段要遍历的项集数量相应减少, 从而降低整体时间成本。

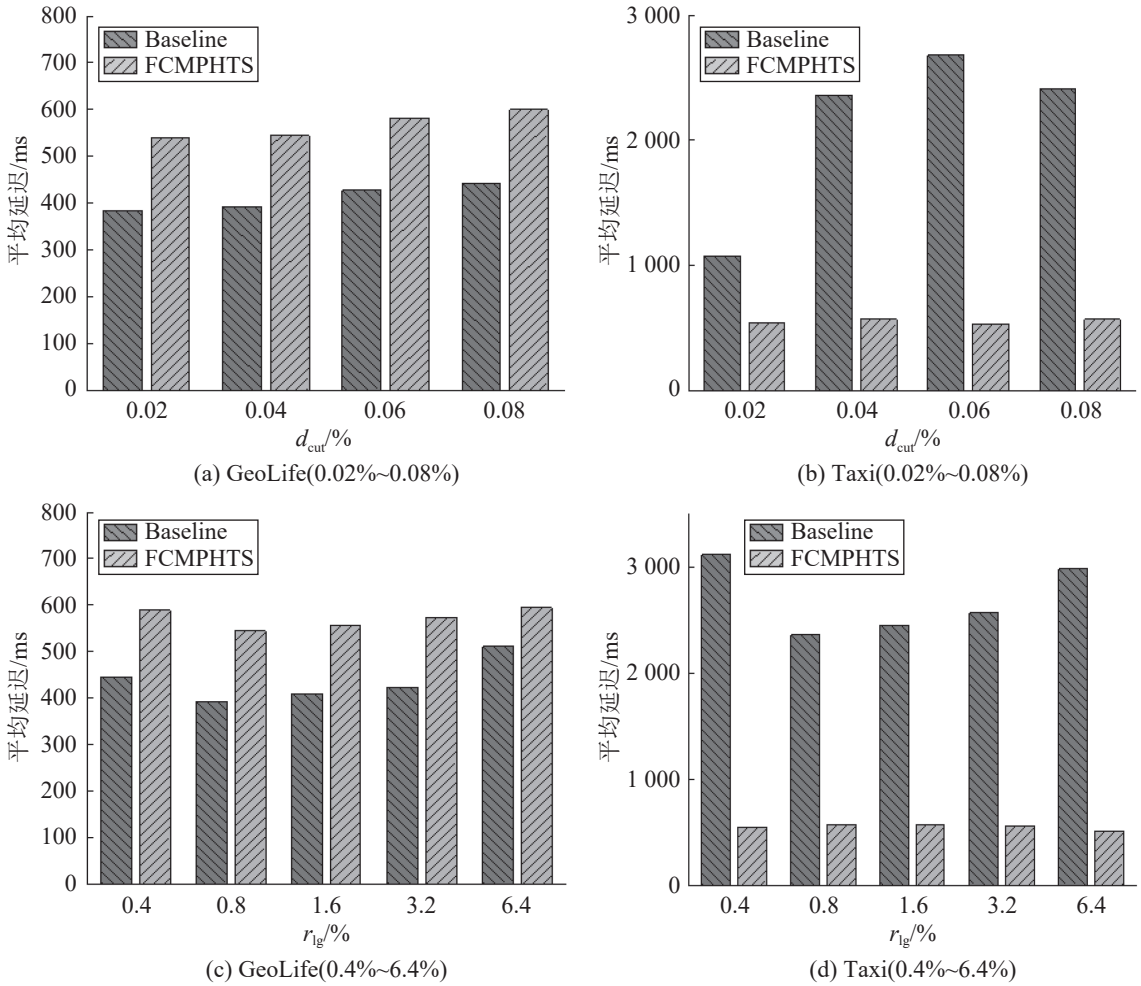


图 6 群组挖掘性能评估结果

6 结束语

提出了一个面向高并发轨迹流的伴随模式挖掘框架，以解决轨迹流伴随模式挖掘中实时响应问题。首先，介绍了框架的总体结构；其次，列举了轨迹数据流和伴随模式的相关定义；然后，基于数据的地理分布和网格边界点的分布设计了一个自适应的网格扩展划分策略。在群组挖掘阶段，提出了一个空间划分索引树，加速将原始轨迹点映射到现实路网点，以检索最近邻群组。通过在真实数据集上进行实验，比较了该框架与基准方法的挖掘性能。实验结果表明，该框架的性能得到显著提升。

参考文献：

[1] Zhang Lingyu, Hu Tao, Min Yue, et al. A taxi order dispatch model based on combinatorial optimization[C]// Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2017: 2151-2159.

[2] Fan Qi, Zhang Dongxiang, Wu Huayu, et al. A general and parallel platform for mining co-movement patterns over large-scale trajectories[J]. Proceedings of the VLDB Endowment, 2016, 10(4): 313-324.

[3] Benkert M, Gudmundsson J, Hubner F, et al. Reporting flock patterns[J]. Computational Geometry, 2008, 41(3): 111-125.

[4] Jeung H, Shen Hengtao, Zhou Xiaofang. Convoy queries in spation-temporal databases [C]// IEEE 24th International Conference on Data Engineering. Piscataway, NJ: IEEE Press, 2008: 1457-1459.

[5] Li Zhenhui, Ding Bolin, Han Jiawei, et al. Swarm: mining relaxed temporal moving object clusters[J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 723-734.

[6] Li Yuxuan, Bailey J, Ku L, et al. Efficient mining of platoon patterns in trajectory databases[J]. Data and Knowledge Engineering, 2015, 100: 167-187.

[7] Wiratma L, Kreveld M V, Loffler M, et al. An experimental evaluation of grouping definitions for moving entities[C]//Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. New York, NY: ACM Press, 2019: 89-

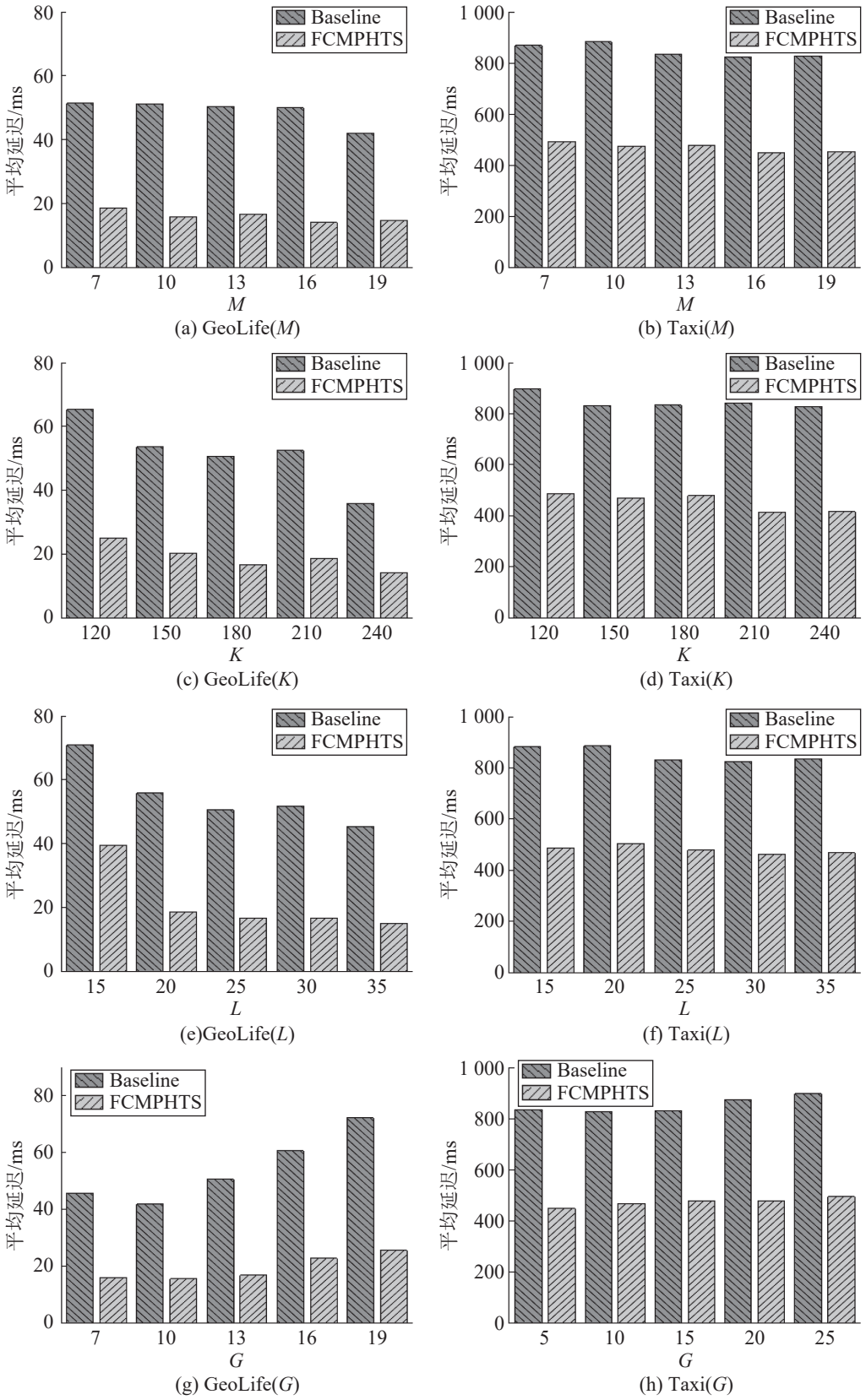


图 7　伴随模式挖掘性能评估结果

96.

[8] Chen Lu, Gao Yunjun, Fang Ziquan, et al. Real-time distributed co-movement pattern detection on streaming trajectories[J]. Proceedings of the VLDB Endowment, 2019, 12(10): 1208-1220.

[9] Li Yuxuan, Bailey J, Kulik L, et al. Efficient mining of platoon patterns in trajectory databases[J]. Data and Knowledge Engineering, 2015, 100: 167-187.

[10] Ester M, Kriegel P, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise [C]//Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining. Palo Alto, CA: AAAI Press, 1996: 226-231.

[11] Yang Dingyu, Guo Jianmei, Wang Zhijie, et al. FastPM: an approach to pattern matching via distributed stream-processing[J]. Information Sciences, 2018, 453: 263-280.

[12] Tritsarolis A, Theodoropoulos G S. Theodoridis Y. Online discovery of co-movement patterns in mobility data[J]. International Journal of Geographical Information Science, 2021, 35(4): 819-845.

[13] Bozkaya T, Ozsoyoglu M, et al. Indexing large metric spaces for similarity search queries[J]. ACM Transactions on Database Systems, 1999, 24(3): 361-404.

[14] Chen Zaiben, Shen Hengtao, Zhou Xiaofang. Discovering popular routes from trajectories[C]//2011 IEEE 27th International Conference on Data Engineering. Piscataway, NJ: IEEE Press, 2011: 900-911.

[15] Zheng Yu, Zhang Lizhu, Xie Xing, et al. Mining interesting locations and travel sequences from GPS trajectories[C]// Proceedings of the 18th International Conference on World Wide Web. New York, NY: ACM Press, 2009: 791-800.

[16] Xian Yongyi, Liu Yan, Xu Chuanfei, et al. Parallel discovery of trajectory companion pattern and system evaluation[J]. Journal of Information Processing, 2020, 28: 538-550.

[17] Zheng Yu, Li Quannan, Chen Yukun, et al. Understanding mobility based on GPS data[C]//Proceedings of the 10th International Conference on Ubiquitous Computing. New York, NY: ACM Press, 2008: 312-321.

[18] Zheng Yu, Xie Xing, Ma Weiying. GeoLife: a collaborative social networking service among user, location and trajectory[J]. IEEE Data Engineering Bulletin, 2010, 33(2): 32-39.

[19] Rabl T, Traub J, Katsifodimos A, et al. Apache flink in current research[J]. It-Information Technology, 2016, 58 (4): 157-165.

[20] Fang Ziquan, Gao Yunjun, Pan Lu, et al. CoMing: a real-time co-movement mining system for streaming trajectories[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York, NY: ACM Press, 2020: 2777-2780.

[21] Chen Lu, Gao Yunjun, Song Xuan, et al. Indexing metric spaces for exact similarity search[J]. ACM Computing Surveys, 2023, 55(6): 1-39.

[22] Zheng Yu. Trajectory data mining: an overview[J]. ACM Transactions on Intelligent Systems and Technology, 2015, 6(3): 1-41.

编辑：张所滨