软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(10):3184−3204 [doi: 10.13328/j.cnki.jos.005418]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

近似到达时间约束下的语义轨迹频繁模式挖掘*

吴 瑕[1,2], 唐祖锴[3], 祝园园[1], 彭煜玮[2], 彭智勇[2]

[1](软件工程国家重点实验室(武汉大学),湖北 武汉 430072)

[2](武汉大学 计算机学院,湖北 武汉 430072)

[3](武汉理工大学 计算机科学与技术学院,湖北 武汉 430070)

通讯作者: 彭智勇, E-mail: peng@whu.edu.cn

摘 要: 随着 GPS 定位技术的不断发展与智能移动设备的普及,轨迹数据的获取变得越来越容易,同时,轨迹数据相关应用的需求也逐渐增多.在轨迹数据上加入语义信息,可以得到体积较小、质量较高、能够更好地反映用户行为的语义轨迹,在其上实现旅游线路推荐、路线预测、用户生活模式挖掘、朋友推荐等应用,可以更好地满足用户需求.挖掘语义轨迹的频繁模式是实现这些应用的技术基础,而在很多情况下,用户对语义轨迹频繁模式常存在到达时间方面的需求,比如按特定时间游玩热门景点的同时需要按时到达车站候车.现有的语义轨迹模式挖掘方法大多没有考虑到达时间的约束,挖掘出的频繁模式缺少到达时间信息;少数方法考虑了精确的到达时间,但因为约束太强会导致无法挖掘到频繁的模式.因此,首次对近似到达时间约束下的语义轨迹频繁模式(approximate arrival-time constrained frequent pattern,简称 AAFP)挖掘方法进行了研究,并给出了其形式化定义;通过时间轴划分提出了挖掘 AAFP 的基线算法,并通过建立索引 AAP-tree 提出了改进后的高效、灵活的 AAFP 挖掘算法;之后提出了信息熵增量公式,并给出了时间轴划分及 AAP-tree 的高效维护方法;最后在真实数据集上进行实验,验证了方法的有效性及高效性.

关键词: 轨迹数据,语义轨迹,近似到达时间,轨迹频繁模式,频繁模式挖掘

中图法分类号: TP311

中文引用格式: 吴瑕,唐祖锴,祝园园,彭煜玮,彭智勇.近似到达时间约束下的语义轨迹频繁模式挖掘.软件学报, 2018,29(10): 3184−3204. http://www.jos.org.cn/1000-9825/5418.htm

英文引用格式: Wu X, Tang ZK, Zhu YY, Peng YW, Peng ZY. Frequent pattern mining with approximate arrival-time in semantic trajectories. Ruan Jian Xue Bao/Journal of Software, 2018,29(10):3184−3204 (in Chinese). http://www.jos.org.cn/1000-9825/5418.htm

Frequent Pattern Mining With Approximate Arrival-Time in Semantic Trajectories

WU Xia[1,2], TANG Zu-Kai[3], ZHU Yuan-Yuan[1], PENG Yu-Wei[2], PENG Zhi-Yong[2]

[1](State Key Laboratory of Software Engineering (Wuhan University), Wuhan 430072, China)

[2](Computer School, Wuhan University, Wuhan 430072, China)

[3](School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China)

Abstract: Along with the development of the GPS positioning technology and smart mobile devices, more and more trajectory data are collected continuously every day. Thus, managing and mining useful information from these trajectories is critical in many application areas. Compared with raw trajectory data, semantic trajectory data equipped with semantic information has better quality, less volume and

*基金项目: 科技部国家重点研发计划(2016YFB1000700); 国家自然科学基金(61502349)

Foundation item: Ministry of Science and Technology of China, National Key Research and Development Program (2016YFB 1000700); National Natural Science Foundation of China (61502349)

收稿时间: 2017-05-07; 修改时间: 2017-08-25; 采用时间: 2017-09-25; jos 在线出版时间: 2018-03-13

CNKI 网络优先出版: 2018-03-13 17:17:54, http://kns.cnki.net/kcms/detail/11.2560.TP.20180313.1717.002.html

higher description ability, and thus it can be used in many applications such as trip recommendation, next location prediction, life pattern understanding, and friend recommendation. Mining frequent pattern in semantic trajectories is the fundamental problem in above tasks. In many circumstances, users may have the requirements on the arrival-time, e.g., users may want to visit a popular view spot at a certain timestamp and then arrive the railway station on time. Most of existing approaches on semantic trajectory pattern mining do not consider the arrival-time, and only a few existing approaches take the accurate arrival-time as the constraint, but they can barely find frequent patterns under such a strict time constraint. This paper, for the first time, studies the approximate arrival-time constrained frequent pattern (AAFP) mining problem. First, a baseline algorithm of mining AAFP is given by dividing the time axis into intervals. Then, an improved flexible algorithm is proposed to significantly improve the efficiency based on the AAP-tree index. Finally, a strategy to maintain the AAP-tree and the set of time axis partitions is introduced based on incremental information entropy. The experimental results on real trajectory datasets validate the effectiveness and efficiency of the proposed algorithms.

Key words: trajectory data; semantic trajectory; approximate arrival-time; trajectory frequent pattern; frequent pattern mining

随着 GPS 定位技术的不断发展与智能移动设备的普及,轨迹数据的获取变得越来越容易,同时轨迹数据相关应用的需求也逐渐增多.轨迹数据(trajectory data)具有数据量大、更新频率快、价值密度低等特点,会导致基于轨迹的挖掘、查询等效率低,效果不理想.若在轨迹数据上加入语义信息,则可得到体积较小、质量较高、能够更好地反映用户行为的语义轨迹(semantic trajectory).因此,近年来不少学者开始关注基于语义轨迹的研究,并在其上实现旅游线路推荐、路线预测、用户生活模式挖掘、朋友推荐等应用,以更好地满足用户需求.

语义轨迹的频繁模式挖掘(frequent pattern mining)是实现这些应用的技术基础[1],例如,一位游客希望系统为他推荐热门旅游路线,系统将频繁模式{景点→美食街→火车站,0.12}推荐给游客,表示模式“景点→美食街→火车站”在历史轨迹集中出现的概率为 12%,是一条热门路线.然而,在很多情况下,用户对语义轨迹模式常存在到达时间方面的需求,比如,用户需要在上午 10:00 游览景点、中午 12:00 到达饭店且下午 16:00 到达火车站,同时希望推荐的线路是热门线路,则需要挖掘出包含到达时间的语义轨迹模式以适应此类应用的需求.比如,模式{⟨景点,10:00⟩→⟨美食街,12:00⟩→⟨火车站,16:00⟩,0.12},我们称这样的模式为在到达时间约束下的语义轨迹频繁模式(arrival-time constrained frequent pattern,简称 AFP).

目前,现有的大多语义轨迹频繁模式挖掘方法无法挖掘出 AFP,因为一部分方法[2,3]完全没有考虑时间,而另一部分方法考虑[4−7]的是行程时间(travel-time),即从一个地点转移到另一个地点所花费的时间,并没有考虑具体到达某个地点的时刻.文献[8]虽然提供了一种可以用于挖掘 AFP 的方法,但是这种方法精确地将到达时刻考虑到语义轨迹模式中,基本无法挖掘到频繁的模式.如图 1 所示,给定 5 条历史语义轨迹,设频繁度为 0.05(在历史轨迹集中出现的概率大于 5%就认为是频繁的),用传统频繁模式挖掘方法可以挖掘到若干频繁模式,但是这些频繁模式都没有到达时间信息;若精确地考虑到达时间,使用文献[8]的方法挖掘 AFP,则挖掘不到频繁度高于 0.05 的 AFP,因为在文献[8]给出的方法中,只要到达时刻不一致,即为不同的模式,如“⟨景点,10:10⟩→⟨美食街,11:55⟩”与“⟨景点,9:50⟩→⟨美食街,12:20⟩”是两个不同的模式.

在现实生活中,用户希望“12:00 去吃饭”,其实并不是要求非常准确的“12:00 整”到达饭店,“12:00 左右”更符合用户的实际需求,因此,本文将考虑语义轨迹在近似到达时间约束下的频繁模式(approximate arrival-time constrained frequent pattern,简称 AAFP)挖掘.如图 1 所示,AAFP 方法能够挖掘出 AFP 方法不能挖掘出的频繁模式,同时,近似地满足用户对到达时间的要求.比如,本例中 AAFP 方法认为“⟨景点,10:10⟩→⟨美食街,11:55⟩”“⟨景点,9:50⟩→⟨美食街,12:20⟩”与“⟨景点,10:05⟩→⟨美食街,12:30⟩”是同一个 AAFP:{⟨景点,10:05⟩→⟨美食街,12:30⟩,0.07}.

然而,考虑近似到达时间后会带来一些挑战.首先,如何合理地划分时间轴在实现“近似到达时间”的同时使其不破坏数据分布特征;其次,语义轨迹频繁模式挖掘效率通常是非常低的[9,10],考虑到达时间后会使基础频繁项增多,使其挖掘效率更低,如何保证挖掘效率是一个难点;最后,在新数据到来后,会改变数据的分布特征,同时改变时间轴划分,从而改变语义轨迹 AAFP 的挖掘结果,如何维护时间轴划分以及挖掘结果也是需要解决的问题.为了解决这些问题,本文首先使用信息熵聚类方法将语义轨迹集中各个地点的时间轴合理地划分开,并提出了挖掘语义轨迹 AAFP 的基线算法.之后,为了改进基线算法,使其更高效、更灵活,本文建立了一个多层混合索

引 AAP-tree,并给出了基于其上的语义轨迹 AAFP 挖掘算法.然后,针对新数据到来后的维护问题,提出了时间轴划分及 AAP-tree 的高效维护方案.最后,在真实数据集上进行实验,并分析了本文方法的效果与效率.

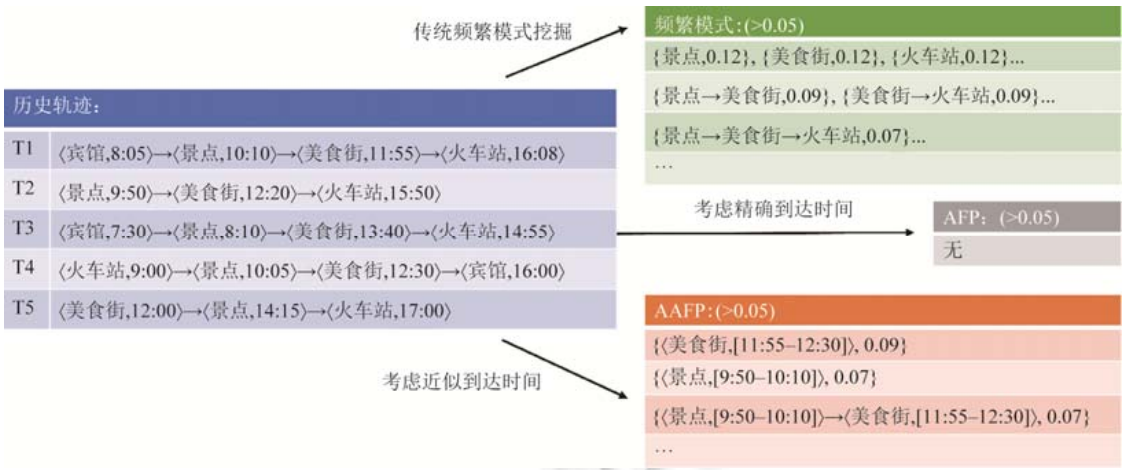


Fig.1 Semantic trajectory frequent pattern mining

图 1 语义轨迹的频繁模式挖掘

本文主要有以下贡献:

(1) 首次研究并形式化定义了语义轨迹的 AAFP;

(2) 通过时间轴划分提出了挖掘 AAFP 的基线算法,并通过建立索引 AAP-tree 提出了改进后的高效、灵活的 AAFP 挖掘算法;

(3) 提出了信息熵增量公式,并给出了时间轴划分及 AAP-tree 的高效维护方法;

(4) 通过在真实数据集上进行实验,验证了本文方法的有效性及高效性.

本文第 1 节介绍相关工作.第 2 节对所解决的问题给出形式化的定义.第 3 节介绍语义轨迹 AAFP 的挖掘方法.第 4 节介绍时间轴划分及 AAP-tree 的维护方法.第 5 节介绍本文方法在真实数据集上的实验结果.第 6 节对本文工作进行总结与展望.

1 相关工作

轨迹数据的研究一直是研究者关注的热点,其中,语义轨迹因比普通轨迹具有更小的体积和更高的数据质量于近年来得到更多的关注.一些研究给出了将裸轨迹转化为语义轨迹的方法,包括文献[5,6,11,12]等,其中,文献[11]提出了丰富轨迹数据语义信息过程的模型.文献[12]提出了一种基于地物范围与时间区间的语义发现方法,当一个物体在某个地物范围内停留的时间超过一定阈值(其中,时间阈值与地物范围的大小成正比)时,就认为这个物体在此处有停留,这个地物范围即被称为一个停留点(stop),一系列的停留点就组成一条语义轨迹.文献[4,5]提出了一种基于轨迹走向与采样点密度的判定停留点的方法,给定一个半径与时间区间,若在圆形区域中的点密度达到给定阈值,则判定为一个停留点,该方法中停留点的判定与具体的地物范围无关.

文献[13–16]等提出了针对普通的轨迹模式的挖掘方法,通常是通过定义距离公式,将轨迹集中的轨迹根据距离聚类,得到密度高的区域作为一个聚集,轨迹在聚集之间转移的模式即为普通轨迹的模式.文献[17–19]等提出了基于路网的轨迹模式挖掘方法,首先将普通轨迹映射到路网上,根据轨迹在路网上的点之间转移的模式即可挖掘出路网轨迹的模式.由于语义轨迹是在停留点之间转移,故上述的这些方法都不适用于语义轨迹.

文献[2,3]提出了一种基于空间划分的语义轨迹频繁模式挖掘方法.该方法首先将空间等分为由六边形平铺组成的网格,轨迹经过一个六边形格子,则这个格子中的地物即为这条轨迹在这个格子中的语义,轨迹由一个格子转移到另一个格子的模式即为语义轨迹的模式.然后将这些模式按照地点先后顺序索引在名为 STP-Tree

的索引中,索引的每个节点代表一个格子,每条路径代表一个模式,节点同时还记录一个概率值,表示到达这个节点的路径所代表的模式在语义轨迹集中出现的概率.最后遍历 STP-Tree 找到所有符合给定频繁度的模式即可完成语义轨迹频繁模式挖掘.文献[4,5]提出了一种基于距离聚类的语义轨迹模式挖掘方法.该方法首先根据给定的半径和时间区间找到所有停留点,将轨迹转化为语义轨迹,然后将所有停留点基于欧式距离聚类,就得到停留点密集的区域,轨迹从一个区域转移到另一个区域的模式以及转移所需要的时间即为语义轨迹的模式.该方法只给出了语义轨迹模式的挖掘方法,并未给出频繁模式的挖掘方法,但通过对算法简单的修改便可得到.文献[6]中的方法类似文献[2,3]中的方法,所不同的是,文献[6]将轨迹经过的每一个地点作为轨迹的语义,并基于这些地点建立了 *T*-pattern Tree,其结构类似 STP-Tree,所不同的是,*T*-pattern Tree 的每一条边记录了语义轨迹从一个地点转移到另一个地点所需要的时间,同样地,只需遍历 *T*-pattern Tree 找到所有符合给定频繁度的模式即可完成语义轨迹频繁模式的挖掘.文献[7]主要是解决了文献[6]中方法挖掘出的模式不够具有代表性的问题,文献[6]以地物作为模式的节点,如"健身房 A_1→电影院 B_1→咖啡厅 C_1"与"健身房 A_2→电影院 B_2→咖啡厅 C_2"被看作两个不同的模式,然而,如此细粒度的模式没有太大的意义.文献[7]中的方法则将两者的分类合并,得出形如"健身房→电影院→咖啡厅"这样的模式,文献[7]只给出了按地物分类挖掘语义轨迹模式的方法,并未给出频繁模式的挖掘方法,但是通过对算法简单的修改便可得到.上述方法中,文献[2,3]中的方法未考虑任何时间因素.文献[4–7]的方法只考虑了行程时间,并未考虑到达时间.

文献[8]提出了一种考虑了到达时间的语义轨迹模式挖掘方法,其首先将单条轨迹上的每个点根据到达时间聚类,将到达时间间隔短的点所在的区域作为一个停留点,将这个区域中的第 1 个点的到达时间作为这个停留点的到达时间标签,然后精确地根据轨迹在不同到达时间的停留点之间转移的模式作为语义轨迹的模式,文献[8]仅给出了语义轨迹模式的挖掘方法,虽然通过简单的修改可以得到频繁模式的挖掘方法,但是如前文所述,若精确地考虑到达时间不仅不能完全符合用户需求,且基本无法挖掘到频繁的模式,因此,不能通过简单修改文献[8]中的方法得到效果较好的考虑到达时间的语义轨迹频繁模式挖掘方法.

2 问题定义

根据文献[12]中将普通轨迹转化为语义轨迹的方法,本文首先给出语义轨迹相关的定义.

定义 1(停留点(stop)). 一条轨迹从进入到离开某个地理区域范围所用的时间超过一定的阈值,则包含这个地理区域及进入离开的时间元组就称为一个停留点,记为 $s=\langle R,t_{in},t_{out}\rangle$,其中,$R=\langle \Delta R,L\rangle$表示停留点的地理区域,$\Delta R$ 是这个地理区域的几何形状,由一系列形如(x,y)的点坐标组成,L 是这个地理区域所对应的地名标签,t_{in} 是轨迹进入这个区域的时刻,t_{out} 是轨迹离开这个区域的时刻,且 $t_{out}-t_{in}\geqslant\Delta,\Delta$ 为给定的一个阈值.

定义 2(语义轨迹(semantic trajectory)). 语义轨迹是一个由 n 个停留点组成的序列,记为 $st=(s_1,s_2,\ldots,s_n)$,对任意 s_i 及 s_{i+1} 有 $t_{ini}<t_{outi}$ 且 $t_{outi}\leqslant t_{ini+1},1\leqslant i\leqslant n$,语义轨迹长度即为所包含的停留点的数量,记为$|st|=n$.由若干语义轨迹组成的集合称为语义轨迹集,记为 ST,$|ST|$为轨迹集 ST 中轨迹的数量.

定义 3(地点(location)). 一个地点 l 由一个地理区域 R 及若干停留点组成,记为 $l=\langle R,\{s_1,s_2,\ldots,s_n\}\rangle$,对$\forall s_i,s_j\in l$ 有 $s_i.R=s_j.R=l.R,1\leqslant i\leqslant n,1\leqslant j\leqslant n,n$ 为 l 中停留点的数量,一个 ST 中所有 l 组成的集合 LS 称为该 ST 的地点集合.一个地点可包含多个停留点,但一个停留点只对应一个地点.

接下来给出语义轨迹在到达时间约束下的频繁模式相关定义(若未特别说明,此后本文中提及的模式与频繁模式均为语义轨迹的模式与频繁模式).

定义 4(到达时间约束下的频繁模式 AFP). 给定语义轨迹集 ST,它的一个到达时间约束下的频繁模式是一组地理区域及到达这个区域的时间所组成的有序序列与这组序列在 ST 中出现的概率的组合,其中,概率值大于给定阈值σ,记为 $\text{AFP}=\{\langle l_1.R,s_1.t_{in}\rangle\rightarrow\langle l_2.R,s_2.t_{in}\rangle\rightarrow\ldots\rightarrow\langle l_n.R,s_n.t_{in}\rangle,p\},1\leqslant n\leqslant\max(|st_i|),st_i\subset ST,1\leqslant i\leqslant|ST|,l_j\in L,s_j\in l_j,1\leqslant j\leqslant n,l_j.R$ 为轨迹停留的地点所对应的地理区域,$s_j.t_{in}$ 为轨迹上的停留点 s_j 的到达时刻,p 为频繁度,即模式$\langle l_1.R,s_1.t_{in}\rangle\rightarrow\langle l_2.R,s_2.t_{in}\rangle\rightarrow\ldots\rightarrow\langle l_n.R,s_n.t_{in}\rangle$在 ST 中出现的概率且 $p\geqslant\sigma$.

定义 5(近似到达时间约束下的频繁模式 AAFP). 给定语义轨迹集 ST 的一个近似到达时间约束下的模式

(approximate arrival-time constrained pattern,简称 AAP)是一组地理区域及到达这个区域的时间区间所组成的有序序列与这组序列在 ST 中出现的概率的组合,记为 $2AP=\{\langle l_1.R,[t_1^{low},t_1^{up}]\rangle\rightarrow\langle l_2.R,[t_2^{low},t_2^{up}]\rangle\rightarrow\ldots\rightarrow\langle l_n.R,[t_n^{low},t_n^{up}]\rangle, p\}$,其中,$1\leqslant n\leqslant\max(|st_i|), st_i\subset ST, 1\leqslant i\leqslant|ST|, l_j\in L, 1\leqslant j\leqslant n, l_j.R$ 为轨迹停留的地点所对应的地理区域,$[t_i^{low},t_i^{up}]$为轨迹到达地点 l_j 时对应的时间段,p 为频繁度,即模式$\langle l_1.R,[t_1^{low},t_1^{up}]\rangle\rightarrow\langle l_2.R,[t_2^{low},t_2^{up}]\rangle\rightarrow\ldots\rightarrow\langle l_n.R,[t_n^{low},t_n^{up}]\rangle$在 ST 中出现的概率.给定一个阈值σ,若一个 AAP 的频繁度 $p\geqslant\sigma$,则这个 AAP 是一个语义轨迹在近似到达时间约束下的频繁模式,记作 AAFP.

最后给出本文问题定义.

问题定义. 近似到达时间约束下的频繁模式挖掘(approximate arrival-time constrained frequent pattern mining)是指,给定一个语义轨迹集 ST 和一个阈值σ,找出 ST 中所有存在且 $p\geqslant\sigma$的 AAFP.

3 近似到达时间约束下的语义轨迹频繁模式挖掘

3.1 时间轴的合理划分

如本文之前所述,用户希望“12:00 去吃饭”,其实并不是要求非常准确的“12:00 整”到达饭店,“12:00 左右”更符合用户的实际需求.若要挖掘如“12:00 左右”这样近似时间段的模式,最直接的方法就是将时间轴划分开,将“时刻”变为“时间段”,然而,划分时间轴时不能简单地每隔一段时间划分一次,这样有可能会从数据密集的位置将数据一分为二,从而破坏了数据的分布特征,影响了数据对用户行为的真实反映.图 2 表示的是用户到达餐馆的历史次数统计,横轴表示的是历史上到达的时刻,表格中数字表示历史在对应时刻到达餐馆的停留点数量,如图 2(a)所示,如果我们采用 3 个小时为粒度进行划分,则在整点的时候就将数据划分开,可见用户“12:00 前到达”与“12:00 后到达”的概率均为 50%,而实际上我们却发现,用户“11:00 前和 13:00 后到达”的概率都很小,“12:00 左右到达”的概率比较高,所以合理的划分方式如图 2(b)所示.由于历史到达次数是以采样时刻来分类的统计数据,因此,本文使用基于信息熵聚类的方法[20]对数据进行聚类并划分时间轴,以避免划分后破坏数据的分布特征.另外,对于每一个地点数据的分布特征不尽相同,比如用户到达“美食街”与“宾馆”的时间的数据分布特征不一样,故我们将分开针对每一个地点进行时间轴划分.

(a) 不合理的时间划分 (b) 合理的时间划分

Fig.2 Time axis dividing

图 2 时间轴的划分

对任意地点 $l\in L$,S_l 为 l 中包含的所有停留点的集合,$|S_l|$表示停留点数量,则 S_l 的信息熵表示为

$$Ent(S_l)=-\sum_{i=1}^{k}p(C_i)\log(p(C_i)),$$

其中,C_i 表示一个类,对应一个时刻 t_i,k 表示类的数量,$1\leqslant i\leqslant k$, $p(C_i)=\frac{|C_i|}{|S_l|}$表示停留点出现在这个类中的概率,其中,$|C_i|$表示这个类中的停留点数量,即历史上在时刻 t_i 到达 l 的停留点数量,$|C_i|\in\mathbb{N}$,$|C_i|\geqslant 0$.需要说明的是,以时间为横轴的一条语义轨迹,横轴的范围通常是一天,即 24 小时,所以 k 的大小取决于采样的时间粒度,如每隔 6 分钟采样 1 次,则 k=240,表示有 240 个类.根据不同的应用需求,轨迹采样的时间粒度不同,为了更具有广泛性,不排除一些应用可能会用到极细的时间粒度,如微秒、毫秒等,甚至更细,所以 k 的数值是可以无限大的.

假设从第 i 个类与第 i+1 个类之间将数据划分,表示为 $D(C_i,C_{i+1})$,若有 k 个类,则划分左边就包含第 1 个类到第 i 个类,同样,划分右边就包含第 i+1 个类到第 k 个类,且每两个相邻类之间都会有一个候选划分,也就是有

k–1 个候选划分.划分后的信息熵表示为

$$E(C_i,C_{i+1})=\frac{|S_l^L|}{|S_l|}Ent(S_l^L)+\frac{|S_l^R|}{|S_l|}Ent(S_l^R) \tag{1}$$

其中, $Ent(S_l^L)$ 为划分左边所有样本的信息熵, $Ent(S_l^R)$ 为划分右边所有样本的信息熵, $|S_l^L|=\sum_{n=1}^{i}|C_n|$, $|S_l^R|=|\sum_{m=i+1}^{k}|C_m|$, $1\leqslant i\leqslant k,n+m=k$.若一个划分 $D(C_i,C_{i+1})$使得公式(1)的值最小,则称这个划分为一个候选划分.

一个候选划分 $D(C_i,C_{i+1})$是一个合理的划分,当且仅当 $E(C_i,C_{i+1})$满足如下条件:

$$Gain(C_i,C_{i+1})>\frac{\log(|S_l|-1)}{|S_l|}+\frac{\Delta(C_i^t,C_{i+1}^t)}{|S_l|} \tag{2}$$

其中,

$$Gain(C_i,C_{i+1})=Ent(S_l)-E(C_i,C_{i+1}),$$

$$\Delta(C_i,C_{i+1})=\log(3^k-2)-[kEnt(S_l)-k_lEnt(S_l^L)-k_rEnt(S_l^R)].$$

k_l 表示 S_l 中类的数量,k_r 表示 S_r 中类的数量.文献[20]对条件(2)的合理性进行了证明.

类似决策树方法,递归地找到所有满足条件(2)的候选划分,则称它们组成的集合 $D(S_l)$为地点 l 在时间轴上的合理划分.我们称该方法在语义轨迹集中每个地点上应用的算法为 MDLP-L 算法,算法伪代码如下.

算法 1. MDLP-L.

输入:地点集合 L,时刻$\{C_1,C_2,\ldots,C_k\}$;

输出:时间轴合理划分集$\{D_1(S_l),D_2(S_l),\ldots,D_{|L|}(S_l)\}$.

```
1.  Main(){
2.  for each l∈L do
3.     MDLP-l(l,{C1,C2,…,Ck})        //每个地点根据各自的数据分布计算时间划分
4.  end for
5.  }
6.  return {D1(Sl),D2(Sl),…,D|L|(Sl)}  //每个地点的时间轴合理划分集合的总和,共|L|个合理划分集合
7.  MDLP-l(l,{C1,C2,…,Ck}) {          //各个地点计算时间划分的函数
8.  for i=1 to k
9.     Ci=the stop set in l where arrival time=ti
10. end for
11. Ent(Sl)=0
12. for i=1 to k
13.    Ent(Sl)=Ent(Sl)+p(Ci)·log(p(Ci))//计算总信息熵
14.    x[i]=p(Ci)log(p(Ci))           //将每个类的概率存到一个数组中以便之后降低计算量
15. end for
16.  D(Sl)=∅
17. for i=1 to k–1
18.    calculate E(Ci,Ci+1) according to formula (1)   //计算每个划分的划分信息熵
19. return Min(E(Ci,Ci+1))
20.    if E(Ci,Ci+1) satisfies equation(2)             //判断是否是合理划分
21.     D(Sl)=D(Sl)∪D(Ci,Ci+1)
22.     MDLP-l(l,{C1,C2,…,Ci}
23.       MDLP-l(l,{Ci+1,C2,…,Ci}                     //递归地找出所有划分
24.    end if
25. end for
```

26. }

算法 1 中,算法的时间复杂度主要与类的数量 k 有关.其中,步骤 8~步骤 11 是计算总信息熵 $Ent(S)$,依次计算每一个类的概率$\{p(C_1),p(C_2),\ldots,p(C_k)\}$,一共计算 k 次,所以其算法时间复杂度为 $O(N)$;步骤 13~步骤 21 是计算划分信息熵,其中,步骤 18 和步骤 19 递归地计算划分,其最坏情况下需要循环 k–1 次,每次循环需要在所有类上计算 1 次总信息熵和 1 次划分信息熵,也就是需要计算 $2k$ 次加法,因此,其算法时间复杂度是 $O(N^2)$;上述两部分的算法之间是线性关系,没有循环嵌套,所以算法总时间复杂度是 $O(N^2)$.

对任意地点 $l\in L$,完成划分之后,相邻的两个划分之间所有的类就属于一个时间段,相应的统计数信息就合并为一个统计数,即,若 $D(C_i,C_{i+1})\in D(S),D(C_j,C_{j+1})\in D(S)$,且 $D(C_i,C_{i+1})$与 $D(C_j,C_{j+1})$相邻,$1\leqslant i<j\leqslant k$,则

$$C_{i+1,j}=\sum_{s=i+1}^{j}C_s,\ |C_{i+1,j}|=\sum_{s=i+1}^{j}|C_s|.$$

$C_{i+1,j}$ 对应时间段$[t_{i+1},t_j]$.为了方便叙述,我们将划分后的地点表达为元组$(l,\{[t_1^{low},t_1^{up}],[t_2^{low},t_2^{up}],\ldots,[t_n^{low},t_n^{up}]\})$,其中,任意 $D(t_i^{up},t_{i+1}^{low})\in D(S)$. 图 3 中以餐馆为例展示了语义轨迹集中一个地点划分时间轴后,每个类由对应一个时刻变为对应一个时间段,表格中的数字表示其对应时刻或时间段的历史到达的停留点的数量.

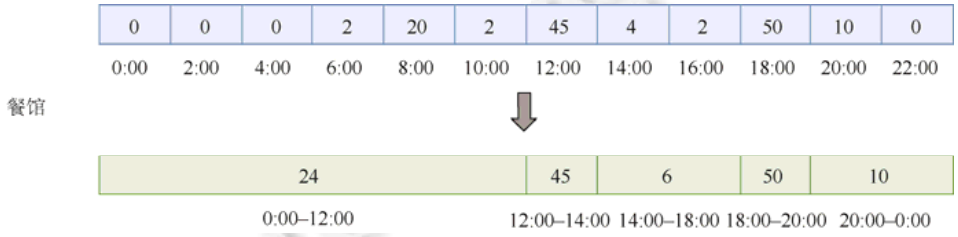


Fig.3 The data before and after time axis dividing

图 3 时间轴划分前后的数据

3.2 基线算法

得到时间轴划分后,接下来考虑如何挖掘 AAFP.我们借鉴文献[2,3,6]中的思路,首先遍历每一条语义轨迹,根据时间划分挖掘出所有存在的 AAP,然后根据 AAP 的定义将这些模式及其出现的频率用 AASTP-tree 索引起来,最后根据用户给定的阈值σ遍历 AASTP-tree,找出所有出现概率大于等于σ的模式,即可完成 AAFP 的挖掘.

对每一条语义轨迹,它所包含的 AAP 即为在其中出现过的所有 AAP 的集合.如,图 1 所示的语义轨迹 T2 就包含有“〈景点,[9:50–10:10]〉”“〈美食街,[11:55–12:30]〉”“〈火车站,[14:55–16:08]〉”“〈景点,[9:50–10:10]〉→〈美食街,[11:55–12:30]〉”“〈美食街,[11:55–12:30]〉→〈火车站,[14:55–16:08]〉”“〈景点,[9:50–10:10]〉→〈火车站,[14:55–16:08]〉”以及“〈景点,[9:50–10:10]〉→〈美食街,[11:55–12:30]〉→〈火车站,[14:55–16:08]〉”共 7 个 AAP.遍历所有语义轨迹,即可得到所有存在的 AAP 集合.

之后,在得到的 AAP 集合上构建 AASTP-tree.AASTP-tree 在 STP-tree[2]的基础上加入了近似到达时间的限制,其结构类似 STP-tree,所不同的是,每个节点的结构为形如$\langle id,l,t_{low},t_{up},p,child\rangle$的元组,其中,$id$ 为节点编号,l 为地点编号,t_{low} 是对应时间段的下界,t_{up} 是对应时间段的上界,p 是该节点对应的 AAP 在语义轨迹集中出现的概率,$child$ 记录该节点的孩子节点.如图 4 所示,AASTP-tree 的每一条存在的路径即表示一个存在的 AAP,节点的 p 即到达该节点路径所对应的 AAP 在语义轨迹集中出现的概率.例如,图 4 中虚线路径表示 AAP:{〈地点 A,[10:01–15:00]〉→〈地点 B,[9:36–14:00]〉→〈地点 A,[15:01–23:59]〉,0.02}.构建 AASTP-tree 的算法与构建 STP- tree 的算法类似,此处不再赘述.

最后,只需遍历 AASTP-tree,找出所有 p 大于阈值σ的 AAP,即可得到所有 AAFP.

本文在考虑近似到达时间后,大幅度地增加了基础频繁项的数量,如图 4 所示,对于从“地点 A”到“地点 B”原本只需考虑“地点 A→地点 B”一种情况,考虑近似到达时间后,“地点 A”被划分为 3 个时间段,“地点 B”被划分为 3 个时间段,则“地点 A→地点 B”就存在 6 种情况(排除时间上不符合先后关系的情况).也就是说,2ASTP-tree

相比 STP-tree,节点数量大为增加.与此同时,索引体积大量增加,遍历索引效率明显降低.在真实的语义轨迹数据集中,语义轨迹经过的地点数量是较多的.例如,假设一个城市包含 10 万个地点,若使用 2ASTP-tree 作为索引,索引每一层的节点数量都以 10 万倍的速度增长,即使每条轨迹平均包含 4 个地点,每个地点平均划分为 3 个时间段,最终索引的节点有$(100000\times3)^4\approx8\times10^{21}$个.所以,虽然理论上 2ASTP-tree 是可行的,而在现实中,由于语义轨迹集数据量庞大,这种穷举式的索引几乎是无法使用的.一种容易想到的改进方法是用户给定阈值σ,索引时仅索引 p 大于等于σ的 AAP.然而,这样的索引非常不灵活,若给定的阈值发生变化,则需重新建立一个全新的索引.因此,需要进一步设计更高效、更灵活的索引来支持 AAFP 的挖掘.

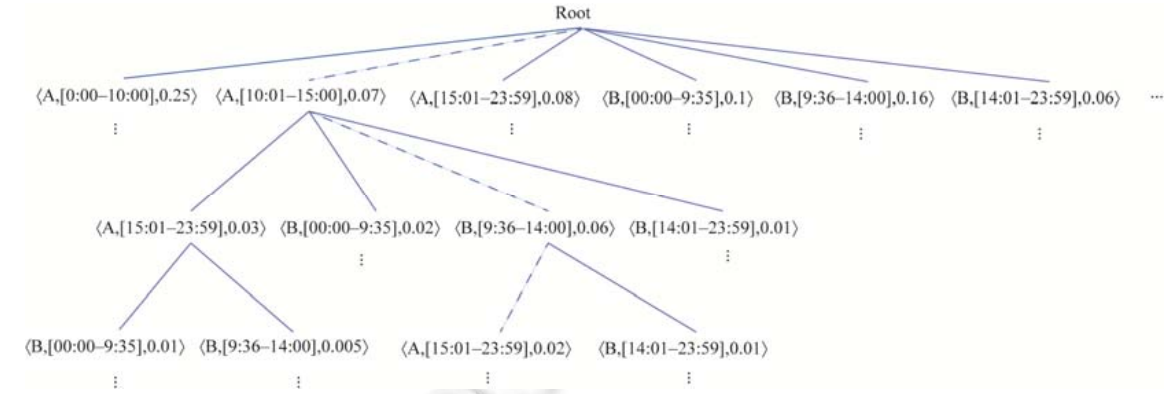


Fig.4 The structure of AASTP-tree

图 4 AASTP-tree 的结构

3.3 AAP-Tree

为了解决基线算法体积大、遍历效率低以及不灵活的问题,本文采取了一种折中的索引方式,将 AAP 的一部分信息索引起来,在挖掘时再计算剩下的一部分信息,以得到最后结果.

我们首先在所有语义轨迹数据上建立一个多层混合索引,称为 AAP-tree.AAP-tree 定义为一个三元组 $T=(N,E,Root(T))$,其中,N 是由所有节点组成的集合,E 是由所有边组成的集合,$Root(T)\in N$ 是一个虚构节点,表示树的根.树的第 1 层记录语义轨迹集中出现的每一个地点,每个节点是一个形如$\langle id,l_i,children,sum\rangle$的元组,其中,$id$ 是节点的编号,l_i 记录地点 id,$children$ 记录其孩子节点,sum 记录地点 l_i 包含的停留点数量;树的第 2 层记录每个地点的时间划分,每个节点是一个形如$\langle id,t_{low},t_{up},children,sum\rangle$的元组,其中,$id$ 是节点的编号,t_{low} 是对应时间段的下界,t_{up} 是对应时间段的上界,$children$ 记录其孩子节点,sum 记录 l_i 中属于该时间段的停留点数量;树的第 3 层是前置地点及后续地点的标记,每个节点是一个形如$\langle id,mark,children,sum\rangle$的元组,其中,$id$ 是节点的编号,$mark$ 有两个取值:pre 和 $post$,$children$ 记录其孩子节点;树的第 4 层记录的是一组地点,每个节点的结构和第 1 层一样,其中,父节点为 pre 的节点表示这个地点是前置地点,即语义轨迹一定先经过了该地点后才经过了第 2 层中记录的对应的地点,其 sum 值记录的是前置地点为该地点的停留点的数量.父节点为 $post$ 的节点表示这个地点是后续地点,即语义轨迹经过了第 2 层中记录的对应的地点后一定经过了该地点,其 sum 值记录的是后续地点为该地点的停留点的数量;需特别说明的是,前置地点或者后置地点可以是不相邻的点,比如,对于语义轨迹"⟨地点 A,12:00⟩→⟨地点 B,13:00⟩→⟨地点 C,17:00⟩"中,既包含有 AAP:{⟨地点 A,12:00⟩→⟨地点 B,13:00⟩,0.05},又包含有 AAP:{⟨地点 A,12:00⟩→⟨地点 C,17:00⟩,0.03},则地点 B 和地点 C 都是地点 A 的后置节点;树的第 5 层记录的是停留点,每个节点是形如$\langle id,S_i\rangle$的元组,id 是节点的编号,S_i 是该节点对应的停留点的 id 集合.

图 5 主要以地点 A 为例展示了 AAP-tree 的大致结构,所有括号内的数字表示节点的 sum 值.第 1 层表示有地点 A、B、C;第 2 层中,A 地点被分为 3 个时间段;第 3 层中,节点 pre 表示其孩子节点是前置地点,节点 $post$ 表示其孩子节点是后续节点;第 4 层中,父节点为 pre 的节点 A、B、C 表示"用户在 0:00–10:00 之间到达 A 地

之前去了 A、B 或 C 地”,父节点为 *post* 的节点 A、B、C 表示“用户在 0:00–10:00 之间到达 A 地之后去了 A、B 或 C 地”;第 5 层中,集合 $\{S_i|i\in\mathbb{N}^+,i\leqslant 6\}$ 中的停留点都是在时间段 0:00–10:00 之间在 A 地停留的停留点,其中,S_2 表示“用户在 0:00–10:00 之间到达 A 地之前去了 B 地”,S_5 表示“用户在 0:00–10:00 之间到达 A 地之后去了 B 地”,其余节点以此类推.建立 AAP-tree 的详细算法见算法 2.

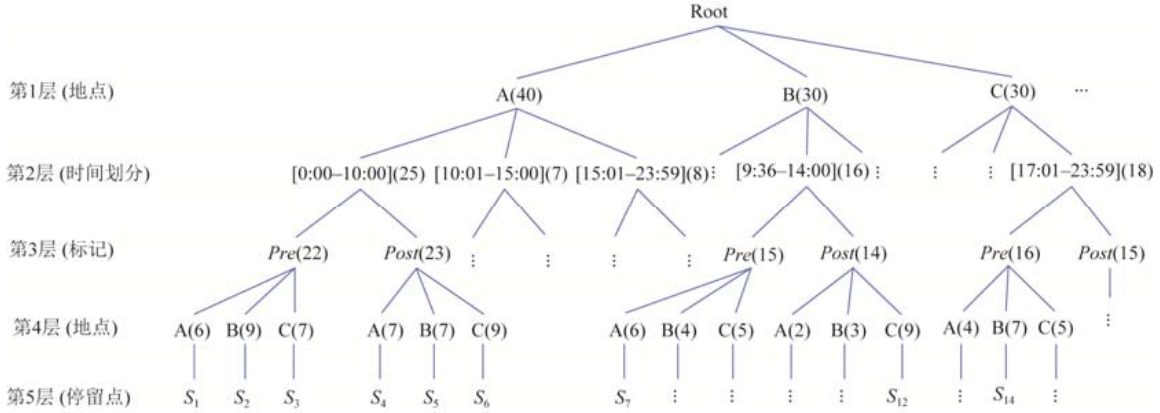


Fig.5 The structure of AAP-tree

图 5 AAP-tree 的结构

算法 2. AAP-tree_Building.

输入:语义轨迹集 *ST*,时间轴划分集 $\{D_1(S_l),D_2(S_l),\ldots,D_{|L|}(S_l)\}$;

输出:AAP-tree.

```
1.  2APTnew AAP-tree()
2.  root=∅
3.  for each stop in ST do
4.     for each layer of 2APT do          //查找索引每一层中与 stop 信息对应的节点
5.        if stop is new to this layer    //如果在当前层没有与 stop 对应信息符合的节点,则添加新节点
6.           create_node()
7.           update(parent.children)      //将新的 children 信息更新到新建节点的父节点中
8.        else                            //如果当前层有与 stop 对应信息符合的节点,则更新节点的 sum 值
9.           update(sum=sum+1)
10.       end if
11.    end for
12.  end for
13.  return 2APT
```

3.4 AAFP的高效挖掘

开始挖掘前,我们首先需要计算语义轨迹集中所有出现过的 AAP 的数量 N_{AAP}.一条语义轨迹,若其有 n 个停留点,则这条语义轨迹包含的 AAP 为 2^{n-1} 个,由此我们可以得到计算 N_{AAP} 的算法,详见算法 3.

算法 3. Count_All_AAP.

输入:语义轨迹集 *ST*;

输出:N_{AAP}.

```
1.  number_all_AAP=0
2.  for each st in ST do          //将数据集中每一条语义轨迹的 AAP 数量单独计算
```

3. $N_{AAP}(st)=2^{(|st|)}-1$ //根据语义轨迹中停留点数量计算其包含的 AAP 数量

7. $N_{AAP}=N_{AAP}+N_{AAP}(st)$ //将每条语义轨迹的 AAP 数量相加

8. end for

9. return N_{AAP}

另外,我们还需先给出一阶包含的相关定义.

定义 6(近似到达时间约束下的 *n* 元模式(approximate arrival-time constrained *n*-pattern,简称 *n*-AAP)). 对于一个 AAP:$\{\langle l_1.R,[t_1^{low}, t_1^{up}]\rangle\rightarrow\langle l_2.R,[t_2^{low}, t_2^{up}]\rangle\rightarrow\ldots\rightarrow\langle l_n.R,[t_n^{low}, t_n^{up}]\rangle,p\}$,其所包含的地点数量为 n 个,则称这个 AAP 为 n-AAP,若其符合 $p\geqslant\sigma$,则记为 n-AAFP.

定义 7(近似到达时间约束下的语义轨迹模式之间的一阶包含关系(the one-step inclusion relation between approximate arrival-time constrained patterns)). 设 fp_1 是一个 n-AAP,fp_2 是一个$(n-1)$-AAP,若有 $fp_1.\langle l_1.R,[t_1^{low}, t_1^{up}]\rangle=fp_2.\langle l_1.R,[t_1^{low},t_1^{up}]\rangle$,$fp_1.\langle l_2.R,[t_2^{low},t_2^{up}]\rangle=fp_2.\langle l_2.R,[t_2^{low},t_2^{up}]\rangle$,…,$fp_1.\langle l_{n-1}.R,[t_{n-1}^{low},t_{n-1}^{up}]\rangle=fp_2.\langle l_{n-1}.R,[t_{n-1}^{low},t_{n-1}^{up}]\rangle$,则称 fp_1 一阶包含 fp_2.

开始挖掘时,我们首先挖掘所有 1-AAFP,从树的根开始,遍历第 1 层及第 2 层,找到所有满足 $sum\geqslant\sigma N_{AAP}$ 的路径 $l_i\rightarrow[t_1^{low},t_1^{up}]$,可得到所有 1-AAFP,结果计入集合 *AAFP*(1)中.例如,在图 4 中,路径 $A\rightarrow$[0:00-10:00]即为 1-AAFP:$\{\langle A,[0:00\text{-}10:00]\rangle,0.25\}$.

根据频繁模式的先验性质[10]可知,一个频繁模式包含的任意子序列均为频繁的,又因为未在 *AAFP*(1)集合内的 1-AAP 都不是 AAFP,即不满足 $p\geqslant\sigma$,于是可得一阶包含这些 1-AAP 的 2-AAP 一定不是 AAFP.因此,我们只需基于集合 *AAFP*(1)继续挖掘即可得到所有符合条件的 2-AAFP.从 *AAFP*(1)中记录的节点继续沿着树往下走,找到所有满足 $sum\geqslant\sigma N_{AAP}$ 的路径 $l_i\rightarrow[t_k^{low},t_k^{up}]\rightarrow post\rightarrow l_j\rightarrow S_{ijk}$,然后找到所有满足 $sum\geqslant\sigma N_{AAP}$ 的路径 $l_j\rightarrow[t_h^{low},t_h^{up}]\rightarrow pre\rightarrow l_i\rightarrow S_{jih}$,$i,j,h,k\in N,i,j,h,k>0,i,j<|L|$,对每一个 i,j,h,k 的取值取 $S_{ijk}(l_j)\cap S_{jih}$,其中,$S_{ijk}(l_j)=\{s_s.post|s_s\in S_{ijk},s_s.post\in l_j\}$表示 S_{ijk} 中所有满足对应地点为 l_j 的后置停留点的集合,若 $S_{ijk}(l_j)\cap S_{jih}$ 中停留点的个数$|S_{ijk}(l_j)\cap S_{jih}|\geqslant\sigma N_{AAP}$,则这些停留点所对应的$\{\langle l_i,[t_k^{low},t_k^{up}]\rangle\rightarrow\langle l_j,[t_h^{low},t_h^{up}]\rangle,p\}$即为 2-AAFP,其中,$p=\dfrac{|S_{ijk}(post)\cap S_{jih}|}{N_{AAP}}$,我们将其记入集合 *AAFP*(2)中.需特别说明的是,由于集合 S_{ijk} 中所有停留点都落在地点 l_i 中,而集合 S_{jih} 中所有停留点都落在地点 l_j 中,两者之间没有交集,因此,为了通过取交集计算出 *sum* 值,就需要取 $S_{ijk}(l_j)$与 S_{jih} 的交集.例如,在图 4 中,路径 $A\rightarrow$[0:00–10:00]$\rightarrow post\rightarrow B\rightarrow S_5$ 表示“在[0:00–10:00]到达 A 地后去了 B 地的模式”,路径 $B\rightarrow$[9:36–14:00]$\rightarrow pre\rightarrow A\rightarrow S_7$ 表示“在[9:36–14:00]到达 B 地前去了 A 地的模式”,$|S_5(B)\cap S_7|$表示“在[0:00–10:00]到达 A 地后于[9:36–14:00]到达 B 地的模式的数量”,若有$|S_5(B)\cap S_7|\geqslant\sigma N_{AAP}$,则可以得出$\{\langle A,[0:00\text{–}10:00]\rangle\rightarrow\langle B,[9:36\text{–}14:00]\rangle,p\}$是一个 2-AAFP.此例中,$S_5$中的停留点对应的地点均为 A,$S_5$(B)表示这些停留点中地点为 B 的后置停留点的集合,则集合 S_5(B)中的停留点对应的地点均为 B,S_7 中的停留点对应的地点也均为 B,这样才可以取两个集合之间的交集计算 *sum* 值以判断该模式是否频繁.

同样地,我们只需基于 *AAFP*(2)继续挖掘 3-AAFP.对于 3-AAFP,我们只需类似挖掘 2-AAFP 的步骤,将到达第 3 个地点的每一种可能的路径所对应的标记为 *pre* 的停留点集合所对应的每一种可能的后置点集合与每一个标记为 *post* 的停留点集合做交集,通过取交集后得到的集合中停留点的数量判断该集合对应的模式是否频繁,即可得到所有 3-AAFP.例如,在图 4 中,基于 S_5(B)$\cap S_7$ 我们已经得到 2-AAFP:$\{\langle A,[0:00\text{–}10:00]\rangle\rightarrow\langle B,[9:36\text{–}14:00]\rangle,p\}$,对于路径 B$\rightarrow$[9:36–14:00]$\rightarrow post\rightarrow$C$\rightarrow S_{12}$ 以及路径 C$\rightarrow$[17:01–23:59]$\rightarrow pre\rightarrow$B$\rightarrow S_{14}$,若$|(S_5(B)\cap S_7\cap S_{12})(C)\cap S_{14}|\geqslant\sigma N_{AAP}$,则可以得出$\{\langle A,[0:00\text{–}10:00]\rangle\rightarrow\langle B,[9:36\text{–}14:00]\rangle\rightarrow\langle C,[17:01\text{–}23:59]\rangle,p'\}$是一个 3-AAFP.

4-AAFP 至 n-AAFP 的挖掘方法与 3-AAFP 类似,当找不到交集中停留点数量$\geqslant\sigma N_{AAP}$的路径时,算法就停止,至此,所有符合条件 $p\geqslant\sigma$的 AAFP 挖掘完毕,完整过程详见算法 4,其中,步骤 1~步骤 5 是挖掘出所有 1-AAFP,步骤 6~步骤 14 是挖掘出所有 2-AAFP,步骤 15~步骤 27 是挖掘 2 阶以上的 AAFP.理论上,算法迭代的次数最高

是 n–2 次,n 等于单条语义轨迹包含停留点数的最大值,随着迭代次数的增加,每次迭代中取交集的次数也在增加.因此,迭代次数越多,算法计算效率越低.然而,根据文献[21]的讨论,在实际情况中,极少存在包含两个停留点以上的语义轨迹频繁模式,加上到达时间限制后,包含两个停留点以上的 AAFP 则更少.因此,算法 4 通常完成步骤 20 以后就不再存在符合条件的路径,很少进入步骤 15~步骤 27 的循环,其实际运行效率是较高的(详见第 5.4 节).

算法 4. AAFP_Mining.

输入:AAP-tree,σ,N_{AAP};

输出:所有 AAFP.

1. for each *node* in *first_layer* of AAP-tree where *first_layer.sum*>=σN_{AAP} do
2. if *second_layer.sum*>=σN_{AAP} //根据 AAP-tree 第 1 层、第 2 层 *sum* 值找出所有 1-AAFP
3. *AAFP*(1).append(*node.path*)
4. end if
5. end for
6. for each 1-AAFP in *AAFP*(1) do //以集合 AAFP(1)为基础挖掘 2-AAFP
7. *stop_set*=find all *node* in *fifth_layer* where *third_layer.sum*>=σ|*All* AAP| and *fourth_layer.sum*>=σN_{AAP} //找出第 3 层、第 4 层中所有 *sum*>=σN_{AAP} 的路径,并找到路径对应的集合
8. for each pair (S_i,S_j)∈*stop_set* do //对每两个集合进行操作
9. *intersection*=S_i(S_j.*l*)∩S_j //取前一个集合中地点等于后一个集合地点的后置点的集合与后一个集合的交集
10. if length(*intersection*)>=σN_{AAP} //如果交集中停留点数量大于σN_{AAP}
11. *AAFP*(2).append((S_i(S_j.*l*)∩S_j).*path*) //则这两个集合路径对应的模式为 2-AAFP
12. end if
13. end for
14. end for
15. for *i* from 2 to max_length(*st*) //2 阶以上的 AAFP 的挖掘
16. if *i-AAFP* !=∅ //如果上一阶 AAFP 集合为空,则算法结束
17. for each *i-AAFP* in *AAFP*(*i*) do //找到所有上一阶 AAFP 对应的集合
18. for each *node* in *stop_set* where *third_layer.mark*=*pre* do //找到所有 AAP-tree 第 5 层中标记为 *pre* 的节点
19. *intersection*=*i-AAFP.S*(*node.S.l*)∩*node.S* //取所有可能的前一个集合的后置点集合与后一个集合的交集
20. if *length*(*intersection*)>=σN_{AAP} //找出所有交集中停留点数量大于σN_{AAP} 的组合
21. *AAFP*(*i*+1).append((*i-AAFP.S*∩*node.S*).*path*) //则这些组合对应的模式为 *i-AAPF*
22. end if
23. end for
24. end for
25. end if
26. end for
27. return {*AAFP*(1),*AAFP*(2),…,*AAFP*(*n*)} //返回所有 AAFP

4 时间轴划分与 AAP-tree 的维护

根据第 3.1 节所述,为了将时间轴划分得合理,本文使用了基于信息熵聚类的方法将时间轴划分开,这样的

划分却会带来一个问题:由于用户的语义轨迹是随着时间不断产生的,新数据的到来就有可能改变数据分布的特征,一旦数据分布特征改变,相应地,第 3.1 节中所计算的信息熵与划分都会随之改变.同时,第 3.3 节中构造的 AAP-tree 的结构也会随之改变.因此,需要对时间轴的划分以及 AAP-tree 进行维护.

4.1 信息熵计算增量方法

由算法 1,计算信息熵 $Ent(S_l)$的时间复杂度为 $O(N)$.对于地点 l,当一条新的数据 s_{new} 到来时,设 $s_{new}\in C_j$,$1\leqslant j\leqslant k$,$p(C_j)$的值就需要重新计算.又因为对$\forall p(C_i)\in\{p(C_1),p(C_2),\ldots,p(C_k)\}$有 $p(C_i)=\dfrac{|C_i|}{|S_l|}$,$1\leqslant i\leqslant k$,$s_{new}$ 的到来导致 $|S_l|_{new}=|S_l|+1$,所以即使只到来了一条新数据 s_{new},仍然需要把每一个类的概率重新计算一遍,算法时间复杂度也是 $O(N)$.

新的数据 s_{new} 到来后,信息熵的变化如下:

$$Ent(S_l)_{new}=-\left(\sum_{i=1}^{j-1}\left(\frac{|C_i|}{|S_l|+1}\log\left(\frac{|C_i|}{|S_l|+1}\right)\right)+\frac{|C_j|+1}{|S_l|+1}\log\left(\frac{|C_j|+1}{|S_l|+1}\right)+\sum_{i=j+1}^{k}\left(\frac{|C_i|}{|S_l|+1}\log\left(\frac{|C_i|}{|S_l|+1}\right)\right)\right).$$

我们用新数据到来后的信息熵减去新数据到来前的信息熵,利用比值的对数函数可以表示为其分子分母对数函数的差的性质来对公式进行拆分计算,整理合并后得到:

$$Ent(S_l)_{new}-Ent(S_l)=\frac{|S_l|\cdot\log\dfrac{|S_l|+1}{|S_l|}-|C_j|\cdot\log\dfrac{|C_j|+1}{|C_j|}-\log\dfrac{|C_j|+1}{|S_l|+1}+Ent(S_l)}{|S_l|+1},$$

变形后有:

$$Ent(S_l)_{new}=\frac{|S_l|\cdot\log\dfrac{|S_l|+1}{|S_l|}-|C_j|\cdot\log\dfrac{|C_j|+1}{|C_j|}-\log\dfrac{|C_j|+1}{|S_l|+1}+(|S_l|+2)\cdot Ent(S_l)}{|S_l|+1}\tag{3}$$

我们称公式(3)为 $s_{new}\in C_j$ 的信息熵增量公式.该公式的推导过程详见附录.

一条新数据 s_{new} 到来后,使用公式(3)进行计算,仅有插入了 s_{new} 的类 C_j 的$|C_j|$需重新计算,$|S_l|$与 $Ent(S_l)$均为已知,所以计算信息熵 $Ent(S_l)$的算法时间复杂度降低为 $O(1)$.

4.2 时间轴划分的更新

在一些特殊情况下,语义轨迹集每次新增的数据只有一个停留点,比如,语义轨迹以流数据形式新增,或者新增数据的数据量极小,这些情况下对时间轴划分进行更新只需使用公式(3)即可快速计算出新的信息熵,然后根据第 3.1 节中的方法计算得到新的时间轴划分.

一般情况下,对语义轨迹集的更新是每隔一段时间将新收集的数据一次性更新到旧数据集中,对于每个地点,一次更新通常都是新增多于 1 个以上的停留点.为了实现对停留点批量到来时信息熵的重新计算,我们将修改公式(3)以得到更具一般性的增量公式.

对于地点 l,假设新到来 m 个点,这些点非平均地分布在 s 个类中:$\bar{C}_1,\bar{C}_2,\ldots,\bar{C}_s$,每个类对应的新增点数量为 $a_1,a_2,\ldots,a_s$ 个,$a_1+a_2+\ldots+a_s=m$,未被更新的类为 $C_1,C_2,\ldots,C_r$,$r+s=k$,k 为类的总数量,则信息熵的变化如下:

$$Ent(S_l)_{new}=-\left(\sum_{i=1}^{r}\frac{|C_i|}{|S_l|+m}\log\left(\frac{|C_i|}{|S_l|+m}\right)+\sum_{j=1}^{s}\frac{|\bar{C}_j|+a_j}{|S_l|+m}\log\left(\frac{|\bar{C}_j|+a_j}{|S_l|+m}\right)\right).$$

同样,我们用新信息熵减去旧信息熵,将公式拆分、消项、合并、整理后就可得到:

$$Ent(S_l)_{new}=\frac{M}{|S_l|+m}+Ent(S_l)\tag{4}$$

其中,

$$M = |S_l| \cdot \log\frac{|S_l|+m}{|S_l|} + mEnt(S_l) - \sum_{j=1}^{s}\left(|\bar{C}_j| \cdot \log\frac{|\bar{C}_j|+a_j}{|\bar{C}_j|} + a_j \log\frac{|\bar{C}_j|+a_j}{|S_l|+m}\right).$$

我们称公式(4)为信息熵批量增量公式.该公式的推导过程详见附录.用公式(4)计算新信息熵的算法复杂度为 $O(M)$,M 的大小主要取决于 s 的个数,与算法 1 中计算 $Ent(S)$的算法对比,若 $s=k$,则 $O(M)=O(N)$.然而,实际更新中,很少有一个地点的每一个类都会加入新增点的情况,大多数情况下,有 $s \ll k$,则 $O(M) \ll O(N)$.故通常情况下,使用公式(4)进行批量更新效率高于使用算法 1 直接进行计算(详见第 5.4 节).下面给出使用公式(4)的时间划分更新算法.

算法 5. MDLP-L_Update.

输入:语义轨迹集 ST,更新数据集 ST_{new},时间轴划分集$\{D_1(S_l),D_2(S_1),\ldots,D_{|L|}(S_1)\}$;

输出:时间轴新划分集$\{D_1^{new}(S_l), D_2^{new}(S_l),\ldots, D_{|L|}^{new}(S_l)\}$.

1. same as algorithm 1 line 1~line 12
2. calculate $Ent(S_l)_{new}$ according to formula (4)
3. same as algorithm 1 line 14~line 17
4. calculate $E(C_i,C_{i+1})_{new}$ according to formula (4)
5. same as algorithm 1 line 19~line 26
6. return $\{D_1^{new}(S_l), D_2^{new}(S_l),\ldots, D_{|L|}^{new}(S_l)\}$

4.3 AAP-Tree 的更新

当计算得到新的时间轴划分之后,AAP-tree 除第 1 层不变外,其他每一层的节点都需要更新.首先更新第 2 层,使每一个节点对应新的时间轴划分,然后更新第 5 层的停留点集合,使每个停留点被包含在正确的集合中,最后更新第 2 层~第 4 层每一个节点 *sum* 值,使其为正确的统计值.详细算法如下.

算法 6. AAP-tree_Update.

输入:AAP-tree,更新数据集 ST_{new},时间轴划分集$\{D_1(S_l),D_2(S_l),\ldots,D_{|L|}(S_l)\}$,时间轴新划分集$\{\{D_1^{new}(S_l), D_2^{new}(S_l),\ldots, D_{|L|}^{new}(S_l)\}$;

输出:新 AAP-tree.

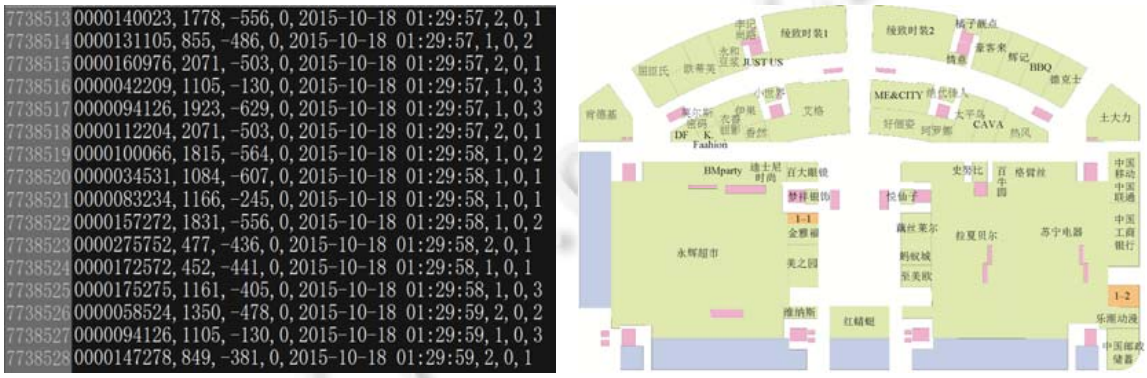
```
1.  for each node in first_layer do
2.    if D_i(S_l)!= D_i^new(S_l)                  //仅更新划分发生了变化的地点
3.      update second_layer with D_i^new(S_l)     //根据新的划分更新索引第 2 层
4.    update third_layer with second_layer        //根据第 2 层的变动更新第 3 层
5.    update fifth_layer with D_i^new(S_l)        //根据新的划分更新索引第 5 层
6.      if node.fifth_layer is new                //若出现第 4 层中没有的地点,则第 4 层加新节点
7.        fourth.layer.create_node()
8.      else if node.fifth_layer=∅                //若变动后第 4 层出现空节点,则删除节点
9.        fourth.layer.delete_node()
10.   end if
11.  update sum in all layer                      //更新每一层的 sum 值
12.  end if
13. end for
14. return new AAP-tree
```

5 实验及分析

5.1 真实数据集及实验环境介绍

实验部分我们使用的是来自河南某商场的真实室内轨迹数据集,该数据集是通过商场无线 AP(access point)设备采集的,一旦用户的手机接入 AP 设备,我们就能通过信号强弱计算用户距离 AP 设备的距离,最后获得形如(UserID,*x*,*y*,areaId,positionTime,stationType,BuildId,FloorId)的轨迹数据,每个元组分别表示用户 ID、*x* 坐标、*y* 坐标、区域 ID、采样时刻、连接状态、建筑 ID、楼层 ID.数据形式如图 6(a)所示.

该数据集的采样频率是 1 次/秒,属于高频采样数据,数据是在 2015 年 9 月从约 30 万名商场用户的移动设备采集获取的.该数据集一共有 43 784 744 个采样点,731 739 条轨迹,数据体积一共 2.11GB.该商场有 3 层楼,一共 164 个商户,即该数据集共 164 个地点.商场地图如图 6(b)所示.



(a) 数据形式 (b) 商场地图

Fig.6 The data set and map

图 6 数据集与地图

本实验是在配置为酷睿 i7-4720(2.60GHz 双核)、16GB 内存的计算机上进行的,系统为 Win 10,使用 Python(3.6.0)+PostgreSQL(9.6)编写程序.

5.2 数据预处理

在数据预处理阶段,我们使用文献[12]中的方法将裸轨迹数据转化为语义轨迹,取Δt=1min,即轨迹在一个地点中停留超过 1 分钟即认定为一个停留点.在转化语义轨迹的同时,我们还将数据采样频率过滤为 1 次/5 秒、去除重复采样、去除噪点,并将数据形式修改为形如(stopID,traID,areaID,t_{in},*pre*,*post*)的语义轨迹,每个元组分别表示停留点 ID、轨迹 ID、地点 ID、进入 POI 的时间、停留点的上一个停留点、停留点的下一个停留点,这样的数据形式可以更好地支持语义轨迹的模式挖掘.处理后的数据详情见表 1,其中,大多数轨迹包含的停留点数量为 4~6 个.可见处理后数据的体积明显下降,数据价值密度明显得到了提高.

Table 1 Over view of data after pre-processing

表 1 预处理后数据概览

数据体积(MB)	停留点数量(个)	轨迹数量(条)	最短轨迹(停留点个数)	最长轨迹(停留点个数)	平均长度(停留点个数)
118	1 609 775	731 739	1	21	2.2

为了更加充分地对比及分析本文方法的效果与效率,我们将处理后的语义轨迹数据集处理为 4 个不同体积、包含不同数量轨迹的数据集,用于评估挖掘方法的效果及效率,见表 2,集合 st_10、st_30、st_50、st_70 中分别包含约 10 万、30 万、50 万、70 万条轨迹,分别占原数据集比例的 15%、40%、70%、100%,其中所包含的 AAP 总数分别约为 340 万个、1 000 万个、1 640 万个、2 300 万个.这些数据集都是在原语义轨迹数据集中

按给定比例随机选择语义轨迹组成的.

Table 2 Over view of experimental data set

表 2 实验使用数据集概览

	st_10	st_30	st_50	st_70
轨迹数量(条)	109 761	292 696	512 217	731 739
轨迹数占原数据集比例(%)	15	40	70	100
AAP 总数(个)	3 360 126	9 950 105	16 391 640	22 955 519

另外,为了评估时间轴划分更新及 AAP-tree 更新效率,我们还按照时间先后选择了前 50 万条轨迹作为历史轨迹集,然后将剩下的轨迹按照时间先后选出 7 万条、14 万条、21 万条组成 3 组数据作为需要更新的数据集,见表 3.3 组数据中包含的停留点数量分别约为 15 万个、30 万个、48 万个.

Table 3 Over view of updating data set

表 3 更新使用数据集概览

	st_new_7	st_new_14	st_new_21
轨迹数量(条)	70 000	140 000	210 000
停留点数(个)	153 835	300 129	482 213

5.3 模拟数据集

为了验证本文挖掘算法的正确性,测试其在较大数据集上的效率以及更充分地对其进行实验分析,我们在真实数据集的基础上生成了一个包含更多语义轨迹的模拟数据集.该模拟数据集依然采用真实数据集的商场地图,首先根据真实数据集计算并统计出用户在不同时刻到达每一个商户的概率值,然后根据这些概率值随机地生成用户在商场中出现的位置与时刻(此处的位置是指各个商户的 areaID)以模拟用户在商场中的行为,最后将这些点连接起来形成一条虚拟的轨迹,若干次模拟后得到最终的模拟数据集.该数据集详细信息见表 4.

Table 4 Over view of synthetic data set

表 4 模拟数据集概览

数据集名称	轨迹数量(条)	AAP 总数(个)
st_90	900 000	35 024 611

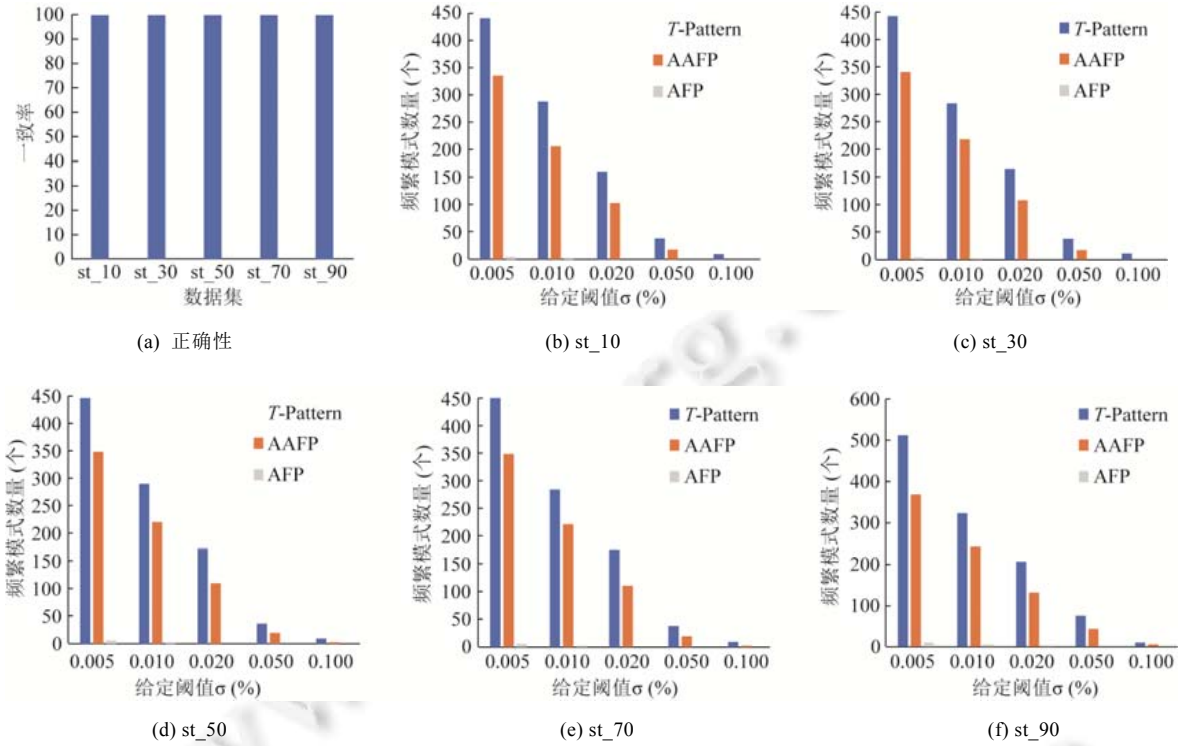
5.4 效果分析

首先,我们分析本文挖掘方法的正确性,由于基线算法是将所有轨迹顺序检索一遍来建立 AASTP-tree,故基线算法能够挖掘出数据集中存在的所有 AAFP.我们将 AAFP-Mining 算法在各个数据集上的挖掘结果逐个地与基线算法的挖掘结果进行对比,考察其一致率,结果如图 7(a)所示.可见,AAFP-Mining 算法的挖掘结果与基线算法是一致的,即 AAFP-Mining 算法的挖掘结果是正确的.

然后,我们分析本文定义的 AAFP 的挖掘效果.根据本文第 1 节相关工作的分析,文献[2,3,6,8]中的方法都可挖掘语义轨迹频繁模式,文献[2,3]在挖掘过程中并未考虑任何时间信息,文献[6]考虑的是行程时间,文献[8]考虑了精确的到达时间,由于文献[2,3]未考虑时间,所以在方法效果的对比工作中,本文选择与文献[6,8]的方法进行对比.

图 7 展示了 3 种方法在数据集 st_10、st_30、st_50 和 st_70 上分别取频繁阈值为 0.005%、0.01%、0.02%、0.05%及 0.1%时挖掘得到的频繁模式数量,其中,*T*-Pattern 为文献[6]中方法挖掘出的考虑行程时间的频繁模式;AFP 为文献[8]中方法挖掘出的考虑精确到达时间的频繁模式;AAFP 为本文方法挖掘出的考虑近似到达时间的频繁模式.可以看出,随着数据集中轨迹数量的增加,3 种方法在真实数据集上挖掘到的频繁模式数量都略有增加,但是增幅很小,说明真实数据集中模式的特征较稳定,挖掘到的模式数量仅在模拟数据集中增幅较大,这是因为模拟数据集中存在的频繁模式与真实数据集相似但并不相同;随着频繁阈值的增加,3 种方法能够挖掘到的频繁模式数量都快速减少,阈值高于 0.05%时,3 种方法都只能挖掘到极少量的频繁模式;*T*-Pattern 方法约束最小,挖掘到的频繁模式数量最多;AFP 方法约束最强,几乎挖掘不到频繁的模式;AAFP 方法挖掘出的频繁模

式数量远多于 AFP 方法,说明在考虑近似到达时间后,本文方法能够挖掘出 AFP 方法不能挖掘到的频繁模式,挖掘效果得到显著提升;AAFP 方法挖掘出的频繁模式少于 *T*-Pattern 方法,但 *T*-Pattern 挖掘出的频繁模式不包含到达时间约束,对于对到达时间有需求的应用无法适用.



(a) 正确性 (b) st_10 (c) st_30

(d) st_50 (e) st_70 (f) st_90

Fig.7 The correctness, and the effectivness of mining on st_10, st_30, st_50, st_70, st_90

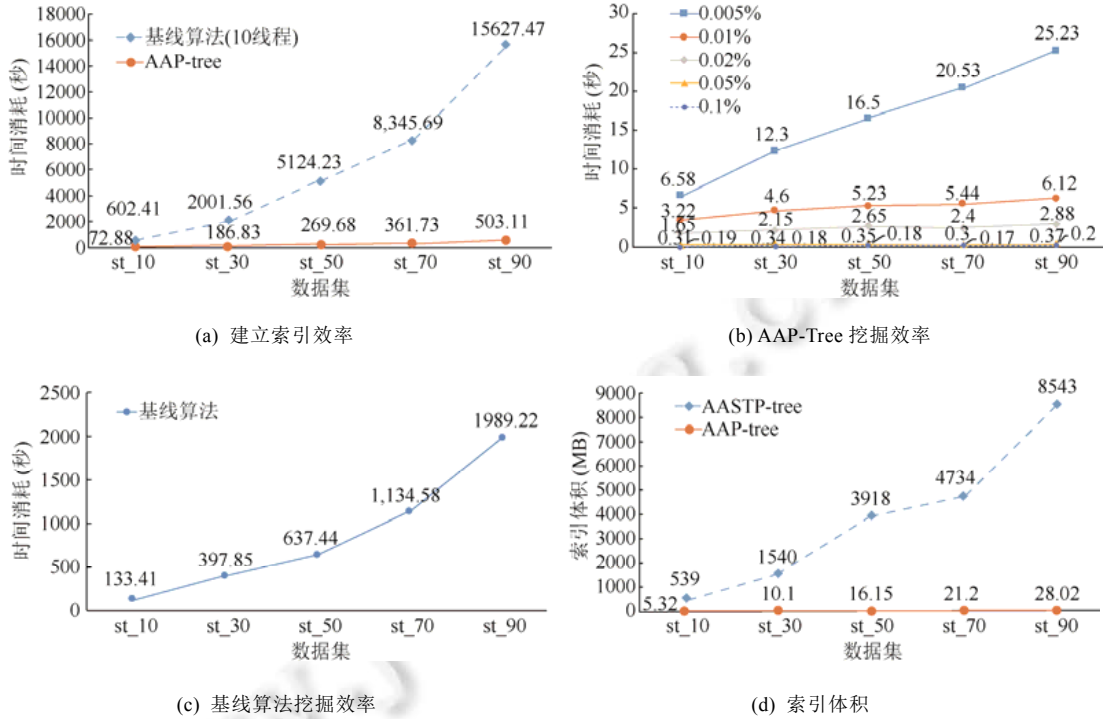
图 7 算法正确性以及其在 st_10、st_30、st_50、st_70、st_90 上的挖掘效果

5.5 效率分析

效率的分析主要分为两个部分,首先是频繁模式挖掘的效率分析,然后是时间轴划分更新和索引更新的效率分析.在频繁模式挖掘效率分析部分,由于现有的频繁模式挖掘方法都无法直接挖掘 AAFP,而前文介绍的基线算法是根据现有方法修改得到,所以我们将主要对基线算法及本文方法的挖掘效率进行对比.

图 8 展示了基线算法的效率与使用 AAP-tree 算法的效率.图 8(a)展示了两种算法在建立索引阶段的效率,其中,基线算法包括预先挖掘出所有 AAP 的时间以及基于所有 AAP 建立索引的时间,由于所消耗的时间过长,我们在 Python 中设置了 5 个线程以降低建立索引所需时间;AAFP_Mining 算法包括计算 AAP 数量的时间与建立 AAP-tree 的时间,即实现算法 Count_All_AAP 与算法 AAP-tree_Building 的时间,图中建立 AAP-tree 所花费的时间为单线程结果;从图中可见,建立 AAP-tree 的算法随着数据量的增长所消耗的时间呈亚线性增长,且效率远高于基线算法,基线算法所消耗的时间呈超线性增长,这主要是因为,AAP-tree_Building 只需遍历所有停留点,基线算法需要遍历所有 AAP,而 AAP 的数量随着停留点数量的增加呈倍数增长.图 8(b)展示了 AAFP_Mining 算法在不同频繁阈值下的挖掘效率,可见,该算法消耗的时间随着数据量的增加其效率呈亚线性增加,随着频繁阈值的降低所消耗的时间呈倍数增加,但是总体挖掘速度尚能接受,当频繁度低至 0.005%时,最慢挖掘时间仅需约 20s.图 8(c)展示了基线算法在各个频繁阈值下的平均挖掘效率(单线程),由于基线算法无论频繁阈值是多少,都需要遍历整个索引,所以阈值的改变对挖掘效率基本没有影响.由图 8(b)和图 8(c)可以看出, AAFP_Mining 算法可以根据不同的阈值设定快速挖掘出符合条件的频繁模式,故 AAFP_Mining 算法相比基线算法更具灵活性.

图 8(d)展示了基线算法中 AASTP-tree 的体积与 AAP-tree 的体积对比情况,虽然 AAP-tree 是冗余型索引,但是由于 AASTP-tree 是穷举型索引,其体积还是远大于 AAP-tree.



(a) 建立索引效率 (b) AAP-Tree 挖掘效率

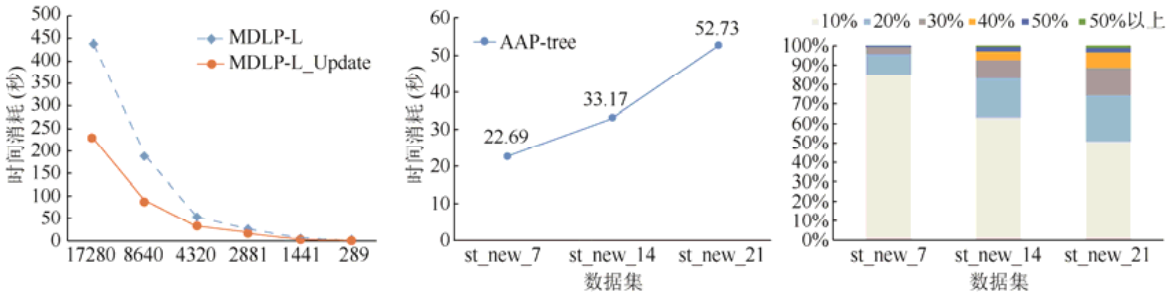
(c) 基线算法挖掘效率 (d) 索引体积

Fig.8 The efficiency and column of baseline algorithm and AAP-tree algorithm

图 8 基线算法效率与 AAP-tree 算法效率及体积

从图 8(b)还可看出,AAFP_Mining 算法的收敛速度较快,随着频繁阈值的增加,算法收敛速度快速提升,这是由于阈值增加后符合条件的 *sum* 值大量减少、每一阶 AAFP 的备选集中 AAFP 的数量急剧下降所致.实验过程中,AAFP_Mining 算法极少出现进入求二阶以上 AAFP 循环的情况,仅在频繁阈值取 0.005%及 0.01%时出现了极少次数,其在各个数据集中出现的平均概率为 0.00006%及 0.00003%.可见,AAFP_Mining 算法在给定频繁阈值时能够高效地挖掘频繁模式,且挖掘效率随着数据量的增加呈亚线性增加.

接下来分析时间轴划分更新和 AAP-tree 更新的效率.图 9(a)展示了新数据到来后,对时间轴划分使用 MDLP-L 算法进行非增量更新以及使用 MDLP-L_Update 算法进行增量更新的效率,算法效率不受更新数据量多少的影响,而是与时间采样粒度有关,即算法效率取决于 k 的取值.实验中,我们将采样频率分别设定为 1 次/5 秒、1 次/10 秒、1 次/20 秒、1 次/30 秒、1 次/60 秒及 1 次/300 秒,对应的 k 值分别为 17 280、8 640、4 320、2 881、1 441、289.可见,两种算法消耗的时间都随着 k 值的下降而呈倍数减少,但 MDLP-L_Update 算法的更新效率还是高于 MDLP-L 算法,这是由于 MDLP-L_Update 算法中信息熵的计算效率是与需要更新的类数量有关,而不是全部类的数量.图 9(c)展示了对 164 个地点进行时间轴划分更新时,3 个更新数据集中的停留点所覆盖的类占类总数的比值.在 st_new_7 中,超过 80%的地点在更新时,新到的停留点仅覆盖了不超过 10%的类;在 st_new_14 中,超过 80%的地点在更新时,新到的停留点仅覆盖了不超过 20%的类;在 st_new_21 中,超过 90%的地点在更新时,新到的停留点仅覆盖了不超过 30%的类;3 个数据集都极少出现一个地点更新的停留点覆盖了 50%类的情况.也就是说,在实际中,需要更新的类数量远小于全部类的数量,因此,图 9(a)中 MDLP-L_Update 算法更新效率高于 MDLP-L 算法.图 9(b)展示了 AAP-tree 的更新效率,可以看到,随着数据量的增长,更新消耗的时间呈线性增加.



(a) 时间轴划分更新效率 (b) AAP-Tree 更新效率 (c) 更新停留点集的时刻覆盖数量

Fig.9 The efficiency of updating time axis dividing and AAP-tree and the distribution of new stops on time axis

图 9 时间轴、AAP-tree 更新效率与更新停留点集时刻覆盖

由图 8 和图 9 可以看出,本文方法无论在数据预处理、索引建立、频繁模式挖掘还是时间轴划分与索引的更新方面都是高效的,同时,本文方法相比现有方法还具有一定的灵活性.

6 总结及展望

挖掘语义轨迹的频繁模式是实现旅游线路推荐、路线预测、用户生活模式挖掘、朋友推荐等应用的技术基础,在很多情况下,用户会对这些应用有到达时间限制的需求,而现有的语义轨迹方法大多没有考虑到达时间限制,少数考虑了到达时间限制却因为限制太强而导致挖掘不到频繁的模式.因此,本文首次研究并形式化定义了语义轨迹的 AAFP,并使用信息熵聚类的方法将语义轨迹集中各个地点的时间轴合理地划分开,并提出了挖掘语义轨迹的 AAFP 的基线算法.之后,为了改进基线算法使其更高效、更灵活,本文建立了一个多层混合索引 AAP-tree,并给出了基于其上的语义轨迹 AAFP 挖掘算法.然后,针对新数据到来后的维护问题,提出了时间轴划分及 AAP-tree 的高效维护方案.最后,在真实数据集上进行实验,实验结果证明了本文方法的有效性与高效性.

在未来的工作中,我们还将根据实际中存在的新颖性需求对数据集进行维护,比如,商场的管理者会希望仅维护近 3 个月的数据集以确保数据能够反映最近的流行趋势,在这样的场景下,本文方法还需在维护时间轴划分与索引时考虑删除点的情况.另外,本文方法主要是针对挖掘频繁的模式,设定频繁阈值可以过滤掉大量的不符合需求的模式以加快挖掘速度,若需要挖掘所有的模式,则本文方法并不适用,故未来还需考虑高效的模式挖掘方法.

References:

[1] Zheng Y. Trajectory data mining: An overview. ACM Trans. on Intelligent Systems and Technology (TIST), 2015,6(3):29. [doi: 10.1145/2743025]

[2] Ying JJC, Lee WC, Weng TC, Tseng VS. Semantic trajectory mining for location prediction. In: Agrawal D, Cruz I, Jensen CS, Ofek E, Tanin E, eds. Proc. of the 19th ACM SIGSPATIAL Int'l Conf. on Advances in Geographic Information Systems. New York: ACM Press, 2011. 34–43. [doi: 10.1145/2093973.2093980]

[3] Ying JC, Chen HS, Lin KW, Lu HC, Tseng VS, Tsai HW, Cheng KH, Lin SC. Semantic trajectory-based high utility item recommendation system. Expert Systems with Applications, 2014,41(10):4762–4776. [doi: 10.1016/j.eswa.2014.01.042]

[4] Li Q, Zheng Y, Xie X, Chen YK, Liu WY, Ma WY. Mining user similarity based on location history. In: Aref WG, Mokbel MF, Schneider M, eds. Proc. of the 16th ACM SIGSPATIAL Int'l Conf. on Advances in Geographic Information Systems. New York: ACM Press, 2008. 34. [doi: 10.1145/1463434.1463477]

[5] Zheng Y, Zhang L, Ma Z, Xie X, Ma WY. Recommending friends and locations based on individual location history. ACM Trans. on the Web (TWEB), 2011,5(1):5. [doi: 10.1145/1921591.1921596]

[6] Monreale A, Pinelli F, Trasarti R, Giannotti F. Wherenext: A location predictor on trajectory pattern mining. In: Elder J, Fogelman FS, eds. Proc. of the 15th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2009. 637–646. [doi: 10.1145/1557019.1557091]

[7] Zhang C, Han J, Shou L, Lu J, Porta TL. Splitter: Mining fine-grained sequential patterns in semantic trajectories. Proc. of the VLDB Endowment, 2014,7(9):769–780. [doi: 10.14778/2732939.2732949]

[8] Chen CC, Kuo CH, Peng WC. Mining spatial-temporal semantic trajectory patterns from raw trajectories. In: Proc. of the 2015 IEEE Int'l Conf. on Data Mining Workshop (ICDMW). New York: IEEE, 2015. 1019–1024. [doi: 10.1109/ICDMW.2015.55]

[9] Jeung H, Yiu ML, Jensen CS. Trajectory pattern mining. In: Computing with Spatial Trajectories. Berlin, Heidelberg: Springer-Verlag, 2011. 143–177. [doi: 10.1007/978-1-4614-1629-6]

[10] Aggarwal CC, Han JW. Frequent Pattern Mining. New York: Springer-Verlag, 2014. [doi: 10.1007/978-3-319-07821-2]

[11] Baglioni M, Renso C, Trasarti R, Wachowicz M. Towards semantic interpretation of movement behavior. In: Sester M, Bernard L, Paelke V, eds. Advances in GIScience. Lecture Notes in Geoinformation and Cartography, Berlin, Heidelberg: Springer-Verlag, 2009. 271–288. [doi: 10.1007/978-3-642-00318-9_14]

[12] Alvares LO, Bogorny V, Kuijpers B, Macedo JAFD, Moelans B, Vaisman A. A model for enriching trajectories with semantic geographical information. In: Samet H, Shahabi C, eds. Proc. of the 15th Annual ACM Int'l Symp. on Advances in Geographic Information Systems. New York: ACM Press, 2007. 1–8. [doi: 10.1145/1341012.1341041]

[13] Giannotti F, Nanni M, Pinelli F, Pedreschi D. Trajectory pattern mining. In: Berkhin P, ed. Proc. of the 13th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2007. 330–339. [doi: 10.1145/1281192.1281230]

[14] Zheng K, Zheng Y, Yuan NJ, Shang S, Zhou X. On discovery of gathering patterns from trajectories. In: Proc. of the 29th IEEE Int'l Conf. on Data Engineering (ICDE). New York: IEEE, 2013. 242–253. [doi: 10.1109/ICDE.2013.6544829]

[15] Hung CC, Peng WC, Lee WC. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. The Int'l Journal on Very Large Data Bases, 2015,24(2):169–192. [doi: 10.1007/s00778-011-0262-6]

[16] Matsubara Y, Li L, Papalexakis E, Lo D, Sakurai Y, Faloutsos C. F-Trail: Finding patterns in taxi trajectories. In: Pei J, Tseng VS, Cao L, Motoda H, Xu G, eds. Proc. of the Pacific-Asia Conf. on Knowledge Discovery and Data Mining. Berlin, Heidelberg: Springer-Verlag, 2013. 86–98. [doi: 10.1007/978-3-642-37453-1_8]

[17] Roh GP, Roh JW, Hwang SW, Yi BK. Supporting pattern-matching queries over trajectories on road networks. IEEE Trans. on Knowledge and Data Engineering, 2011,23(11):1753–1758. [doi: 10.1109/TKDE.2010.189]

[18] Roh GP, Hwang S. TPM: Supporting pattern matching queries for road-network trajectory data. In: Ailamaki A, Amer-Yahia S, Pate J, Risch T, Senellart P, Stoyanovich J, eds. Proc. of the 14th Int'l Conference on Extending Database Technology. New York: ACM Press, 2011. 554–557. [doi: 10.1145/1951365.1951439]

[19] Qiu M, Pi D. Mining frequent trajectory patterns in road network based on similar trajectory. In: Yin H, ed. Proc. of the Intelligent Data Engineering and Automated Learning (IDEAL 2016). Lecture Notes in Computer Science, 2016. 46–57. [doi: 10.1007/978-3-319-46257-8_6]

[20] Fayyad UM. Multi-Interval discretization of continuous-valued attributes for classification learning. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. 1993. 1022–1027.

[21] Kim Y, Han J, Yuan C. TOPTRAC: Topical trajectory pattern mining. In: Cao LB, Zhang CQ, eds. Proc. of the 21st ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2015. 587–596. [doi: 10.1145/2783258.2783342]

附录

A.1 信息熵增量公式推导

根据第 3.1 节,地点 l 停留点分布的信息熵计算公式为 $Ent(S_l)=-\sum_{i=1}^{k}p(C_i)\log(p(C_i))=-\sum_{i=1}^{k}\frac{|C_i|}{|S_l|}\log\left(\frac{|C_i|}{|S_l|}\right)$,其中,$1\leqslant i\leqslant k$.当一条新的数据 s_{new} 到来时,设 $s_{new}\in C_j$,$1\leqslant j\leqslant k$,信息熵的变化如下:

$$Ent(S_l)_{new}=-\left(\sum_{j-1}^{i=1}\left(\frac{|C_i|}{|S_l|+1}\log\left(\frac{|C_i|}{|S_l|+1}\right)\right)+\frac{|C_j|+1}{|S_l|+1}\log\left(\frac{|C_j|+1}{|S_l|+1}\right)+\sum_{i=j+1}^{k}\left(\frac{|C_i|}{|S_l|+1}\log\left(\frac{|C_i|}{|S_l|+1}\right)\right)\right).$$

根据比值的对数函数可以表示为其分子分母对数函数的差的性质,上述两个公式可以变形为

$$Ent(S_l)=-\sum_{i=1}^{k}\frac{|C_i|}{|S_l|}(\log(|C_i|)-\log(|S_l|)) \tag{5}$$

$$\left.\begin{aligned}Ent(S_l)_{new}=-\left(\sum_{j-1}^{i=1}\left(\frac{|C_i|}{|S_l|+1}(\log(|C_i|)-\log(|S_l|+1)\right)+\frac{|C_j|+1}{|S_l|+1}\left(\log(|C_j|+1)-\log(|S_l|+1)\right)+\right.\\ \left.\sum_{i=j+1}^{k}\left(\frac{|C_i|}{|S_l|+1}\left(\log(|C_i|)-\log(|S_l|+1)\right)\right)\right)\end{aligned}\right\} \tag{6}$$

用式(6)减去式(5)得到的分母与分子如下:

$$\left.\begin{aligned}&den\left(Ent(S_l)_{new}-Ent(S_l)\right)=-(|S_l|+1)|S_l|num\left(Ent(S_l)_{new}-Ent(S_l)\right)=\left[\sum_{j-1}^{i=1}|S_l||C_i|\log(|C_i|)+|S_l||C_j|\log(|C_j|+1)+\right.\\ &\left.\sum_{i=j+1}^{k}|S_l||C_i|\log(|C_i|)-\underline{\sum_{i=1}^{k}|S_l||C_i|\log(|S_l|+1)}+|S_l|\log(|C_j|+1)-|S_l|\log(|S_l|+1)\right]\\ &-\left[\sum_{i=1}^{j-1}|S_l||C_i|\log(|C_i|)+|S_l||C_j|\log(|C_j|)+\sum_{i=j+1}^{k}|S_l||C_i|\log(|C_i|)-\underline{\sum_{i=1}^{k}|S_l||C_i|\log(|S_l|)}-\sum_{i=1}^{k}(|C_i|\log(|C_i|)-|C_i|\log(|S_l|))\right]\end{aligned}\right\} \tag{7}$$

式(7)中,标记为灰色的部分可以抵消,标记为下划线的部分相加后可得:

$$-\sum_{i=1}^{k}|S_l||C_i|\log\left(\frac{|S_l|+1}{|S_l|}\right)=-|S_l|\cdot\log\left(\frac{|S_l|+1}{|S_l|}\right)\cdot\sum_{i=1}^{k}|C_i|=-|S_l|^2\cdot\log\left(\frac{|S_l|+1}{|S_l|}\right) \tag{8}$$

式(7)中余下部分整理后可得:

$$\left.\begin{aligned}&|S_l||C_j|\log(|C_j|+1)+|S_l|\log(|C_j|+1)-|S_l|\log(|S_l|+1)-|S_l||C_j|\log(|C_j|)-\sum_{i=1}^{k}(|C_i|\log(|C_i|)-|C_i|\log(|S_l|))\\ &=|S_l||C_j|\log\left(\frac{|C_j|+1}{|C_j|}\right)+|S_l|\log\left(\frac{|C_j|+1}{|S_l|+1}\right)-|S_l|\cdot\sum_{i=1}^{k}\frac{|C_i|}{|S_l|}\log\left(\frac{|C_i|}{|S_l|}\right)\\ &=|S_l||C_j|\log\left(\frac{|C_j|+1}{|C_j|}\right)+|S_l|\log\left(\frac{|C_j|+1}{|S_l|+1}\right)-|S_l|Ent(S_l)\end{aligned}\right\} \tag{9}$$

合并式(8)、式(9),并将分母分子合并即可得:

$$Ent(S_l)_{new}-Ent(S_l)=\frac{|S_l|\cdot\log\left(\frac{|S_l|+1}{|S_l|}\right)-|C_j|\cdot\log\left(\frac{|C_j|+1}{|C_j|}\right)-\log\left(\frac{|C_j|+1}{|S_l|+1}\right)+Ent(S_l)}{|S_l|+1}.$$

A1.2 信息熵批量增量公式推导

对于地点 l,假设新到来 m 个点,这些点非平均地分布在 s 个类中:$\bar{C}_1,\bar{C}_2,\ldots,\bar{C}_s$,每个类对应的新增点数量为 $a_1,a_2,\ldots,a_s$ 个,$a_1+a_2+\ldots+a_s=m$,未被更新的类为 $C_1,C_2,\ldots,C_r$,$r+s=k$,k 为类的总数量,则信息熵的变化如下:

$$Ent(S_l)_{new}=-\left(\sum_{i=1}^{r}\frac{|C_i|}{|S_l|+m}\log\left(\frac{|C_i|}{|S_l|+m}\right)+\sum_{j=1}^{s}\frac{|\bar{C}_j|+a_j}{|S_l|+m}\log\left(\frac{|\bar{C}_j|+a_j}{|S_l|+m}\right)\right).$$

类似信息熵增量公式的推导过程,$Ent(S_l)_{new}$–$Ent(S_l)$后可得到分母与分子分别为

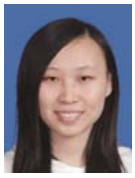
$$\left.\begin{aligned}
&den\left(Ent\left(S_l\right)_{new}-Ent\left(S_l\right)\right)=-\left(\left|S_l\right|+m\right)\left|S_l\right|num\left(Ent\left(S_l\right)_{new}-Ent\left(S_l\right)\right)\\
&=\left[\sum_{i=1}^{r}\left|S_l\right|\left|C_i\right|\log\left(\left|C_i\right|\right)+\sum_{j=1}^{s}\left|S_l\right|\left|\bar{C}_j\right|\log\log\left(\left\|\bar{C}_j\right\|+a_j\right)\right.\\
&\underline{\left.-m\sum_{i=1}^{k}\left|S_l\right|\left|C_i\right|\log\left(\left|S_l\right|+m\right)}+\left|S_l\right|\log\left(\left|C_j\right|+a_j\right)-\left|S_l\right|\log\left(\left|S_l\right|+m\right)\right]\\
&-\left[\sum_{i=1}^{r}\left|S_l\right|\left|C_i\right|\log\left(\left|C_i\right|\right)+\sum_{j=1}^{s}\left|S_l\right|\left\|\bar{C}_j\right\|\log\left(\left\|\bar{C}_j\right\|\right)-\underline{m\sum_{i=1}^{k}\left|S_l\right|\left|C_i\right|\log\left(\left|S_l\right|\right)}\right.\\
&\left.-m\sum_{i=1}^{k}\left(\left|C_i\right|\log\left(\left|C_i\right|\right)-\left|C_i\right|\log\left(\left|S_l\right|\right)\right)\right]
\end{aligned}\right\}\tag{10}$$

同理,式(10)中灰色部分可抵消,将下划线部分合并,整理余项并将分母分子合并后就可得到:

$$Ent\left(S_l\right)_{new}-Ent\left(S_l\right)=\frac{M}{\left|S_l\right|+m},$$

其中,

$$M=\left|S_l\right|\cdot\log\left(\frac{\left|S_l\right|+m}{\left|S_l\right|}\right)+mEnt\left(S_l\right)-\sum_{j=1}^{s}\left(\left|\bar{C}_j\right|\cdot\log\left(\frac{\left|\bar{C}_j\right|+a_j}{\left|\bar{C}_j\right|}\right)+a_j\log\left(\frac{\left|\bar{C}_j\right|+a_j}{\left|S_l\right|+m}\right)\right).$$



吴珉(1986－),女,云南昆明人,硕士,主要研究领域为轨迹数据管理.

唐祖锴(1977－),男,博士,副教授,CCF 专业会员,主要研究领域为软件工程,物联网工程,数据库技术.

祝园园(1984－),女,博士,副教授,CCF 专业会员,主要研究领域为数据库,数据挖掘.

彭煜玮(1980－),男,博士,副教授,CCF 专业会员,主要研究领域为时空数据管理,数据库管理系统,高端制造业大数据管理.

彭智勇(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为复杂数据管理,可信数据管理,Web 数据管理.