

DCPFS:分布式轨迹流伴随模式挖掘框架

张康威, 张敬伟, 杨青, 胡晓丽, 单美静

引用本文

张康威, 张敬伟, 杨青, 胡晓丽, 单美静 [DCPFS:分布式轨迹流伴随模式挖掘框架](#) [J]. 计算机科学, 2022, 49(11A): 211100268-10.

ZHANG Kang-wei, ZHANG Jing-wei, YANG Qing, HU Xiao-li, SHAN Mei-jing. [DCPFS:Distributed Companion Patterns Mining Framework for Streaming Trajectories](#) [J]. Computer Science, 2022, 49(11A): 211100268-10.

相似文章推荐 (请使用火狐或 IE 浏览器查看文章)

Similar articles recommended (Please use Firefox or IE to view the article)

[基于人工蜂群算法的多维函数优化加速方法](#)

Acceleration Method for Multidimensional Function Optimization Based on Artificial Bee Colony Algorithm

计算机科学, 2022, 49(11A): 211200075-6. <https://doi.org/10.11896/jsjx.211200075>

[基于申威众核处理器的Office口令恢复向量化研究](#)

Study on Office Password Recovery Vectorization Technology Based on Sunway Many-core Processor

计算机科学, 2022, 49(11A): 210900176-5. <https://doi.org/10.11896/jsjx.210900176>

[开放式环境下基于向量表征与计算的动态访问控制](#)

Vector Representation and Computation Based Dynamic Access Control in Open Environment

计算机科学, 2022, 49(11A): 210900217-7. <https://doi.org/10.11896/jsjx.210900217>

[基于流量分析发现未知UDP反射放大协议](#)

Discovery of Unknown UDP Reflection Amplification Protocol Based on Traffic Analysis

计算机科学, 2022, 49(11A): 211000089-5. <https://doi.org/10.11896/jsjx.211000089>

[面向通用一致性优化的通信高效的异步ADMM算法](#)

Communication Efficient Asynchronous ADMM for General Form Consensus Optimization

计算机科学, 2022, 49(11): 309-315. <https://doi.org/10.11896/jsjx.211200006>

DCPFS:分布式轨迹流伴随模式挖掘框架

张康威¹ 张敬伟¹ 杨 青² 胡晓丽¹ 单美静³

1 广西可信软件重点实验室(桂林电子科技大学) 广西 桂林 541004
2 广西自动检测技术与仪器重点实验室(桂林电子科技大学) 广西 桂林 541004
3 华东政法大学信息科学与技术系 上海 201620
(zkw951@126.com)

摘 要 随着定位技术的广泛使用,产生了以轨迹流形式收集的海量时空数据,如何从中挖掘有用的信息得到越来越多学者的关注。从轨迹流中挖掘伴随模式指在同一时间内发现具有高度相似行为的群体,对于交通管理、推荐系统的实时应用至关重要。然而,现有的研究只达到秒级响应,面对大规模轨迹数据难以在毫秒级的时间内快速响应。因此,提出了分布式轨迹流挖掘框架 DCPFS。框架的主要模块包括:1)为了减少基于密度的聚类算法 DBSCAN 由于大规模数据带来的大量时间消耗,研究基于分布式部署方案,设计了数据分区策略和聚类合并算法,确保聚类的并行性及准确性;2)由于现实中轨迹移动具有方向性,在聚类阶段增加方向维度以减少冗余聚类;3)鉴于模式挖掘阶段涉及对聚类结果的交叉,设计了并行交叉算法来提高挖掘效率;4)基于 Flink 分布式大数据流处理平台实现了 DCPFS。以成都市出租车 GPS 数据集和谷歌生活数据集为例进行实验,验证了所提框架比基准方法具有更快的响应速度。

关键词: 大数据;轨迹流;伴随模式;密度聚类;分布式

中图法分类号 TP311.13

DCPFS:Distributed Companion Patterns Mining Framework for Streaming Trajectories

ZHANG Kang-wei¹, ZHANG Jing-wei¹, YANG Qing², HU Xiao-li¹ and SHAN Mei-jing³

1 Guangxi Key Laboratory of Trusted Software(Guilin University of Electronic Technology), Guilin, Guangxi 541004, China
2 Guangxi Key Laboratory of Automatic Detecting Technology and Instruments (Guilin University of Electronic Technology), Guilin, Guangxi 541004, China
3 Department of Information Science and Technology, East China University of Political Science and Law, Shanghai 201620, China

Abstract The widespread use of location technology leads to huge volumes of spatio-temporal data collected in the form of trajectory data streams. How to discover useful information from it has attracted more and more scholars' attention. Mining companion pattern from trajectory stream refers to discovering groups with highly similar behaviors at the same time, which is essential for real-time applications of traffic management and recommendation systems. However, the existing research only achieves a second-level response, and it is difficult to respond quickly in milliseconds to large-scale trajectory data. Therefore, this paper proposes a distributed companion patterns mining framework DCPFS. The main contents of our work include: 1) In order to reduce the time consumption of the density-based clustering algorithm DBSCAN for large-scale data, this paper proposes a data partition strategy and clustering merging algorithm based on a distributed deployment plan to ensure clustering parallelism and accuracy. 2) Because the trajectory movement is directional in reality, we increase the direction dimension to reduce the redundancy in the clustering. 3) We designed a parallel intersection algorithm to improve the efficiency of the intersection of clustering results in the pattern mining stage. 4) We implement DCPFS on the Flink distributed big data processing platform and use Chengdu taxi GPS dataset and Google life dataset for experiments. Comprehensive empirical study demonstrates that the proposed framework has faster response speed than the baseline method.

Keywords Big data, Trajectory streams, Companion patterns, Density-based clustering, Distributed

1 引言 数据^[1],记录了各种应用中人、车辆和动物的移动,如社交网络、交通管理、科学研究和军事侦察。1)在 Foursquare 中,以用户访问餐厅和购物中心的顺序作为轨迹。在许多像 Ge-

定位设备和位置跟踪技术的进步,产生了大量的轨迹

基金项目:国家自然科学基金(61862013,U1811264,U1711263);广西自然科学基金(2020GXNSFAA159117,2018GXNSFAA281199);广西可信软件重点实验室重点课题(KX202052);广西自动检测技术与仪器重点实验室主任基金课题(YQ21102)

This work was supported by the National Natural Science Foundation of China(61862013,U1811264,U1711263), Natural Science Foundation of Guangxi(2020GXNSFAA159117,2018GXNSFAA281199), Key Project of Guangxi Key Laboratory of Trusted Software(KX202052) and Foundation of Guangxi Key Laboratory of Automatic Detection Technology and Instrument(YQ21102).

通信作者:单美静(shanmeijing@ecupl.edu.cn)

olife 这样的 GPS 轨迹分享网站上,人们上传自己的旅行或运动路线和朋友分享。2)大城市的许多出租车都安装了 GPS 传感器,其实时位置以数据流的形式报告给交通管理系统。3)生物学家们为其研究记录候鸟等动物的运动轨迹。4)战场中军事卫星监视指定区域,收集可能的入侵者的行动,这些轨迹被军事网络收集。在上述应用中,人们期望发现一起移动的对象组,即伴随。例如,交通监管部门希望发现相同路线上的汽车,以监控道路拥堵状况;科学家想研究物种迁移的路线。伴随信息还可以用于资源配置、安全管理、传染病控制等方面。

近年来,基于轨迹数据流的伴随模式挖掘技术不断产生,但现有系统仅适用于小规模数据集,在大规模轨迹数据中表现不佳,主要原因及挑战如下。

(1)实时性:轨迹数据流具有无限增长、快速到达、实时更新且难以存储的特性,因此需要及时进行处理。这就要求轨迹流数据中的伴随模式挖掘算法满足时间上高效的要求,而传统算法并不能很好地满足以上要求。

(2)方向性:在真实的轨迹流伴随模式中,数据点移动方向多样。鉴于传统的 DBSCAN 对移动对象进行聚类,容易产生冗余数据信息。为了达到较好的聚类效果,挖掘算法需要具有方向性,而现有的算法并未考虑方向因素。

(3)并行性:城市级别的轨迹数据实时到达,规模巨大,使得单台服务器和单线程的处理难以负载。因此,需要多台服务器对数据分块处理并增加算法的并行度,而现有的算法都是单节点单线程算法,无法应对大规模轨迹数据。

本文针对上述轨迹流伴随模式挖掘分析的挑战,提出了新的解决方案。该方案包括两个部分:基于方向的分布式聚类 and 并行的伴随模式挖掘,并提出了新的分布式流聚类算法和模式挖掘算法。

本文的主要贡献如下:

(1)基于轨迹数据的地理信息设计分布式策略,均匀划分地理区域,将数据分发到不同的节点;设计聚类合并算法 MC (Merge Cluster),合并聚类结果。

(2)根据轨迹数据的特性,抽取轨迹数据的方向信息,设计基于方向的 ADBSCAN (Angle-DBSCAN)算法,聚合方向且位置相近的轨迹点。

(3)在伴随模式匹配阶段提出 PCPM (Parallel Companion Pattern Mining)算法,设计多线程的匹配方式和并行的模式匹配,提高伴随模式挖掘效率。

(4)对以上内容进行整合,基于 Flink 分布式大数据流处理平台,设计了分布式轨迹流伴随模式挖掘框架 DCPFS,并以谷歌生活数据集和成都市交通数据为例进行实验,验证了本框架具有更快的处理速度。

2 相关工作

轨迹数据的伴随模式挖掘具有较高的研究和应用价值,目前轨迹数据的伴随模式挖掘主要分为历史轨迹和轨迹流两种数据形式。本节将介绍伴随模式的发展进程。

2.1 历史轨迹伴随模式挖掘研究进展

关于静态轨迹数据的伴随模式挖掘,当前已经有了较完善的研究。Benkert 等^[2]要求满足一定数量的群体对象在指定半径的圆形范围内共同移动连续且足够长的时间戳,命名

为成群模式(Flock)。Jeung 等^[3]针对成群模式无法检测指定圆形范围外的对象,提出了护航模式(Convoy),该模式要求群体对象之间密度相连,共同移动连续且足够长的时间戳。但是护航模式无法检测出有对象加入或离开的情况,因此 Jeung 等^[4]要求在无对象加入和离开时,群体间时间戳连续且密度相连,有对象加入或离开时要求时间连续且密度相连,称为发展护航模式(Evolving Convoy)。对于以上模式都无法应对时间不连续的情况,Li 等^[5]提出了蜂群模式(Swarm),规定密度相连的群体对象,时间戳可以不连续,总的时间大于一定的阈值。蜂群模式对时间要求的放松会挖掘出噪声模式,因此 Li 等^[6]提出了排组模式(Platoon),规定密度相连的群体对象的时间戳不要求全部连续,但密度相连的连续时间满足要求且密度相连的总时间戳满足要求。鉴于以上内容都是在单机环境,Orakzai 等^[7]设计了分布式策略,将相同时间戳的数据散列到同一个节点独立处理,多个节点共同处理全部数据。Zhang 等^[8]针对分布式策略中存在大量松散连接的问题,提出了两阶段伴随模式挖掘框架。通过数据预处理优化、伴随优化等,使伴随模式挖掘具有更好的性能。

2.2 轨迹流伴随模式挖掘研究进展

关于轨迹流的伴随模式挖掘,学术界也有许多的研究。Vieira 等^[9]将成群模式应用在轨迹流上,提出了剪枝算法减少冗余的候选簇,并设计了网格聚类算法以节省聚类时间。Tang 等^[10]在护航模式的基础上提出了伴游模式,针对轨迹流挖掘对时间效率要求较高,设计了一种规模更小且关系更加稳定的数据来结构加速聚类过程。Li 等^[11]为了提高伴游模式的响应速度,首先对轨迹数据在线约简,然后对新到达的数据构建时间堆,通过在堆中更新移动对象之间的关系而不需要对每个时间戳到达的数据聚类,提高了挖掘的效率。Lan 等^[12]引入了时间窗口和微组的概念,轨迹流数据落在时间窗口内并且当前时间窗口内的一组轨迹可以由一条轨迹代替,那么这组轨迹被称为微组,之后对微组实时更新。Naseirian 等^[13]结合蜂群模式和伴游模式,允许小组间成员可以相互交流,只要多数成员不发生移动,且满足规定时间长度阈值。Shein 等^[14]改进了 Zheng 等^[15]的方法,在交叉阶段使用 Jaccard 计算簇的相似度,在验证阶段使用海明权重计算对象占比,有效提升了挖掘的时间性能。Xian 等^[16]针对大规模轨迹数据的异构性,设计了轨迹槽模型,将槽点附近相同对象的轨迹数据划分为折线,以欧氏距离为基础设计折线之间的距离度量方式,并使用 K-D 树均衡分区数据,设计基于折线的 DBSCAN 聚类,提高了传统伴随模式挖掘的负载能力,但是在实时响应中需要收集完轨迹槽后再进行处理,在处理大规模轨迹流时存在一定延时。

轨迹数据的伴随模式研究主要包含以下几种类型:Flock,Convoy,Evolving convoy,Swarm,Platoon。近年来的研究也主要是针对以上伴随模式在实时应用场景中的优化。本文设计的轨迹流伴随模式挖掘框架相较于以往的微批处理策略更接近实时性,实验部分将会证明我们的观点。

3 问题定义

本节将对轨迹流的伴随模式挖掘框架进行概述。我们引入伴随模式的框架和算法,表 1 列出了文中所使用的符号。

表 1 符号定义
Table 1 Symbol definition

Symbol	Definition	Symbol	Definition
S	轨迹流	s, s_i, s_j	轨迹流中的快照
O_{s_i}	快照 s_i 包含移动对象的集合	x_i, y_i	在快照 s_i 的空间坐标
C_{s_i}	快照 s_i 聚类的集合	o_1, o_2, o_i	移动对象
R	伴随模式	$c_{s_i}^k$	快照 s_i 的第 k 个簇
R'	候选伴随模式	δ_s	伴随模式的数量阈值
ϵ	距离阈值	δ_t	伴随模式的周期阈值
μ	密度阈值	δ_a	密度连接的角度阈值
N_ϵ	密度可达对象集合	θ_j^i	对象 o_j 在快照 s_i 的角度

定义 1(轨迹流) 一条轨迹流 S 包含一系列的快照集合 $S = \{s_1, s_2, \dots, s_i, \dots, s_n\}$, 其中 s_i 是在时间戳 i 的快照 s 。

定义 2(快照) 每个快照 s_i 包含移动对象的集合 $O_{s_i} = \{(o_1, x_1, y_1), \dots, (o_i, x_i, y_i), \dots, (o_n, x_n, y_n)\}$, 其中 x_i, y_i 表示对象 o_i 在快照 s_i 的空间坐标。

定义 3(密度可达) 令 O 表示一个快照中对象的集合, ϵ 表示距离阈值, μ 表示密度阈值, 则 $N_\epsilon = \{o_j \in O | dist(o_i, o_j) \leq \epsilon\}$ 。如果 $o_j \subset N_\epsilon$ 且 $|N_\epsilon| \geq \mu$, 那么从对象 o_i 到对象 o_j 是密度可达的。

定义 4(密度连接) 令 O 表示一个快照中对象的集合, 如果存在对象的连接链 $\{o_1, \dots, o_n\} \in O$, 其中 $o_1 = o_j, o_n = o_i$, 从 o_i 到 o_{i+1} 是密度可达的, 则 o_i 与 o_j 是密度相连的。

定义 5(快照聚类) 给定一个快照 s_i , 定义 $C_{s_i} = \{c_{s_i}^1, c_{s_i}^2, \dots, c_{s_i}^k, \dots, c_{s_i}^n\}$, C_{s_i} 表示快照聚类的集合, 其中 $c_{s_i}^k$ 是快照 s_i 的第 k 个簇。

定义 6(伴随模式) 令 δ_s 表示数量阈值, δ_t 表示周期阈值, 一组轨迹对象 R 被称为伴随模式, 如果满足以下条件:

- (1) R 的成员之间在时间周期 t 内是密度连接的, 其中 $t \geq \delta_t$;
- (2) R 的成员数量 $R.size \geq \delta_s$ 。

定义 7(候选伴随) 令 δ_s 表示数量阈值, δ_t 表示周期阈值, 一组轨迹对象 R' 被称为候选伴随, 如果满足以下条件:

- (1) R' 的成员之间在时间周期 t 内是密度连接的, 其中 $t < \delta_t$;
- (2) R' 的成员数量 $R'.size \geq \delta_s$ 。

例 1 图 1 给出模式挖掘的过程, 其中包含 6 个对象在 7 个快照中的位置。假设集合对象数量阈值为 4, 时间阈值为 3, $\{o_1, o_2, o_3, o_4, o_5\}$ 在起始时间相互接近且向同一方向运动, o_1 在 t_3 位置由于角度与其他对象的角度偏差大于阈值而脱离集合, 其中 $\{o_2, o_3, o_4, o_5\}$ 数量和时间阈值在 t_4 时刻被实时输出。随着时间的推移, o_6 在 t_5 时刻被加入到集合中, 由于在 t_5 时刻不满足时间阈值, 包含 o_6 的集合对象只输出 $\{o_2, o_3, o_4, o_5\}$, 当到达 t_7 时刻 $\{o_2, o_3, o_4, o_5, o_6\}$ 满足时间阈值被输出。

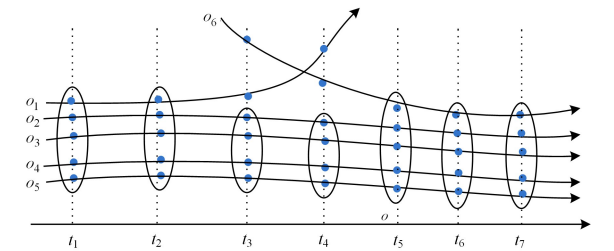


图 1 模式挖掘过程
Fig. 1 Pattern mining process

4 分布式挖掘框架

聚类操作在伴随模式挖掘中占有重要的地位, 且在整个过程中消耗了大量的时间。在真实场景中, 轨迹数据集包含大量的快照数, 并且随着时间的推移移动对象不断产生快照, 在面对城市级别的大规模轨迹数据, 基于单机模式的挖掘算法难以做到快速响应。现有的分布式部署方案被应用于聚类阶段, 使得响应速度提升数倍。目前的聚类方法对每个快照下的对象进行聚类, 各快照下的聚类操作互不影响, 但是在轨迹数据实时到达的场景中, 需要对当前的快照进行及时聚类, 以多个快照为范围的聚类的方式并不适用。考虑到现实场景中距离较远的对象很难被聚类为一个簇, 因此将当前地理范围内的位置信息划分为多个区域, 每个区域赋予唯一的编号, 相同区域中的对象产生的信息被收集到同一个节点处理, 在不同区域中移动的对象产生的位置信息由不同的节点处理。每个节点对收集到的轨迹点执行聚类操作, 由于移动对象一般会朝着某一方向前进, 运动方向相差过大的对象被聚类到一起不具有实际意义反而增加了聚类的冗余。因此在聚类阶段考虑移动对象的角度因素, 削减了聚类结果的规模, 减少了下一阶段聚类合并和模式挖掘中需要对比的对象数量和时间消耗。在地理分区的边缘, 由于地理划分的原因, 可能存在距离相近的对象被聚类到不同的簇中, 因此需要扫描所有簇将存在相同对象的簇合并为一个簇。最后, 将当前快照的所有簇与上一个快照的所有簇进行对比, 如果存在交叉的对象, 交叉的结果会被存储为候选的伴随模式, 但候选伴随模式持续存在且满足快照数量时被确定为伴随模式。由于每个快照中的簇数量有限, 交叉操作仅需要很少的时间开销。因此充分利用当前计算机的多核特性, 使用并行方案执行交叉操作, 提高了交叉过程的效率。

轨迹流伴随模式挖掘框架如图 2 所示, 包含数据处理、数据分发、密度聚类、聚类合并和模式挖掘 5 个阶段。为满足分布式聚类的需求, 需要对轨迹数据分区。图 2(a) 根据轨迹数据的地理位置, 划分多个分区; 图 2(b) 对接收到的数据流按所在分区执行 Keyby 操作, 将相同分区内的数据发送到同一个节点; 图 2(c) 为分布式处理过程, 将框架最耗时的 ADBSCAN 聚类操作分发给多个节点执行, 当数据量增加, 只需要增加节点数量即可满足快速响应的需求; 图 2(d) 聚合所有节点的聚类簇, 合并拥有相同对象的簇; 图 2(e) 由于模式挖掘中的交叉操作数据量有限, 因此仅单节点处理足以满足要求, 同时为了提高交叉的效率, 采用多线程方式, 并行执行交叉步骤。DCPFS 框架算法流程如算法 1 所示。序号 1—2 为数据预处理阶段, 根据地理区域的划分将数据划分为多个区域, 其中区域的数量等于分布式节点的数量; 序号 3 中用于监测不断到达的数据流; 序号 4—15 为当前快照创建时间窗口, 窗口大小为当前快照时间; 序号 5—7 根据第一步中得到的每个对象的分区编号执行 Keyby 算子, 利用哈希函数分发到不同的节点, 其中具有相同分区编号的对象被发送到相同的节点; 序号 8—10 为 Map 阶段, 每个节点对接收到的当前分区数据执行 ADBSCAN, 并得到当前分区的密度聚类簇集合 C_{s_i} ; 序号 11—13 为第一个 Reduce 阶段, 对 Map 阶段聚类簇集合 C_{s_i} 汇聚到主节点执行 MC 算法, 输出合并后簇的集合 C'_{s_i} ; 序号

14—16 为第二个 Reduce 阶段,对第一个 Reduce 合并后簇的集合 C'_{s_i} 执行 PCPM 算法,与候选伴随集合 R 取交集,生成新的候选伴随,并输出当前快照的伴随模式结果 r' 。

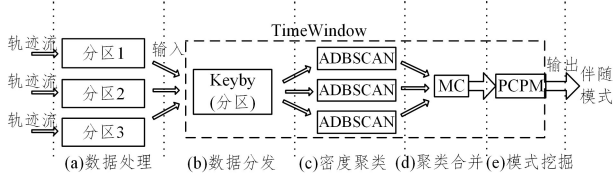


图2 轨迹流伴随模式挖掘框架

Fig. 2 Companion patterns mining framework for streaming trajectories

算法1 DCPFS 框架

输入: 轨迹数据流集合 S , 聚类半径 ϵ , 最小簇基数 δ_s , 夹角阈值 δ_a , 伴随模式时间周期 δ_t

输出: 伴随模式 R

1. (a) 预处理阶段, 数据分区;
2. 得到分区边界 n_i , 输出每个对象的分区编号 $Region_i$;
3. for each coming snapshots of S do // 监测不断到达的轨迹数据
4. CreateTimeWindow // 创建时间窗口, 以每个快照时间设定窗口的大小;
5. (b) Keyby 阶段, 数据分发;
6. Keyby($Region_i$);
7. 输出分区后的数据集合并发送到不同的节点。
8. (c) Map 阶段, 密度聚类;
9. ADBSCAN($S_i, \epsilon, \minPts, \text{angle}$);
10. 输出基于密度的聚类簇集合 C_{s_i} ;
11. (d) 第一个 Reduce, 聚类合并;
12. MC(C_{s_i});
13. 输出合并后簇的集合 C'_{s_i} ;
14. (e) 第二个 Reduce, 模式挖掘;
15. PCPM(C'_{s_i}, R);
16. 输出伴随模式 r' , 候选伴随 R ;
17. End for.

图3给出了DCPFS框架挖掘伴随对象的示例。在该示例中, 首先将数据划分为3个分区, 从3个分区产生的对象集合 $\{o_1, o_2, o_3, o_4, o_5, o_6\}, \{o_6, o_7, o_8, o_9, o_{10}, o_{11}\}, \{o_{11}, o_{12}, o_{13}, o_{14}, o_{15}, o_{16}\}$ 被标记不同的分区编号 N_1, N_2, N_3 。我们将标记为 N_1, N_2, N_3 的数据集分别发送到不同的节点。 $\{o_1, o_2, o_3, o_4, o_5, o_6\}, \{o_6, o_7, o_8, o_9, o_{10}, o_{11}\}, \{o_{11}, o_{12}, o_{13}, o_{14}, o_{15}, o_{16}\}$ 分别在节点1, 节点2和节点3被执行 ADBSCAN 密度聚类操作, 分别得到聚类结果簇 $\{o_1, o_2, o_3\}, \{o_4, o_5, o_6\}, \{o_6, o_7, o_8\}, \{o_9, o_{10}, o_{11}\}, \{o_{11}, o_{12}, o_{13}\}, \{o_{14}, o_{15}, o_{16}\}$, 其中每个簇中的对象之间满足定义4。可以看到, $\{o_4, o_5, o_6\}$ 和 $\{o_6, o_7, o_8\}, \{o_9, o_{10}, o_{11}\}$ 和 $\{o_{11}, o_{12}, o_{13}\}$ 存在相同的对象。因此将 $\{o_1, o_2, o_3\}, \{o_4, o_5, o_6\}, \{o_6, o_7, o_8\}, \{o_9, o_{10}, o_{11}\}, \{o_{11}, o_{12}, o_{13}\}, \{o_{14}, o_{15}, o_{16}\}$ 汇集到相同节点执行 MC 聚类合并操作, $\{o_4, o_5, o_6\}$ 与 $\{o_6, o_7, o_8\}$ 被合并为 $\{o_4, o_5, o_6, o_7, o_8\}$, $\{o_9, o_{10}, o_{11}\}$ 与 $\{o_{11}, o_{12}, o_{13}\}$ 被合并为 $\{o_9, o_{10}, o_{11}, o_{12}, o_{13}\}$, 最终得到4个簇 $\{o_1, o_2, o_3\}, \{o_4, o_5, o_6, o_7, o_8\}, \{o_9, o_{10}, o_{11}, o_{12}, o_{13}\}, \{o_{14}, o_{15}, o_{16}\}$ 。在执行 PCPM 模式挖掘操作时, 假设上一个快照模式挖掘的结果为 $\{o_1, o_2, o_3\}, \{o_5, o_6, o_7, o_8\}, \{o_{10}, o_{11}, o_{12}\}$ 且被缓存, 于是从内存中取出 $\{o_1, o_2, o_3\}, \{o_5, o_6, o_7, o_8\}, \{o_{10}, o_{11}, o_{12}\}$, 与

当前聚类合并后的簇 $\{o_1, o_2, o_3\}, \{o_4, o_5, o_6, o_7, o_8\}, \{o_9, o_{10}, o_{11}, o_{12}, o_{13}\}, \{o_{14}, o_{15}, o_{16}\}$ 对比并相互取交集得到 $\{o_1, o_2, o_3\}, \{o_5, o_6, o_7, o_8\}, \{o_{10}, o_{11}, o_{12}\}$ 。由于 $\{n, p, q\}$ 是新产生的簇且与缓存中的簇没有相同对象, 因此 $\{o_1, o_2, o_3\}, \{o_5, o_6, o_7, o_8\}, \{o_{10}, o_{11}, o_{12}\}, \{o_{14}, o_{15}, o_{16}\}$ 满足定义7, 加入并替换缓存对象, 且如果在缓存中存在簇满足定义6, 会被立即输出为伴随模式。

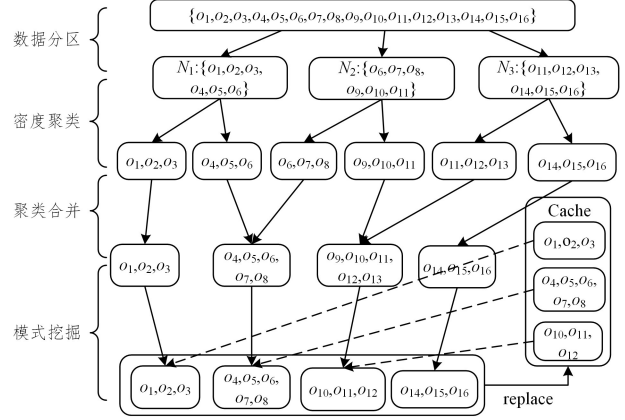


图3 DCPFS 过程示例图

Fig. 3 Diagram of DCPFS process example

5 数据分区及分布式聚类

在轨迹流的伴随模式挖掘中, 聚类操作需要消耗大量时间。本节设计了轨迹数据分区策略, 提出了融合方向的分布式聚类算法 ADBSCAN, 并设计了聚类合并算法 MC。

5.1 数据分区

在面对大规模轨迹数据时, 单机环境往往难以处理, 而分布式环境下必须对数据分区, 使每个节点单独处理不同的数据。本节结合轨迹数据的地理分布的特点, 设计了基于地理分区的数据划分方法, 其主要思想是将轨迹数据按照经纬度放入有序存储的集合中, 并根据规则等分集合并选取边界点, 将轨迹数据按照边界点切分到不同区域, 并发送到不同节点进行处理。本节描述了详细的分区方案。

定义8(分区边界) 给定轨迹流快照 s_i 中轨迹的集合 O_{s_i} , x_i 和 y_i 表示对象 o_i 在快照 s_i 的空间坐标, 假设 $List$ 为有序列表存储集合中所有对象的 x 轴坐标, 则分区边界的位置表示如下:

$$n_i = \frac{size}{N}(N-i)$$

其中, $size$ 表示 $List$ 的容量, N 表示分区的数量, 且 $0 \leq i < N$ 。

分区过程如下:

(1) 计算轨迹数据集的最大和最小经纬度, 建立轨迹数据集的最小外接矩形 Rectangle;

(2) 对 Rectangle 进行划分, 使每个分区中包含相同数量的对象, 并根据定义8分别得到分区位置编号 n_i 。

(3) 由 n_i 生成对应的分区 $Region_i$, 其中 $Region_i$ 的范围为 $(n_i - r, n_i + 1 + r)$, 这里令 $r = \epsilon$, ϵ 为密度连接的距离阈值。

图4详细介绍了轨迹数据集的分区过程: 首次计算得到轨迹数据集的最大和最小经纬度位置 (x_1, y_1) 和 (x_2, y_2) , 并建立最小外接矩形 Rectangle; 其次对 Rectangle 的长边划分, 根据分布式节点数量 n , 计算得到 $n_1 - n_5$ 的位置, 最后由 r

计算得到 $Region_1 - Region_4$ 的范围为 $(n_1 - r, n_2 + r) - (n_4 - r, n_5 + r)$ 。

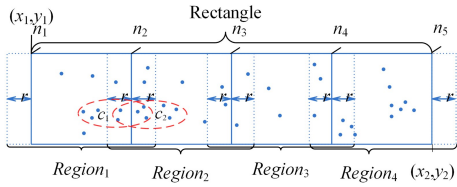


图4 数据分区策略(电子版为彩图)

Fig.4 Data partition strategy

5.2 ADBSCAN 算法

在真实场景中,挖掘运动方向相似的轨迹比挖掘运动方向不同的轨迹更为实际。现有的研究大多采用基于欧几里德距离的 DBSCAN 算法,该算法不考虑轨迹的运动方向,只考虑距离因素,导致大量不实用的轨迹点聚类。ADBSCAN 是基于密度的聚类算法,考虑到物体运动方向和距离的两个因素,引入了主要受城市道路弯曲角度和城市道路交叉口角度影响的角度阈值 δ_a 。

图 5 给出了 DBSCAN 算法的聚类结果。首先,在每个从节点上,找出核心对象,红色点表示核心对象的位置,圆形区域表示其邻域范围。其次,从某一核心对象出发,产生密度可达的聚类簇,当访问完所有点则结束计算。ADBSCAN 算法在 DBSCAN 算法的基础上,计算当前快照与上一个快照位置向量与正北方向的夹角,从而加入方向维度信息。

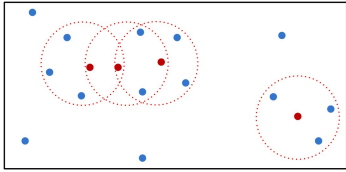


图5 DBSCAN 算法聚类结果

Fig.5 Clustering results of DBSCAN algorithm

ADBSCAN 要求密度相连的群体对象中,核心对象与邻域范围内的点方向差小于阈值 δ_a 。

定义 9(移动方向) 给定两个相邻快照 $s_{i-1}, s_i, (x_1, y_1)$ 和 (x_2, y_2) 是对象 o_j 在快照 s_{i-1} 和 s_i 的坐标,轨迹向量与正北方向的夹角即移动方向 θ_j^i 可以表示为:

$$\theta_j^i = \begin{cases} -\arccos \frac{y_2 - y_1}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}, & x_1 > x_2 \\ \arccos \frac{y_2 - y_1}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}, & x_1 < x_2 \end{cases}$$

定义 10(方向差) 给定快照 s_i 两个轨迹点 o_1, o_2 以及两点的移动方向 θ_1^i, θ_2^i ,两个轨迹点的方向差 $D_{\text{angle}}(o_1, o_2)$ 可以表示为:

$$D_{\text{angle}}(o_1, o_2) = \begin{cases} |\theta_1^i - \theta_2^i|, & \theta_1^i > 0, \theta_2^i > 0 \text{ or } \theta_1^i < 0, \theta_2^i < 0 \\ \min\{2\pi - |\theta_1^i - \theta_2^i|, |\theta_1^i - \theta_2^i|\}, & \theta_1^i > 0, \theta_2^i < 0 \text{ or } \theta_1^i < 0, \theta_2^i > 0 \end{cases}$$

定义 11(距离度量) 给定快照 s_i 两个轨迹点 o_1, o_2 , 以及经纬度 (x_1, y_1) 和 (x_2, y_2) , 两个轨迹点的空间距离 $D_{\text{space}}(o_1, o_2)$ 可以表示为:

$$D_{\text{space}}(o_1, o_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

定义 12 (ADBSCAN) 给定快照 s_i , 对于 $\forall o_i \in c_{s_i}^k, \forall o_j \in c_{s_i}^k$, 则 o_i 与 o_j 之间密度连接满足 $D_{\text{space}}(o_1, o_2) < \epsilon$ 和

$D_{\text{angle}}(o_1, o_2) < \delta_a$, 其中 $c_{s_i}^k$ 是快照 s_i 的第 k 个簇, δ_a 是密度连接的角度阈值。

ADBSCAN 算法如算法 2 所示,算法中的距离度量采用欧氏距离。其中序号 2 开始遍历数据集 D 的所有点;序号 3 判断当前遍历到的对象是否已经被访问过,如果被访问过则跳过此对象,访问下一个对象,如果未被访问过则标记为核心对象;序号 4 设置 p 为已访问,找出与点 p 距离不大于 r 且角度差不大于 δ_a 的所有点集合 N ;序号 5-21,如果 p 的邻域范围内对象数满足 δ_s ,则遍历 p 的邻域集合 N ;序号 7-14,如果 N 内的对象 p' 已经被访问,则判断其是否是噪声点,如果是噪声点,说明 p' 不属于任何簇,将 p' 标记为非噪声点并聚类到当前簇;序号 15-19,如果 p' 未被访问,则查找 p' 的邻域 N' 并加入 N 中;序号 22,如果 p 的邻域内对象数量小于 δ_s ,则将 p 标记为噪声点;序号 25 最终返回聚类后的簇的集合 C_{s_i} 。

算法 2 ADBSCAN //融合角度的密度聚类算法

输入:包含 n 个数据的第 i 个快照的数据集 S_i ,半径阈值 r ,角度阈值 δ_a ,邻域密度阈值 δ_s 。

输出:基于密度的聚类簇集合 C_{s_i}

1. 初始化 D 中所有的点为 unvisited;
2. for each p in D do
3. if p .visit = visited then continue;
4. 置 p 为 visited,找出与 p 满足 $D_{\text{space}} < \epsilon$ 且 $D_{\text{angle}} < \delta_a$ 的点集合 N ;
5. if N .size() $> \delta_s$ then
6. for each p' in N do
7. if p' .visit == visited then
8. if p' 未被聚类到某个簇;
9. if p' 被标记为噪声点 then
10. 将 p' 取消标记为噪声点;
11. end if
12. 将 p' 聚类到当前簇;
13. end if
14. else
15. 找出与 p' 距离大于 r 的所有点集合 N' ;
16. if N' .size $\geq \delta_s$ then
17. 将集合 N' 加入集合 N 中;
18. end if
19. end if
20. end for
21. else
22. 标记点 p 为噪声点;
23. end if
24. end for
25. return 聚类后的簇集合 C_{s_i} .

ADBSCAN 算法需要访问当前快照中的所有对象,区域查询的次数决定了算法的时间复杂度,ADBSCAN 的时间复杂度为 $O(n^2)$,由于加入了角度维度,其更加符合轨迹的移动方向,剪枝无用的对象,为执行下一步的聚类合并和模式挖掘节省了时间和空间上的开销。

5.3 MC 算法

如图 4 中所示椭圆区域内的两个簇 c_1 和 c_2 ,如果在单机运行环境中 c_1 和 c_2 密度可达,会被划分为一个簇。然而在分布式环境中, c_1 和 c_2 被划分为两个区域,分别属于 $Region_1$ 和 $Region_2$ 。因此需要对分布式聚类结果进行聚合,将包含相同对象的簇合并。下面给出了 MC 的定义及计算方法。

定义 13(MC) 当前快照 s_i 下对于给定的两个簇 $c_{s_i}^k, c_{s_i}^l$,

如果 $c_{s_i}^i \in Region_1, c_{s_j}^j \in Region_2, Region_1 \cap Region_2 \neq \emptyset$, 则 $\exists o_1$ 使得 $o_1 \in c_{s_i}^i$ 且 $o_1 \in c_{s_j}^j$, 因此 $c_{s_i}^i \cap c_{s_j}^j \neq \emptyset$, 那么 $c_{s_i}^i \bowtie c_{s_j}^j$ 。

与直接按照分区聚类相比, MC 算法解决了分布式聚类中大簇被分割成多个簇造成结果不准确的问题, 提升了分布式聚类的准确性。

聚类合并算法 MC 的具体过程如算法 3 所示。序号 1 和序号 2, 第一个指针停留在第 i 个簇, 第二个指针 j 遍历 i 之后的簇。序号 3 判断后遍历到的簇是否存在于第一个簇 $Region_i$ 的相邻 $Region_j$ 中, 如果不存在, 则继续查找下一个簇, 序号 4-6 若两个簇 c_i 和 c_j 有相同的对象, 则将 c_j 中的所有对象添加到 c_i 中, 并将 c_j 从 C 中移除。重复上述过程。

算法 3 MC//聚类合并算法

输入: 所有分区簇的集合 C_{s_i}

输出: 合并簇后的集合 C'_{s_i}

```

1. for each cluster  $i \in C_{s_i}$  do
2.   for each cluster  $j \in C_{s_i}$  do
3.     If  $j \notin Region_{i+1}$  then break;
4.     if  $c_i \cap c_j > \mu$  then
5.        $c_i \bowtie c_j$ ;
6.     end if
7.   end for
8. end for
9. return  $C'_{s_i}$ .
```

令 n_1 表示分区的数量, n_2 表示所有示集合中簇的数量, m 表示当前簇的平均大小。则算法 2 的时间复杂度为 $O(n_2^2 / n_1 * m_1)$ 。由于簇的数量在可控范围内, 需要比较每个簇中的对象, 因此 MC 的时间复杂度为 $O(n^2)$ 。

图 6 给出了聚类的合并过程。其中 $Region_1$ 包含簇 c_1 , $Region_2$ 包含簇 c_2 , $Region_3$ 包含簇 c_3 。 c_1 包含对象 $\{o_1, o_2, o_3, o_4\}$, c_2 包含对象 $\{o_2, o_3, o_5, o_6, o_7\}$, c_3 包含对象 $\{o_6, o_7, o_8, o_9, o_{10}\}$ 。 $Region_1$ 与 $Region_2$ 相邻, $Region_2$ 与 $Region_3$ 相邻, $Region_1$ 与 $Region_3$ 不相邻。因此 c_1 与 c_2 相交得到共同对象 $\{o_2, o_3\}$ 。合并 c_1 与 c_2 并去除重复的对象, 合并后的簇包含对象 $\{o_1, o_2, o_3, o_4, o_5, o_6, o_7\}$ 。由于 c_1 与 c_3 在不相邻的区域中, 因此 c_1 不与 c_3 取交集。因为 c_2 属于 $Region_2$ 且 $Region_2$ 与 $Region_3$ 相邻, 因此 c_1 和 c_2 合并后与 c_3 取交集。聚类合并过程依次从 $Region_1$ 到 $Region_3$ 扫描每个区域中的簇, 将有相同元素的簇依次合并, 合并后的簇与剩余的簇取交集, 并将拥有相同对象的簇合并。

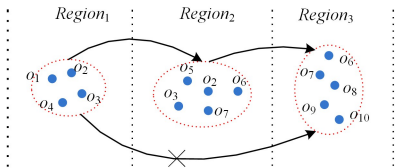


图 6 MC 算法

Fig. 6 MC algorithm

6 伴随模式挖掘

从大规模的轨迹数据中挖掘满足条件的伴随模式是一项费时的工作。在分布式聚类阶段产生成百上千个簇, 挖掘伴随模式需要遍历所有簇的所有对象, 若设计分布式方案相比

单机处理又存在较大的通信开销。随着时代的进步, 计算机处理器从单核单进程逐渐扩展到多核多进程。本节根据当前计算机多核多线程的特性, 设计了多线程伴随模式挖掘算法。算法的主要思想如下。

从节点的聚类结果汇集到主节点, 将具有相同对象的簇合并, 当前时间的簇与候选伴随取交集, 当交集的结果达到阈值 δ_s 被存储为候选伴随。一旦候选伴随的周期超过阈值 δ_t , 会被立即输出为伴随对象。同时, 一旦候选的规模小于数量阈值 δ_s , 就不再是合格的候选伴随, 应该从内存中删除。在候选和当前时间交叉阶段, 设计了并行方式, 使得候选伴随分别与当前时间的簇取交集; 采用线程池控制并行度, 同时为当前簇添加同步锁, 使得一个簇在同一时刻只被一个候选伴随取交集, 保证交叉阶段的准确性。

定义 14 (PCPM) 令 r 表示候选伴随, δ_s 表示阈值大小, 如果 r 中存在超过 $size(r) - \delta_s$ 的对象并出现在交叉簇中, 则 r 和剩余的簇交叉不会产生任何大于 δ_s 的结果。

定义 15 (封闭候选) 对于一个候选伴随 r_i , 如果不存在任何候选 r_j , 使得 $r_i \subseteq r_j$ 且 r_i 的周期小于 r_j 的周期, 则 r_i 是一个封闭候选。

算法 4 列出了伴随模式挖掘阶段算法 PCPM。序号 1 初始化新的候选伴随集合 R' , R' 的初始值为空。序号 2-序号 16 通过取交集来增加候选伴随, 其中序号 3 创建多线程, 加快取交集的速度, 序号 5-7 使用定义 13 从候选集中删除相交对象, 并在下一个相交前检查其剩余大小, 提前结束数量小于阈值的候选伴随与簇的交叉过程。序号 8 及时删除已经取过交集且剩余大小为 $r.size - \delta_s$ 的候选伴随。序号 9-序号 10 将交叉后满足阈值 δ_s 的结果存储在候选伴随模式集合中, 在存储过程中为集合 R' 加锁, 防止两个线程同时写入导致数据错误。序号 11-13, 如果候选伴随模式集合满足时间阈值则被实时输出。序号 18-20 使用定义 14 检查是否有一个包含相同对象但持续周期较长的候选 r_i , 如果没有, 则将 c_j 添加到候选伴随集合中。

算法 4 PCPM //并行的伴随模式挖掘算法

输入: 轨迹数据流 S , 对象数量阈值 δ_s , 时间周期阈值 δ_t , 候选簇集合 R

输出: 每一组满足条件的伴随集合 r'

```

1. 初始化新的候选伴随集合  $R'$ ;
2. for each  $r_i \in R$  do
3.   创建线程
4.   for each  $c_j \in C$  do
5.     If  $r_i.size < \delta_s$  then break;
6.     创建一个新的候选伴随  $r_i' \leftarrow r_i \cap c_j$ ;
7.      $r_i'.duration = r_i.duration + s.duration$ ;
8.     从  $r_i$  中移除已经交叉的对象;
9.     if  $r_i'.size \geq \delta_s$  then
10.      为  $R'$  加锁并将  $r_i'$  添加到  $R'$  中;
11.      if  $r_i'.duration \geq \delta_t$  then
12.        输出  $r_i'$ ;
13.      end if
14.    end if
15.  关闭线程
16. end for
17. for each  $c_j$  do
18.   If  $c_j$  是封闭候选 then
19.     $c_j$  添加到  $R'$ ;
```


20. end if
21. end for
22. $R \leftarrow R'$.

令 n_1 表示对象的数量, n_2 表示候选伴随集合 R 的大小, 算法 4 的时间复杂度为 $O(n_1 * n_2)$ 。因此 DCPFS 算法的时间复杂度为 $O(n_1^2 + n_1 * n_2)$

证明: 在聚类阶段, 算法需要 $O(n_1^2)$ 的时间复杂度去执行密度聚类。在聚类合并阶段, MC 的时间复杂度为 $O(n_1^2)$ 。在交叉阶段, 假设 m_1 是簇的数量, m_2 是候选伴随的数量, l_1 表示簇的平均大小, l_2 表示候选簇的平均大小, 则单个交叉任务的时间复杂度为 $l_1 * l_2$ 。假如 $m_1 * l_1 = n_1, m_2 * l_2 = n_2$, 因此交叉阶段的时间复杂度为 $O(n_1 * n_2)$, 总的时间复杂度为 $O(n_1^2 + n_1^2 + n_1 * n_2)$, 由于包含修剪策略和并行算法, 因此可以节省大约 50% 的时间。

图 7 给出了交叉的过程。假设每个快照包含 1 个时间戳, 大小阈值 δ_s 为 3, 时间阈值 δ_t 为 4。当快照 s_1 到达, s_1 中的两个簇作为候选伴随, 即 r_1 和 r_2 。快照 s_2 到达产生新簇并与 s_1 取交集, 产生新的候选伴随 r_3 。快照 s_3 到达, 由于 r_2 与快照 s_3 取交集, 被剪枝为 $\{o_8, o_9, o_{10}\}$ 。快照 s_4 到达, 由于 r_2 与快照 s_4 取交集, 小于大小阈值 δ_s , 因此 r_2 被删除。最后, 由于 r_1 满足时间阈值 δ_t , 算法输出 $\{o_1, o_2, o_3, o_4\}$, 在此过程中 R 最多包含 19 个对象。

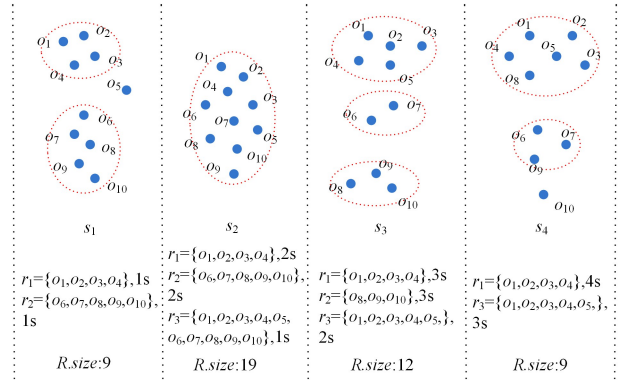


图 7 coMin 算法的交叉过实验

Fig. 7 Crossover process of coMin algorithm

7 实验

7.1 实验数据

本文使用两组真实场景的数据集。
Taxi^[15-16]: 该数据集为成都市 2014 年 8 月 3 日 13640 辆出租车超过 2 千万条 GPS 记录, 其中剔除了从 0 点到凌晨 6 点之间的数据。
GeoLife^[17-18]: 该数据集保存了 182 名用户在 2008 年 4 月 1 日的旅游记录。对于每个用户, 收集 GPS 信息。

表 2 轨迹数据集信息

Table 2 Trajectory dataset information

Dataset	Number of objects	Number of timestamps	Number of trajectory points
GeoLife	182	17 278	76 220
Taxi	13 640	2 160	27 363 616

7.2 实验环境

本实验运行在拥有 5 个节点的 Flink^[19] 集群上, 每个

节点的操作系统版本为 CentOS7.0, 处理器为 6-core Inter Xeon CPU Gold 5117CPU @2.00GHz, RAM 为 20 GB, 磁盘容量为 400 GB。Java 虚拟机采用 JDK1.8, 分布式版本为 Flink1.7.2, 并搭建在 Hadoop2.7.2 的 Yarn 上。所有代码都是用 Java 语言编写, 使用编辑器为 IDEA2020.1。使用 Maven4.0.0 配置相关依赖, 并打包上传到 Flink 集群。

7.3 数据预处理

本文对原始数据集的时间戳重新编号, 使时间戳从 1 开始, 并设置固定采样率为 30 秒/次。在处理缺失数据时, 使用线性插值法填充缺失值, 并删减掉比固定频率小得多余数据。根据同一 ID 当前时间与下一个时间点的相连向量计算与正北方向的夹角 ($0^\circ \sim 360^\circ$)。然后将数据集进行离线地理划分。处理后的数据集包含如下信息: 用户 ID、纬度、经度、时间戳、角度、分区, 如 (6136, 30.663005, 104.089374, 810, 202, 11)。

7.4 实验对比及分析

7.4.1 DCPFS 参数设置

文本设置了多个参数, 用来评估各参数对 DCPFS 性能的影响并对各参数进行试验验证。参数设置如表 3 所列, 其中加粗字体为默认值。

表 3 实验参数设置

Table 3 Experimental parameter setting

Parameter	Taxi	GeoLife
<i>eps</i>	10, 30 , 50, 70, 90	30, 50 , 70, 100, 200
<i>minPts</i>	2, 8 , 15, 25, 40	5, 8 , 12, 16, 20
<i>duration</i>	5, 10 , 30, 50, 80	60, 80 , 100, 120, 140
<i>angle</i>	5, 10 , 15, 30, 60	5, 10 , 20, 30, 40

其中, *eps* 表示最小聚类半径, *minPts* 表示最小聚类数量, *duration* 表示伴随模式的最小周期, *angle* 表示角度阈值。

为了更加准确地评估算法在每个快照的时间效率, 本文采用平均快照处理时间的方式。伴随模式挖掘框架在每个快照的平均快照处理时间的计算式如下:

$$Time = \frac{\text{数据集的伴随模式挖掘总时间}}{\text{快照数}}$$

7.4.2 DCPFS 与 Baseline 的性能对比

在实验之前, 首先使用采样点数据测试 DCPFS 的挖掘数量与 Baseline^[10] 方法得到挖掘结果数量一致, 证实了 DCPFS 计算结果的正确性。

在实时应用的场景中, 快照可能在 1~2 s 内被送达。为了满足实时性的要求, 伴随模式的挖掘需要在下一个快照被送达之前及时响应。为了检测本文提出的分布式方法的高效性, 将本文方法与 Baseline 方法进行了比较。其中, 在 Geolife 中, 17 278 个快照中分布 76 220 个轨迹点, 平均每个快照包含约 5 个轨迹点; 在 Taxi 中, 2 160 个快照中分布 27 363 616 个轨迹点, 平均每个快照中包含约 12 668 个轨迹点。Geolife 数据较 Taxi 更加稀疏, 因此仅需要极短的时间就可以作出响应。

图 8(a)、图 8(b) 给出了 *eps* 的变化对伴随对象发现效率的影响。在图 8(a) 中, DCPFS 比 Baseline 方法减少了约 75% 的时间消耗。可以看出, DCPFS 和 Baseline 对 *eps* 的值较敏感, 当 *eps* 为 100 m 时, DCPFS 和 Baseline 可以在最短的时间内作出响应, 当 *eps* 的值为 200 时, Baseline 的响应时间比 *eps*

的值为 100 时增加了 2 倍。这是由于在 GeoLife 数据集中, 旅行团队一般在较近的范围内活动, 而旅行者分别会参与到不同的游玩项目中, 相互之间又有一定的距离。 eps 在 GeoLife 的合理设置可以有效减少不必要的时间消耗。在图 8(b) 中可以看到, 随着 eps 的值增大, Baseline 的响应时间逐渐增加, eps 的值增大到 90 m 时, Baseline 需要 7 000 ms 的时间作出响应, 这远远超出了实时响应的时限。随着 eps 的增加, DCPFS 的处理时间缓慢增加, 但依然可以在 1 s 内作出响应。这是由于 Taxi 数据集来自真实的交通车辆数据, 对象密度较大, 而 Baseline 在单节点难以处理, 从而导致严重的超时现象。

图 8(c)、图 8(d) 给出了 $minpts$ 的变化对伴随对象发现效率的影响。在图 8(c) 中可以看到, 当 $minpts$ 的值较小时, DCPFS 比 Baseline 方法减少了约 70% 的时间消耗, 随着 $minpts$ 值的增大, 当 $minpts$ 的值为 20 时, DCPFS 比 Baseline 方法仅减少了约 50% 的时间消耗, 这是因为在 ADBSCAN 阶段, $minpts$ 的值较小时会得到小而多的簇, 随着 $minpts$ 值的增大会得到大而少的簇, 导致多个簇中的对象属于不同的分区, 增加了聚类合并阶段的时间消耗。在图 8(d) 中可以看到, Taxi 数据集中当 $minpts$ 的值为 2 时, Baseline 大约需要 23 000 ms 作出响应, 远远超出了实时伴随模式挖掘的时间要求。这是因为在 ADBSCAN 阶段, $minpts$ 的值为 2 时会挖掘大量无用的小簇, 带来大量的时间开销。然而 DCPFS 将小簇分布在多个节点, 有效地分散了主节点的压力, 说明 DCPFS 在 $minpts$ 变化时具有很好的稳定性。DCPFS 相比 Baseline 方法效率提升了约 50%, 因为大量冗余的小簇使得聚类合并

和交叉阶段的时间消耗增加, 降低了 DCPFS 的效率。

图 8(e)、图 8(f) 给出了 $angle$ 的变化对伴随对象发现效率的影响。图 8(e) 中, Baseline 和 DCPFS 对 $angle$ 的变化不敏感。这是因为, 无论角度设置值怎样变化, ADBSCAN 依然需要扫描整个快照内的对象。相反, 通过设置不同的角度值可以发现更符合真实场景的簇。总而言之, 在 GeoLife 数据集上, DCPFS 比 Baseline 平均减少了约 75% 的时间消耗。图 8(f) 中, 当 $angle$ 的值为 5 时, DCPFS 的时间效率仅仅比 Baseline 提升了约 30%, 这是因为当 $angle$ 的值为 5 时, ADBSCAN 阶段由于角度值设置过小, 会产生较多的小簇, 在 DCPFS 中较多的小簇会增加聚类合并阶段和模式挖掘阶段的时间消耗, 进而影响 DCPFS 的整体性能。从总体性能看, $angle$ 的变化对 Baseline 影响较小。在 Taxi 数据集上, 除了因 $angle$ 值过小引起的小簇现象, DCPFS 比 Baseline 减少了约 70% 的时间消耗。

图 8(g)、图 8(h) 给出了 $duration$ 的变化对伴随对象发现效率的影响。图 8(g) 中, 当 $duration$ 的值大于 80, 随着 $duration$ 值的增加, Baseline 的响应时间逐渐缩短, 算法的时间消耗逐渐降低, 这是因为随着 $duration$ 值的增大, 一些较短的伴随模式会被剪枝, 挖掘出更长的伴随模式, 使得算法的时间性能提升。总而言之, 在 GeoLife 数据集上, 不同的 $duration$ 下 DCPFS 比 Baseline 平均减少了约 70% 的时间消耗。在图 8(h) 中可以看到, Baseline 和 DCPFS 对 $duration$ 的变化在 Taxi 数据集上不敏感, 这是由于 Taxi 数据集中包含较多的长轨迹, 较小的 $duration$ 对长轨迹依然能作出响应。

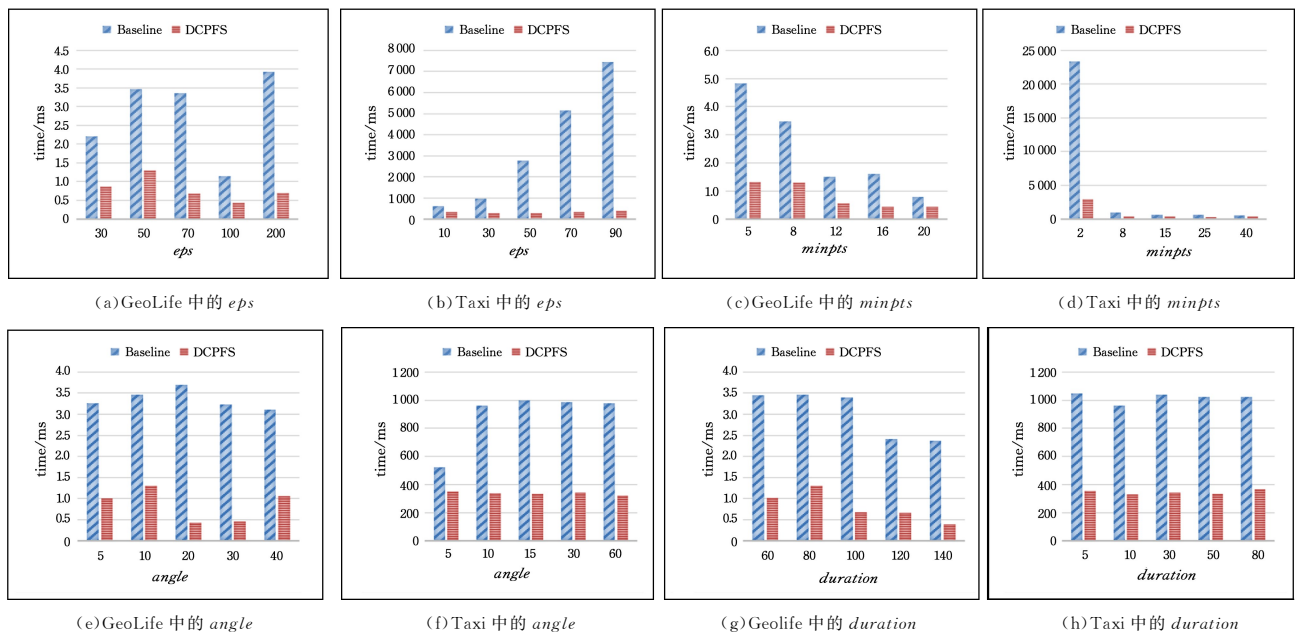


图 8 DCPFS 对伴随模式挖掘性能的评估

Fig. 8 Evaluation of companion pattern mining performance by DCPFS

7.4.3 DCPFS 与 TCompanion 的性能对比

本节对比了 DCPFS 与 TCompanion^[16] 的性能。TCompanion 是基于轨迹槽模型的分布式伴随模式挖掘框架, 采用微批处理的思想。TCompanion 有 3 种度量方式, 分别为 Point-to-Point, Point-to-Polyline, Polyline-to-Polyline。为了

方便与 DCPFS 对比, 这里我们设置 TCompanion 的度量方式为 Point-to-Point 度量, 称为 TCompanion-P2P。在 GeoLife 和 Taxi 中的默认值分别为 $eps=50$, $minpts=8$, $angle=10$, $duration=80$ 和 $eps=30$, $minpts=8$, $angle=10$, $duration=10$ 。

图 9(a)、图 9(b)给出了与 TCompanion-P2P 时间效率的对比。可以看到在 GeoLife 和 Taxi 上 DCPFS 处理每个快照的时间要优于 TCompanion-P2P,这是由于与 TCompanion-P2P 使用槽点的微批处理相比,DCPFS 不受槽点设置的限制,在伴随模式挖掘中更加灵活。

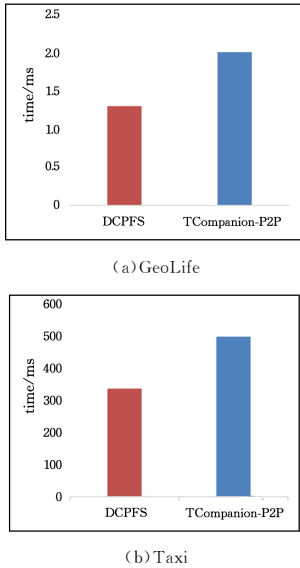


图 9 DCPFS 与 TCompanion-P2P 性能对比

Fig. 9 Performance comparison between DCPFS and TCompanion-P2P

7.4.4 分区数量对 DCPFS 效率的影响

本节测试了分区数量对 DCPFS 的影响。在 DCPFS 中,相同分区内的数据会被发送到相同的节点去处理,因此分区数量与节点数量的关系影响着伴随模式挖掘的效率。本节采用 5 个分布式节点,通过调节轨迹分区数量,测试分区数量与节点数量之间的关系。

图 10(a)、图 10(b)给出了分区数量对 DCPFS 效率的影响,这里设置模式节点数为 5。

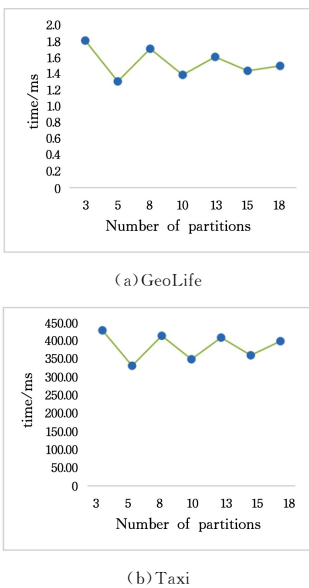


图 10 分区数量对 DCPFS 效率的影响

Fig. 10 Effect of partition number on DCPFS efficiency

如图所示,当分区数量小于节点数量时产生空闲节点,5 台节点仅有 4 台在处理数据。当分区数量大于节点数量

时,其中一台或几台节点会处理 2 倍于其他节点的数据,产生数据倾斜,使负载不均衡。当分区数量与节点数相等或者是节点数的倍数时,系统达到负载均衡,时间消耗最低。

结束语 本文针对轨迹流伴随模式挖掘实时响应问题,提出了一种基于分布式的轨迹流伴随模式框架。首先对该框架的整体结构进行了介绍;其次列举了轨迹数据流和伴随模式的相关定义;然后根据数据的地理分布设计了数据分区策略,在分布式阶段设计了融合角度的密度聚类算法。为了解决分布式聚类中数据分区导致的聚类结果缺失问题,设计了聚类合并算法对聚类结果进行合并,并设计了剪枝策略;在模式挖掘阶段设计了并行挖掘方法,有效地提高了模式挖掘的效率。最后使用真实世界的数据集对运行效率进行了评估。实验结果表明,DCPFS 在轨迹流的伴随模式挖掘相比传统算法时间消耗大大降低,可以满足大规模轨迹数据实时响应的需求。下一步工作将研究实时的分布式数据分区问题,保证聚类阶段高效的实时负载均衡,提升挖掘效率。

参考文献

- [1] ZHENG Y. Trajectory Data Mining: An Overview[M]. ACM, 2015.
- [2] BENKERT M, GUDMUNDSSON J, HUBNER F, et al. Reporting Flock Patterns[J]. Computational Geometry, 2010, 41(3): 111-125.
- [3] JEUNG H, SHEN H T, ZHOU X. Convoy queries in spatio-temporal databases[C]// IEEE 24th International Conference on Data Engineering, New York, IEEE, 2008: 1457-1459.
- [4] JEUNG, YIU H, MAN L, et al. Discovery of Convoys in Trajectory Databases[J]. Computer Science, 2010, 1(1): 1068-1080.
- [5] LI Z, DING B L, HAN J W, et al. Swarm: Mining relaxed temporal moving object clusters[J]. Proceedings of the Very Large Database Endowment, 2010, 3(1/2): 723-734.
- [6] LI Y, BAILEY J, KU L. Efficient mining of platoon patterns in trajectory databases[J]. Data & Knowledge Engineering, 2015, 100: 167-187.
- [7] ORAKZAI F, CALDERS T, Pedersen T B. Distributed convoy pattern mining[C]// Proc. of the 17th Int. Conf. on Mobile Data Management, Piscataway, NJ: IEEE, 2016: 122-131.
- [8] ZHANG J W, LIU S J, YANG Q. DMFUCP: A Distributed Mining Framework for Universal Companion Patterns on Large-Scale Trajectory Data[J]. Journal of Computer Research and Development, 2022, 59(3): 647-660.
- [9] VIEIRA M R, BAKALOV P, TSOTRAS V J. On-line discovery of flock patterns in spatio-temporal data[C]// Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, Washington, USA, 2009: 286-295.
- [10] TANG L A, ZHENG Y, YUAN J, et al. On Discovery of Traveling Companions from Streaming Trajectories[C]// IEEE 28th International Conference on Data Engineering, Washington, 2012: 186-197.
- [11] LI X, CEIKUTE V, JENSEN C S, et al. Effective Online Group Discovery in Trajectory Databases[J]. IEEE Transactions on Knowledge & Data Engineering, 2013, 25(12): 2752-2766.
- [12] LAN R, YUY, CAO L, et al. Discovering Evolving Moving Object Groups from Massive-Scale Trajectory Stream[C]// IEEE

International Conference on Mobile Data Management. IEEE, 2017:256-265.

[13] NASERIAN E, WANG X H, XU X L, et al. A Framework of Loose Travelling Companion Discovery from Human Trajectories [J]. IEEE Transactionson Moblie Computing, 2018, 17 (11): 2497-2511.

[14] SHEIN T T, PUNTHEERANURAK S, IMAMURA M. Discovery of evolving companion from trajectory data streams[J]. Knowledge and Information Systems, 2020, 62:3509-3533.

[15] ZHENG Y, ZHANG L, XIE X, et al. Mining interesting locations and travel sequences from GPS trajectories[C]//Proc. of Int. Conf. on World Wild Web. New York: ACM, 2009: 791-800.

[16] XIAN Y, LIU Y, XU C, et al. Parallel Discovery of Trajectory Companion Pattern and System Evaluation[J]. 2020, 28(10): 538-550.

[17] ZHENG Y, CHEN Y, XIE X, et al. Understanding mobility based on GPS data[C]// Proceedings of ACM Conference on Ubiquitous Computing. New York: ACM, 2008: 312-321.

[18] ZHENG Y, XIE X, MA W. GeoLife: A collaborative social networking service among user, location and trajectory[J]. IEEE

Data Engineering Bulletin, 2010, 33(2): 32-39.

[19] TILMANN R, JONAS T, ASTERIOS K, et al. Apache Flink in current research[J]. 2016, 58(4): 157-165.

[20] YUAN J, ZHENG Y, ZHANG C. et al. T-Drive: Driving Directions Based on Taxi Trajectories[C]// Proceedings of the 18th Annual ACM International Conference on Advances in Geographic Information Systems. ACM, 2010. 99-108.



ZHANG Kang-wei, born in 1995, post-graduate. His main research interests include large-scale trajectory data stream pattern mining and analysis.



SHAN Mei-jing, born in 1979, Ph.D, as-sociate professor. Her main research interests include big data analysis and information security.