计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(11):212−221 [doi: 10.15888/j.cnki.csa.009289]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于划分的自适应轨迹拐点提取压缩算法①

郑汉捷[1,2,3], 邬群勇[1,2,3], 尹延中[1,2,3], 王涵菁[1,2,3], 张　晨[1,2,3]

[1](空间数据挖掘与信息共享教育部重点实验室 (福州大学), 福州 350108)
[2](数字中国研究院 (福建), 福州 350003)
[3](卫星空间信息技术综合应用国家地方联合工程研究中心, 福州 350108)
通信作者: 邬群勇, E-mail: qywu@fzu.edu.cn

摘　要: 海量的轨迹数据为管理分析和数据挖掘工作带来了巨大的挑战, 轨迹压缩技术成为解决这一问题的一种有效方案. 针对目前多数轨迹压缩算法需要人为干预设定阈值的问题, 融合特征聚类与轨迹划分的思想提出了一种自适应的轨迹拐点提取压缩算法. 算法从轨迹的全局方向特征与局部方向特征出发考虑, 依次进行了轨迹粗划分、子轨迹合并以及轨迹细划分的工作. 实验结果显示, 随着轨迹规模的增大, 与其他算法相比, 该算法基本能够在保持更高压缩率的同时产生更低的方向误差. 提出的算法具有自适应和高精度拐点识别的优势, 在其他轨迹压缩场景之下仍有着较高的参考价值.

关键词: 轨迹数据; 轨迹压缩; 聚类簇

引用格式: 郑汉捷,邬群勇,尹延中,王涵菁,张晨.基于划分的自适应轨迹拐点提取压缩算法.计算机系统应用,2023,32(11):212–221. http://www.c-s-a.org.cn/1003-3254/9289.html

Adaptive Trajectory Inflexion Extraction and Compression Algorithm Based on Partition

ZHENG Han-Jie[1,2,3], WU Qun-Yong[1,2,3], YIN Yan-Zhong[1,2,3], WANG Han-Jing[1,2,3], ZHANG Chen[1,2,3]

[1](Key Lab of Spatial Data Mining & Information Sharing of Ministry of Education (Fuzhou University), Fuzhou 350108, China)
[2](The Academy of Digital China (Fujian), Fuzhou 350003, China)
[3](National & Local Joint Engineering Research Center of Satellite Geospatial Information Technology, Fuzhou 350108, China)

Abstract: Massive trajectory data pose challenges to management analysis and data mining, and trajectory compression technology has become an effective solution to this problem. Aiming at the problem that most current trajectory compression algorithms need human intervention to set thresholds, this study proposes an adaptive trajectory inflection point extraction and compression algorithm which combines the idea of feature clustering and trajectory partition. Based on the global and local direction characteristics of the trajectory, the algorithm carries out the rough trajectory division, sub-trajectory merging, and fine trajectory division. The experimental results show that with the increasing trajectory size, the proposed algorithm can produce lower direction error and maintain a higher compression rate than other algorithms. The algorithm features adaptive and high-precision inflection point recognition and still has a high reference value under other trajectory compression scenarios.

Key words: trajectory data; trajectory compression; clustering group

随着定位技术和移动通信技术的快速发展和应用普及, 轨迹数据的获取变得越发容易, 已经在不同的领域突显出重要的研究价值[1]. 然而, 较高的采样频率会产生大量的轨迹记录, 从而导致轨迹数据规模的爆炸式增长. 对于一线城市而言, 仅是出租车的轨迹数据量, 一天就已经能够达到 TB 级以上[2]. 这将严重影响数据

① 基金项目: 国家自然科学基金 (42201500, 41471333); 福建省科技计划引导项目 (2021H0036)
收稿时间: 2023-04-13; 修改时间: 2023-05-17; 采用时间: 2023-06-01; csa 在线出版时间: 2023-09-21
CNKI 网络首发时间: 2023-09-22

存储、管理与查询分析效率, 给服务器带来极大的压力, 为进一步的数据挖掘工作带来巨大的挑战, 且轨迹中往往包含大量的冗余数据, 导致了不必要的存储空间浪费. 为减少轨迹数据存储空间和简化轨迹数据分析, 轨迹数据的压缩存储成为加速轨迹模式挖掘的一种方式和研究热点[3]. 然而, 轨迹数据同时具有时序特征和空间特征, 常规的时序数据压缩算法并不能简单适用于轨迹数据压缩的场景, 需要充分考虑轨迹的特殊性, 设计更有效的轨迹压缩算法以满足海量轨迹数据管理的需求.

常见的轨迹压缩算法可以分为以下几类[4]: 基于路网约束的轨迹压缩, 此类算法考虑到车辆受限于道路的特点, 仅适用于车辆轨迹的压缩工作[5,6]; 基于相似度量的轨迹压缩, 此类算法通过度量轨迹间的相似性, 将相似的轨迹进行统一的压缩, 适用于存在较多相似轨迹的压缩场景[7]; 基于语义信息的轨迹压缩, 此类算法一般通过提取轨迹中的语义信息进行压缩, 具有较好的可读性, 但是将导致轨迹空间信息的丢失[8,9]; 基于特征点提取的轨迹压缩, 此类算法限制条件较少, 适用于大多数轨迹压缩场景, 本文将主要研究此类压缩算法.

基于特征点提取的轨迹压缩关键在于如何寻找到轨迹中具有代表性的特征点, 轨迹中的特征点通常具有丰富的信息能够很好地反映出轨迹的形状以及运动规律. 根据选择特征点方式的不同, 轨迹压缩算法分为基于全局特征的轨迹压缩以及基于局部特征的轨迹压缩[1]. 道格拉斯-普客算法 (Douglas-Peucker, DP)[10] 是最为经典的基于全局特征的轨迹压缩算法, 它以一种从上到下的思想, 逐步对轨迹进行迭代划分, 并使用垂直欧氏距离度量轨迹的压缩误差. Meratnia 等[11] 在 DP 算法的基础上考虑了轨迹的时空特征, 使用时间同步欧氏距离 (synchronous Euclidean distance, SED) 代替垂直欧氏距离进行轨迹迭代划分, 提出了从上到下的时间比压缩算法 (top down time ratio, TD-TR). DP 和 TD-TR 算法作为经典的全局压缩算法, 能够很好地保留轨迹的整体形状, 但是由于其递归的特性, 算法时间复杂度为$O\left(n^2\right)$. 与上述两种考虑位置的压缩算法不同, SP 算法[12] 是保留方向的轨迹压缩算法, 算法基于一个给定的方向误差构建轨迹与误差阈值的图结构, 通过找到图中的最短路径, 得到保留方向信息的轨迹. 由于选择合适的方向误差阈值对于不同的应用具有挑战性, 作者又提出了 Error-Search 算法[13], 该算法能够搜索在一定压缩率下方向误差最低的压缩轨迹. 同样这两个算法的时间复杂度也较高, 因此不适合处理大型轨迹数据集.

基于局部特征的轨迹压缩算法相比而言有着更快的执行效率, 其时间复杂度一般为$O(n)$, 对于需要实时在线的轨迹压缩需求, 基于局部特征的压缩算法较为合适. 垂距限值法和角度限值法是其中比较有代表性的方法: 通过依次遍历轨迹中的三元组, 每一步进行垂距 (或角度) 的判断决定是否保留轨迹点. 李升宏等[14] 借鉴垂距限值法和角度限值法的思想, 提出了余弦垂距判别算法 (cosine vertical distance discrimination, CVDD), 压缩效果要优于 DP 算法. 邬群勇等[15] 对轨迹进行开放角计算, 去除了轨迹中方向变化较小的中间点. Yuan 等[16] 考虑了轨迹的方向运动特征, 提出了一种基于轨迹转角的拐点提取方法. 上述方法拥有较高的执行效率, 但是难以识别出局部变化小而累积变化大的时空弯曲现象, 因此很有可能会遗漏某些重要的轨迹特征点. 田智慧等[17] 对基于轨迹转角的方法[16] 进行改进, 在角度阈值的基础上增加了累积变向角阈值, 一定程度上减少了关键特征点的丢失, 同时考虑速度特性, 使算法能识别出相对完整的特征点. 相对而言, 基于窗口的轨迹压缩算法将更能满足实时压缩的需求. STTrace 算法[18] 是一种基于 SED 进行度量的在线轨迹压缩算法, 该算法依赖于一个固定大小的缓冲区, 通过每次删除缓冲区中 SED 误差最小的轨迹点进行压缩工作.

有学者考虑将聚类算法与轨迹压缩算法相结合, 在进行压缩之前, 通过对轨迹的特征进行聚类以获取到潜在的轨迹特征点. Lin 等[19] 提出了保留轨迹速度特性的自适应轨迹压缩算法 (adaptive trajectory simplification, ATS), 该算法首先从全局的角度对轨迹的速度特征进行聚类, 确定出合理的速度间隔区间, 并基于基尼系数对轨迹进行递归切分, 切分后的子轨迹再应用自适应的 DP 算法. 张甜等[20] 在 ATS 算法的基础上进行改进, 使用轮廓系数 (silhouette coefficient, SC)[21] 确定聚类簇数, 且使用 TD-TR 算法替代 DP 算法, 考虑了轨迹的时间特性. 杨家轩等[22] 同时对轨迹的方向特征和速度特征进行聚类, 在得到潜在的轨迹特征点后, 使用信息熵对轨迹进行切分, 切分后应用 TD-TR 算法实现二次压缩. 上述方法的优势在于能够较好地保留轨迹的运动特征, 但是实现较为复杂, 且在对轨

迹划分后又应用了 DP 或 TD-TR 算法, 因此也具有较高的时间复杂度.

文献 [12] 已证明保留轨迹方向的压缩 (direction-preserving trajectory simplification, DPTS) 比起保留轨迹位置的压缩 (position-preserving trajectory simplification, PPTS) 而言一般有着更好的压缩效果. 使用 DPTS 算法进行轨迹压缩, 将保证压缩后轨迹在方向上的误差较小, 同时在位置上的误差也能被控制在一定范围内. 相反, PPTS 却无法保证方向误差. 因此, 本文所设计的轨迹压缩算法将考虑对轨迹方向信息的保留, 也属于一种 DPTS 算法.

针对目前大多数轨迹压缩算法存在的需要人为选择阈值的问题, 本文从轨迹的全局方向特征和局部方向特征出发考虑, 提出一种自适应的轨迹拐点提取压缩算法, 在避免人为选择参数的同时, 实现高精度的自适应轨迹拐点提取.

1 轨迹相关定义

轨迹压缩问题定义如下: 给定一条轨迹 TR, 描述为 $TR=\{p_1,p_2,\cdots,p_i,\cdots,p_n\}\ (1\leqslant i\leqslant n)$, 其中 p_i 为轨迹点, 描述为一个三元组 $p_i=\langle lon_i,lat_i,t_i\rangle\ (t_1<t_2<\cdots<t_i<t_n)$, 包括轨迹点的经度、纬度和采样时间戳, 一条轨迹的总采样点数则描述为 $|TR|$. 推导出一条压缩轨迹 $TR_{\text{sim}}=\left\{p_1,p_{s_1},p_{s_2},\cdots,p_{s_m},p_n\right\}(1<s_1<s_2<s_m<n)$, 其中轨迹起始点 p_1 和 p_n 必定被保留在轨迹中, $p_{s_i}\ (1\leqslant i\leqslant m)$ 则代表压缩后保留的轨迹的特征点.

在轨迹 TR 中, 按序列顺序取任意两个轨迹点 p_i 和 $p_j\ (1\leqslant i<j\leqslant n)$ 之间的轨迹点组成的新轨迹, 称为子轨迹, 描述为 $STR=\left\{p_i,p_{i+1},\cdots,p_j\right\}(j>i+1)$. 其中 p_i 和 p_j 为子轨迹的边界轨迹点, $\text{TRS}_i=\overline{p_ip_{i+1}}$ 和 $\text{TRS}_{j-1}=\overline{p_{j-1}p_j}$ 为子轨迹的边界轨迹段.

依次计算轨迹 TR 中 2 个相邻轨迹点的方向向量, 得到一条长度为 $n-1$ 的方向向量序列 $\left\{\overrightarrow{\text{TRS}_1},\overrightarrow{\text{TRS}_2},\cdots,\overrightarrow{\text{TRS}_{n-1}}\right\}$, 其中每一项为轨迹段方向, 描述为 $\overrightarrow{\text{TRS}_i}=(p_{i+1}\cdot lon-p_i\cdot lon,\ p_{i+1}\cdot lat-p_i\cdot lat)$.

2 自适应轨迹拐点提取压缩算法

综合轨迹特征聚类的思想[20] 和无监督的序列划分方法[23] 提出一种自适应的轨迹拐点提取压缩算法, 算法流程如图 1 所示. 首先, 从轨迹的全局方向特征进行考虑, 通过计算得到轨迹的方向向量序列; 再确定合适的类簇数, 使用聚类算法以方向向量序列中的每一项为基本单元进行聚类, 将轨迹映射成一条类簇标签序列, 在标签变化处进行轨迹划分得到了粗划分下的子轨迹序列; 其次, 遍历子轨迹序列, 搜索到仅有两个轨迹点的子轨迹作为待合并子轨迹, 通过判断基于余弦相似性的子轨迹合并条件来决定是否将其与相邻子轨迹进行合并, 完成对所有子轨迹的判断后得到合并后的子轨迹序列; 之后, 对于每一条子轨迹的边界轨迹段, 计算其轮廓系数以决定是否对子轨迹的边界进行移动, 完成对所有子轨迹的判断后得到边界移动后的子轨迹序列. 相邻的子轨迹间共享一个边界轨迹点, 提取此类轨迹点即为轨迹的拐点. 最后保留轨迹的起点、拐点和终点, 其轨迹序列为压缩后轨迹.

2.1 基于轨迹方向特征聚类的轨迹粗划分

本文使用 K-means 对轨迹的方向向量进行聚类, 以获得具有代表性的方向间隔. 聚类要求输入一个 k 值以表示数据分类簇, 一般可以根据先验知识或肘部法[19] 得出. 本文将轨迹的行驶方向划分为 8 个核心方向, 方位角依次为 0, 45°,⋯, 315°. 以核心方向作为区间中心, 8 个核心方向恰好将行驶方向划分为 8 个区间, 分别为 [337.5°, 360°]∪[0, 22.5°), [22.5°, 67.5°), ⋯, [292.5°, 337.5°). 据此, 给出了确定 k 值的方法: 对于一条轨迹 TR 的方向向量序列, 计算每一个向量的方位角, 记录每一个向量方位角所属的方位角区间, 最后统计向量所属的方位角区间总个数, 将其作为 K-means 聚类的 k 值.

聚类算法中的核心步骤在于针对数据集中的每一个向量, 计算它到 k 个聚类中心的距离并将其分配到距离最小的聚类中心所对应的类中, 通常使用欧氏距离度量向量与聚类中心的距离. 针对轨迹方向向量进行聚类, 倾向于计算向量之间的方向相似性, 因此本文使用余弦距离进行距离度量, 计算公式如式 (1) 所示. 余弦距离的取值范围为[0,2], 越趋近于 0, 表示向量之间的方向更加吻合.

$$cosineDistance(x,y)=1-\frac{\sum_{i=1}^{n}x_iy_i}{\sqrt{\sum_{i=1}^{n}x_i^2}\sqrt{\sum_{i=1}^{n}y_i^2}} \tag{1}$$

其中, x 和 y 为两个向量, x_i 和 y_i 分别描述向量 x 和 y 中的每一个分量.

基于余弦距离对轨迹的方向向量序列进行聚类后,

每一个方向向量被分配了一个标签以标识其所属的类别, 一条轨迹映射为一条标签序列, 在标签变化处进行轨迹划分实现轨迹的粗划分.

对轨迹粗划分的时间复杂度进行分析. 这部分的算法时间复杂度基本由 K-means 算法决定, 为O(*lnk*), 其中 *l* 为迭代次数, *n* 为轨迹段数, *k* 为类簇数.

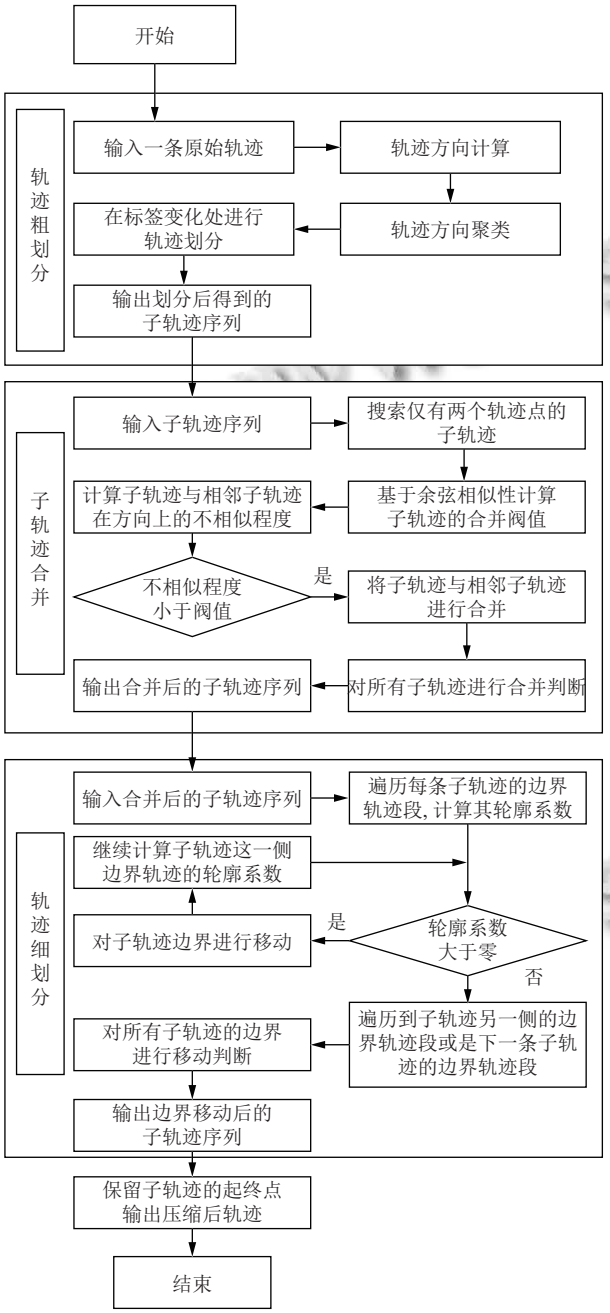


图 1 算法框架

2.2 基于余弦相似性的子轨迹合并

通过分析发现, 粗划分后的子轨迹序列中的部分子轨迹仅包含两个轨迹点, 对于这种情况, 需要考虑将该子轨迹与其相邻的子轨迹进行合并. 对轨迹方向特征聚类后会得到一系列的类簇划分边界, 如果某一向量距离划分边界越近, 则说明该向量距离其相邻的类簇也越近, 认为该向量与其相邻的类簇中的部分向量在一定条件下也是相似的, 称之是可匹配的. 本文定义一个阈值以判断向量是否是可匹配的, 阈值将根据不同的向量进行自适应计算, 如式 (2) 所示:

$$mergeThreshold(x) = \max_{x^i \in cluster_x}\{cosineDistance(x, x^i)\} \tag{2}$$

其中, x 为某一向量, x^i为与 x 属于同一类簇$cluster_x$的向量, 向量间的相似性度量使用余弦距离.

根据向量在所属类簇中的位置自适应计算合适的子轨迹合并阈值, 如果向量靠近类簇的划分边界, 那么将会得到较大的阈值; 反之, 则会得到较小的阈值. 计算待合并子轨迹与其相邻子轨迹方向的不相似程度, 将计算的结果与阈值进行对比, 若小于阈值, 则进行合并, 否则不进行合并. 越大的阈值将表现出越强的子轨迹合并倾向, 而越小的阈值将表现出越弱的子轨迹合并倾向. 通过考虑子轨迹间的局部方向相似情况, 对子轨迹进行适当的合并, 能够有效地减少某些潜在错误的轨迹划分结果, 一定程度上提高了轨迹划分的准确性. 子轨迹合并的伪代码如算法 1 所示.

算法 1. 子轨迹合并算法

输入: 子轨迹序列$STRS = [\{p_1,\cdots,p_i\},\{p_i,\cdots,p_j\},\cdots,\{p_m,\cdots,p_n\}](1<i<j<\cdots<m<n)$

输出: 原地得到合并处理后的子轨迹序列 STRS

1) For *STR* from $\{p_1,\cdots,p_i\}$ to $\{p_m,\cdots,p_n\}$: /*遍历每一条子轨迹*/
2) If $|STR| == 2$: /*如果当前子轨迹仅有两个轨迹点 (一个轨迹段), 需要判断合并条件*/
3) 计算 *STR* 中唯一轨迹段方向向量$\overrightarrow{\text{TRS}}$的合并阈值$mergeThreshold(\overrightarrow{\text{TRS}})$
4) 计算 *STR* 与其相邻子轨迹STR_{pre}和STR_{next}在方向上的不相似程度 (分别计算$\overrightarrow{\text{TRS}}$与$STR_{\text{pre}}$及$STR_{\text{next}}$中所有轨迹段方向的余弦距离的均值), 并取二者中的最小值 minSim
5) If minSim≤$mergeThreshold(\overrightarrow{\text{TRS}})$:
6) 将 *STR* 与STR_{pre}和STR_{next}中方向更相近的那条子轨迹进行合并
7) Else:
8) 不对 *STR* 进行合并处理
9) Else:
10) Continue

对子轨迹合并的时间复杂度进行分析. 该部分的时间复杂度将主要由计算子轨迹合并阈值以及计算当

前子轨迹与相邻子轨迹的不相似程度决定. 计算合并阈值的时间复杂度为O(c), 计算当前子轨迹与相邻子轨迹的不相似程度的时间复杂度为O($p+q$), 总时间复杂度为O($n(c+p+q)$), 其中 n 为子轨迹数, c 为与当前待合并子轨迹内唯一方向向量属于同一类的向量数, p 和 q 分别为与当前待合并子轨迹相邻的两条子轨迹中的轨迹段数.

2.3 基于轮廓系数的轨迹细划分

子轨迹合并之后, 将对每一条子轨迹的边界进行调整移动以进一步提高轨迹划分的准确性. 轨迹细划分上主要考虑两点[24], 一是如何衡量子轨迹内的同质性 (即同一子轨迹内的轨迹点 (或轨迹段) 具有较为相似的特征); 二是如何衡量相邻子轨迹之间的异质性 (即相邻子轨迹之间具有不相似的运动行为). 轮廓系数[21]通过考虑数据向量的内聚度和分离度来评价聚类效果的好坏. 本文将使用轮廓系数来评价轨迹基于方向特征的划分效果, 计算如式 (3) 所示:

$$S(\overrightarrow{\mathrm{TRS}_i})=\frac{b\left(\overrightarrow{\mathrm{TRS}_i}\right)-a\left(\overrightarrow{\mathrm{TRS}_i}\right)}{\max\left\{a\left(\overrightarrow{\mathrm{TRS}_i}\right),b\left(\overrightarrow{\mathrm{TRS}_i}\right)\right\}} \tag{3}$$

其中,

$$a\left(\overrightarrow{\mathrm{TRS}_i}\right)=\frac{1}{|STR|-1}\sum_{j=1}^{|STR|-1} cosineDistance(\overrightarrow{\mathrm{TRS}_i},\overrightarrow{\mathrm{TRS}_j})$$

$$b\left(\overrightarrow{\mathrm{TRS}_i}\right)=\frac{1}{|STR_{\mathrm{nearest}}|-1}\sum_{j=1}^{|STR_{\mathrm{nearest}}|-1} cosineDistance(\overrightarrow{\mathrm{TRS}_i},\overrightarrow{\mathrm{TRS}_j})$$

其中, 轮廓系数$S(\overrightarrow{\mathrm{TRS}_i})$的值介于 [−1, 1], $S(\overrightarrow{\mathrm{TRS}_i})$越趋近于 1 代表轨迹段方向$\overrightarrow{\mathrm{TRS}_i}$的内聚度和分离度都相对较优; 反之, 则代表轨迹段方向$\overrightarrow{\mathrm{TRS}_i}$的内聚度和分离度都相对较差. $a\left(\overrightarrow{\mathrm{TRS}_i}\right)$是计算轨迹段方向的内聚度 (同质性), 即描述为当前轨迹段方向$\overrightarrow{\mathrm{TRS}_i}$与同一子轨迹内其他轨迹段方向$\overrightarrow{\mathrm{TRS}_j}$的不相似程度的平均值; $b\left(\overrightarrow{\mathrm{TRS}_i}\right)$是计算当前轨迹段方向的外离度 (异质性), 描述为当前轨迹段方向$\overrightarrow{\mathrm{TRS}_i}$与其直接相邻子轨迹内的所有轨迹段方向$\overrightarrow{\mathrm{TRS}_j}$的不相似程度的平均值. 其中相似性的度量使用余弦距离 (即式 (1)) 计算.

轨迹属于一种时间序列数据, 一条轨迹的轨迹点 (或轨迹段) 之间有着严格的先后顺序, 因此轨迹的划分边界只能进行线性的移动, 且边界移动时, 只需优先考虑位于子轨迹边界附近的轨迹点 (或轨迹段). 因此, 本文在进行轮廓系数计算时, 沿着线性方向逐个移动优先对子轨迹的边界轨迹段进行计算判断. 如图 2 所示, 一条轨迹被划分为了 3 条子轨迹, 每条子轨迹的边界轨迹点被标为深色而边界轨迹段则被标为虚线, 在进行子轨迹的边界移动时, 将对边界轨迹段 (即图中虚线部分) 先进行判断. 值得关注的是, 第 1 条子轨迹和最后一条子轨迹都只有一个边界轨迹点和一个边界轨迹段. 如图 3 所示为抽象的子轨迹边界移动示意图, 图上轨迹被划分为 3 个子轨迹 $STR(1)$、$STR(2)$ 和 $STR(3)$, 子轨迹上的每一个轨迹段抽象为图中的一个点, 且边界轨迹段被标记为深色, 子轨迹边界则用虚线表示. 子轨迹划分边界移动调整的伪代码如算法 2 所示.

算法 2. 子轨迹边界移动算法

输入: 子轨迹序列STRS = [{p_1,⋯,p_i},{p_i,⋯,p_j},⋯,{p_m,⋯,p_n}](1<i<j<⋯<m<n)

输出: 原地得到边界移动后的子轨迹序列 STRS

1) 初始化一个变量 Flag 用以标识前一子轨迹是否移动了相邻边界
2) For STR from {p_1,⋯,p_i} to {p_m,⋯,p_n}: /*遍历每一条子轨迹*/
3) If |STR| = = 2: /*子轨迹中仅有两个轨迹点 (一个轨迹段), 不进行划分边界调整*/
4) Continue
5) If STR 不是第一条子轨迹且 $Flag$==False:
6) While True:
7) If |STR| = = 2: /*经过边界调整后, 当前子轨迹中仅有两个轨迹点 (一个轨迹段)*/
8) 根据算法 1 判断 STR 的合并条件
9) Break
10) 得到 STR 中的第 1 个轨迹段$\mathrm{TRS}_{\mathrm{first}}$以及其方向$\overrightarrow{\mathrm{TRS}_{\mathrm{first}}}$
11) 计算$\overrightarrow{\mathrm{TRS}_{\mathrm{first}}}$的轮廓系数$S(\overrightarrow{\mathrm{TRS}_{\mathrm{first}}})$
12) If $S(\overrightarrow{\mathrm{TRS}_{\mathrm{first}}})\geqslant 0$: /*如果轮廓系数为正, 边界不移动*/
13) Break
14) Else: //如果轮廓系数为负, 边界右移
15) 将$\mathrm{TRS}_{\mathrm{first}}$插入到前一条子轨迹$STR_{\mathrm{pre}}$的末尾
16) 将$\mathrm{TRS}_{\mathrm{first}}$从 STR 的开头移除
17) Flag=False
18) If STR 不是最后一条子轨迹:
19) While True:
20) If |STR| = = 2: /*经过边界调整后, 当前子轨迹中仅有两个轨迹点 (一个轨迹段)*/
21) 根据算法 1 判断 STR 的合并条件
22) Break
23) 得到 STR 中的最后一个轨迹段$\mathrm{TRS}_{\mathrm{last}}$以及其方向$\overrightarrow{\mathrm{TRS}_{\mathrm{last}}}$
24) 计算$\overrightarrow{\mathrm{TRS}_{\mathrm{last}}}$的轮廓系数$S(\overrightarrow{\mathrm{TRS}_{\mathrm{last}}})$
25) If $S(\overrightarrow{\mathrm{TRS}_{\mathrm{last}}})\geqslant 0$: /*如果轮廓系数为正, 边界不移动*/

26) Break
27) Else: //如果轮廓系数为负, 边界左移
28) Flag=True //边界移动, 更新 Flag
29) 将TRS_{last}插入到后一条子轨迹STR_{next}的开头
30) 将TRS_{last}从 *STR* 的末尾移除

对轨迹细划分的时间复杂度进行分析. 该部分的时间复杂度将主要由子轨迹边界移动的次数以及轮廓系数的计算决定. 判断子轨迹边界移动条件的总次数为$2m(n-1)$, 计算一个轨迹段方向的轮廓系数的时间复杂度为$O(i+j)$, 总的时间复杂度为$O(2m(n-1)(i+j))$, 其中 n 为子轨迹数, m 为待移动边界子轨迹某一侧边界移动的次数, i 为待移动边界子轨迹内的轨迹段数, j 为与当前子轨迹的边界轨迹段直接相邻的子轨迹内的轨迹段数.

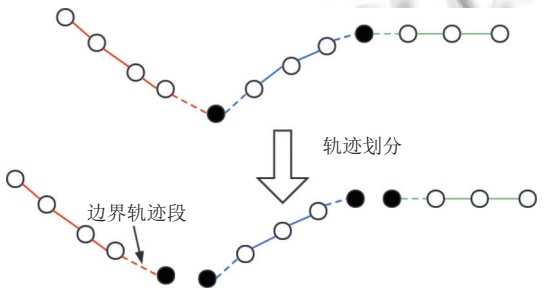


图 2 轨迹划分示意

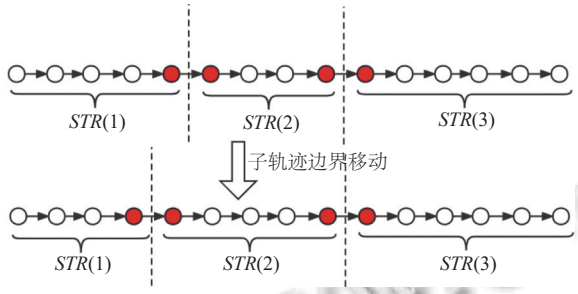


图 3 子轨迹边界移动示意

3 轨迹压缩算法性能评估

3.1 实验配置及数据集

实验配置具体如下: 处理器为 Intel(R) Core(TM) i7-10700K CPU @ 3.80 GHz 16 GB RAM; 实验环境为 Windows 10 操作系统和 Anaconda3 开发环境, 并基于 Python 3.8 实现.

为验证本文算法针对不同规模轨迹数据的方向特征提取及压缩效果, 实验数据使用 (Microsoft Research Asia) Geolife 项目[25–27] 中收集的由 182 名用户在 5 年多 (2007 年 4 月–2012 年 8 月) 的时间内产生的 GPS 轨迹. 该数据集中的 GPS 轨迹由一系列采样时间点表示, 每个点包含纬度、经度和海拔的信息, 轨迹具有较高的采样率, 91.5% 的轨迹采样密度较高.

对数据集进行简单预处理后, 提取部分轨迹作为最终的实验数据集, 并根据轨迹的点数将轨迹集分为 4 个等级, 分别为短轨迹 (10–300 个点)、中长轨迹 (300–1 000 个点)、长轨迹 (1 000–2 000 个点) 和超长轨迹集 (>2 000 个点), 数据集详细信息如表 1 所示.

表 1 数据集信息

数据集等级	轨迹数量 (条)	轨迹点数 (个)
短轨迹	2 102	10–300
中长轨迹	1 430	300–1 000
长轨迹	409	1 000–2 000
超长轨迹	247	>2 000

本文主要考虑将角度的压缩算法[15–17] 、融合聚类的压缩算法[19–20,22] 和 SP 算法[12] 这 3 类算法作为实验的对比算法. 其中基于角度的压缩算法拥有简单高效的优势, 在轨迹预处理工作中较为常见, 本文选择文献 [17] 中的算法 (不考虑速度特征) 作为代表此类的对比算法; 融合聚类的压缩算法与本文算法的实现思路有一定的相似之处, 都采用聚类的手段先对轨迹特征进行聚类以发现潜在的轨迹特征点, 本文选择文献 [22] 中的方法 (不考虑速度特征, 且不对划分后的轨迹进行二次压缩) 作为代表此类的对比算法; SP 算法属于该领域较为经典的算法, 文献 [12] 中首次证明了保留方向信息的压缩算法能够在保持方向误差的同时限制空间距离误差, 因此本文选择文献 [12] 中 SP 算法作为对比算法.

3.2 性能度量指标

本文选取运行时间、压缩率以及平均方向误差 3 个指标来评估所提出算法的性能. 运行时间通常用来衡量一个算法的执行效率; 压缩率通过计算压缩后的轨迹点数与原始轨迹点数的差距来评价压缩程度; 平均方向误差用来评价轨迹压缩前后在方向这一运动特征上的损失程度, 同时也可以很好地衡量轨迹拐点提取的精度.

本文定义压缩率的计算如式 (4) 所示:

$$compressionRatio(TR, TR_{\text{sim}}) = \left(1 - \frac{|TR_{\text{sim}}| - 2}{|TR| - 2}\right) \times 100\% \tag{4}$$

其中, TR 与 TR_{sim} 分别代表原始轨迹和压缩后轨迹,

$|TR|$与$|TR_{\text{sim}}|$则分别代表压缩前后轨迹中所包含的点数. 压缩率越高表示压缩后轨迹中所包含的点数越少, 从而压缩程度也越深.

本文定义平均方向误差的计算如式 (5) 所示:

$$aveDir_{\text{error}}(TR, TR_{\text{sim}}) = \frac{1}{|TR|-1}\sum_{i=1}^{|TR_{\text{sim}}|-1}\sum_{j=1}^{m}\theta(\overrightarrow{\text{TRS}_j}, \overrightarrow{\text{TRS}_{\text{sim}_i}}) \tag{5}$$

其中, $|TR|-1$与$|TR_{\text{sim}}|-1$分别代表轨迹压缩前后所包含的轨迹段数, m 代表原始轨迹 TR 在压缩轨迹TR_{sim}第 i 个轨迹段上的冗余轨迹段数, $\theta(\overrightarrow{\text{TRS}_j}, \overrightarrow{\text{TRS}_{\text{sim}_i}})$描述$\overrightarrow{\text{TRS}_j}$和$\overrightarrow{\text{TRS}_{\text{sim}_i}}$的向量夹角.

3.3 算法性能指标对比分析

如图 4 所示, 对 4 种算法在同一压缩率水平下的运行时间进行了对比. 整体来看, 4 种算法的运行时间都随着轨迹规模的增大而提高. 其中, 基于转角的算法的运行时间变化幅度较小, 最高耗时不超过 100 ms, 因此, 该算法有着最高的运行效率; 相对而言, SP 算法在运行时间上变化幅度较大, 随着轨迹规模的增大, 与其余 3 种算法的差距逐渐明显, 说明 SP 算法并不适用于规模较大的轨迹压缩工作; 本文算法介于中间, 比起基于信息熵的算法和 SP 算法耗时较低.

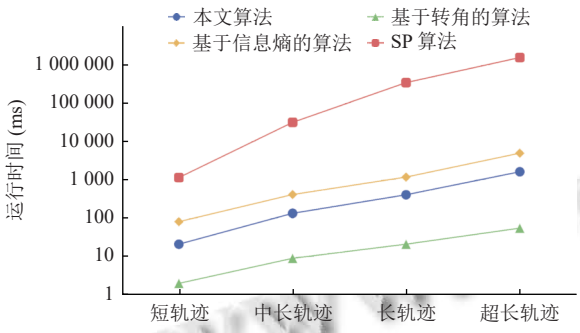


图 4　同一压缩率水平下的运行时间对比

如图 5 所示, 对 4 种算法在同一方向误差水平下的压缩率进行了对比. 整体来看, 4 种算法的压缩率都随着轨迹规模的增大而提高. 其中, SP 算法与本文算法的压缩率较为相近, 最高都超过了 80%, 本文算法的压缩率仅略高于 SP 算法. 另外, 随着轨迹规模的增大与其余两种算法的差距也越来越大. 使用本文算法进行压缩的压缩率在任何的轨迹规模下都基本高于另外 3 种算法, 说明本文算法所提取的轨迹拐点精度较高, 能用更少的轨迹点保留相当的轨迹方向信息.

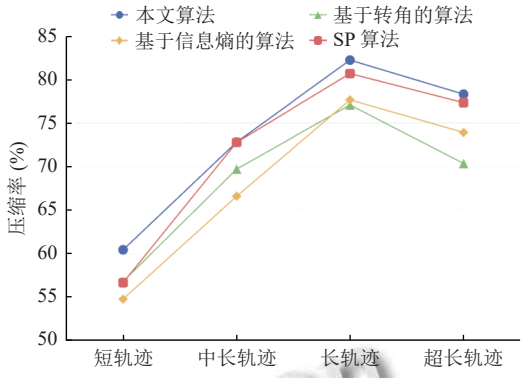


图 5　同一方向误差水平下的压缩率对比

如图 6 所示, 对 4 种算法在同一压缩率水平下的方向误差进行了对比. 整体来看, 基于转角的算法和基于信息熵的算法所产生的方向误差都随着轨迹规模的增大而提高; 而 SP 算法和本文算法产生的方向误差变化较为稳定, 基本能保持在 5°以下, 也与另外两种算法的差距随着轨迹规模的增大而逐渐明显. 使用本文算法进行压缩产生的方向误差在任何轨迹规模下都基本低于另外 3 种算法, 这再一次说明本文算法提取的轨迹拐点精度较高, 能用相当的轨迹点保留更多的轨迹方向信息.

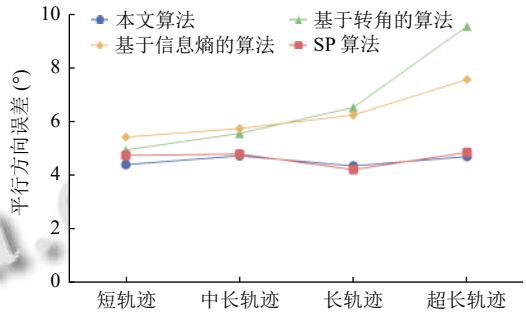
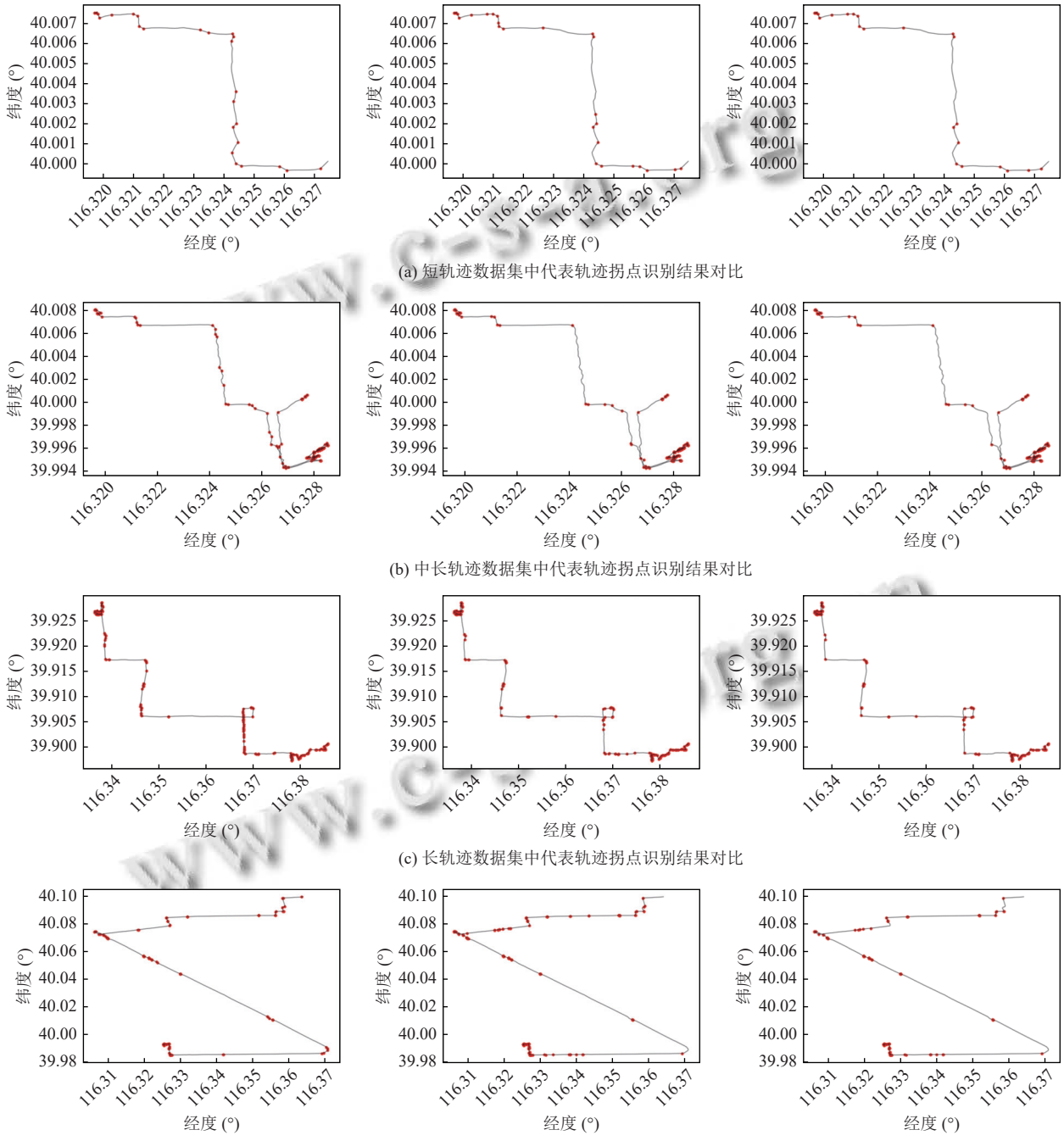


图 6　同一压缩率水平下的平均方向误差对比

综上, 基于转角的算法有着较高的执行效率, 但是压缩效果一般, 因此可应用于对压缩性能要求不高的场景; SP 算法在压缩效果上较好, 但是由于其高昂的运行耗时, 仅适用于短轨迹集或中长轨迹集; 基于信息熵的算法在本文的实验中表现不突出, 随着数据集规模的增大, 压缩效果逐渐超过基于转角的算法; 本文算法在压缩效果上基本与 SP 算法持平, 然而在算法耗时上却比 SP 算法有着明显优势, 且本文算法无需额外的输入参数, 能够根据轨迹自身的特点自适应识别轨迹拐点并压缩, 因此适用于更多的轨迹压缩场景.

3.4 自适应的轨迹拐点识别结果分析

为证明本文所提算法对于轨迹方向特征点识别的有效性, 分别可视化呈现了短轨迹数据集、中长轨迹数据集、长轨迹数据集和超长轨迹数据集中的一条代表轨迹的拐点识别结果, 每条轨迹的点数分别为 77、494、1 283 和 2 619, 并再一次将文献 [17] 中使用的基于轨迹转角的方向特征提取算法 (不考虑速度特征) 作为对比算法进行实验分析, 该算法需要两个参数分别为转角阈值和累积转角阈值, 结果如图 7 所示, 每一行为一条代表轨迹, 从左到右依次为本文算法、基于转角的算法 (取较小阈值) 和基于转角的算法 (取较大阈值) 对应的结果; 算法的有关度量指标如表 2 所示.



(a) 短轨迹数据集中代表轨迹拐点识别结果对比

(b) 中长轨迹数据集中代表轨迹拐点识别结果对比

(c) 长轨迹数据集中代表轨迹拐点识别结果对比

(d) 超长轨迹数据集中代表轨迹拐点识别结果对比

图 7 代表轨迹拐点提取可视结果对比

表 2 代表轨迹拐点提取效果评估表

轨迹压缩算法	短轨迹代表		中长轨迹代表		长轨迹代表		超长轨迹代表	
	压缩率 (%)	平均方向误差 (°)	压缩率 (%)	平均方向误差 (°)	压缩率 (%)	平均方向误差 (°)	压缩率 (%)	平均方向误差 (°)
本文算法	64.0	3.9	63.2	6.3	83.1	5.0	89.0	4.1
基于转角的算法 (取较小阈值)	62.7	5.2	61.4	6.5	80.8	5.5	83.1	4.6
基于转角的算法 (取较大阈值)	66.7	5.3	64.0	7.2	84.9	7.1	86.2	6.5

综合分析图 7 和表 2 所示的结果, 以短轨迹数据集中的一条代表轨迹为例, 使用本文算法可以在保证平均方向误差为 3.9°的同时, 达到 64.0% 的压缩率; 使用基于转角的算法 (取较小阈值) 得到了更低的压缩率, 保留了更多的轨迹点, 但是产生的平均方向误差却高于本文算法的结果, 为 5.2°, 说明算法识别出的轨迹拐点有部分是冗余的, 且同时存在遗漏部分重要拐点的情况; 使用基于转角的算法 (取较大阈值) 得到了更高的压缩率, 不存在明显的冗余拐点识别, 却也遗漏了轨迹中部分重要的轨迹拐点, 因此平均方向误差也相对较高, 为 5.3°. 通过分析, 对于基于转角的算法而言, 继续调小阈值能够将方向误差降至与本文算法相当, 但却会导致更低的压缩率, 减少压缩的程度. 所以, 这也能再次说明本文算法识别轨迹拐点的准确性.

本文算法能够识别出较为准确的轨迹拐点, 没有明显的拐点遗漏或是识别过多冗余拐点的情况, 能够在保持较高压缩率的同时产生较少的方向误差. 而文献 [17] 方法的拐点识别结果受阈值影响较大, 对于同一条轨迹而言, 取不同的阈值可能会出现遗漏某些关键拐点或是识别出许多不具有代表性的拐点的情况, 从而导致不准确的拐点提取结果. 且对于不同的轨迹, 需要选择不同的阈值才能分别实现较好的拐点识别效果, 在实际应用中, 由于轨迹数据集海量的特点, 几乎不可能以人为的方式为不同的轨迹设置不同的参数阈值, 通常只能为完整的轨迹数据集确定一个相对不错的阈值, 这也将导致部分轨迹的压缩效果较好而另一部分轨迹的压缩效果较差的情况.

大多需要进行人为干预选择参数的轨迹压缩算法都存在上述类似问题, 而本文所提出的自适应轨迹拐点提取压缩算法却很好地解决了这个问题, 能够根据轨迹自身的特点, 自适应地在合适的位置识别出轨迹拐点, 且识别的结果较准确, 得到了较好的压缩效果.

4 结束语

本文针对基于方向特征的移动对象原始轨迹压缩问题, 融合特征聚类和轨迹划分的思想研发了一种自适应的轨迹拐点提取压缩算法. 算法从每条轨迹的全局方向特征和局部方向特征出发, 能够根据每条轨迹不同的方向变化情况, 进行自适应的拐点识别, 多数情况轨迹拐点识别的效果也较为准确. 本文算法具有自适应和高精度拐点识别的优势, 在其他轨迹压缩场景之下仍有着较高的参考价值, 本文算法也可应用于轨迹划分工作.

参考文献

1 梁明, 陈文静, 段平, 等. 轨迹压缩的典型方法评价. 测绘通报, 2019, (4): 60–64, 70.

2 Chen AB, Liu LF. An online trajectory compression based on retrace point detection. Proceedings of the 20th IEEE International Conference on Communication Technology. Nanning: IEEE, 2020. 1478–1482.

3 He XR, Kempe D. Stability of influence maximization. Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2014. 1256–1265.

4 Sun PH, Xia SX, Yuan G, *et al*. An overview of moving object trajectory compression algorithms. Mathematical Problems in Engineering, 2016, 2016: 6587309.

5 Chen C, Ding Y, Xie XF, *et al*. *TrajCompressor*: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change. IEEE Transactions on Intelligent Transportation Systems, 2020, 21(5): 2012–2028. [doi: 10.1109/TITS.2019.2910591]

6 苏建花, 赵旭俊, 蔡江辉. 采用轨迹压缩和路网划分的车辆异常轨迹检测. 小型微型计算机系统, 2022, 43(7): 1438–1444.

7 Makris A, da Silva CL, Bogorny V, *et al*. Evaluating the

effect of compressing algorithms for trajectory similarity and classification problems. Geoinformatica, 2021, 25(4): 679–711. [doi: 10.1007/s10707-021-00434-1]

8 周燕. 基于 Spark 的语义轨迹频繁模式提取方法及其应用 [硕士学位论文]. 武汉: 湖北工业大学, 2020.

9 Tang J, Liu LF, Wu JG. A trajectory partition method based on combined movement features. Wireless Communications and Mobile Computing, 2019, 2019: 7803293.

10 Douglas DH, Peucker TK. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: The International Journal for Geographic Information and Geovisualization, 1973, 10(2): 112–122. [doi: 10.3138/FM57-6770-U75U-7727]

11 Meratnia N, de By RA. Spatiotemporal compression techniques for moving point objects. Proceedings of the 9th International Conference on Extending Database Technology. Heraklion: Springer, 2004. 765–782.

12 Cheng L, Wong RCW, Jagadish HV. Direction-preserving trajectory simplification. Proceedings of the VLDB Endowment, 2013, 6(10): 949–960. [doi: 10.14778/2536206.2536221]

13 Cheng L, Wong RCW, Jagadish HV. Trajectory simplification: On minimizing the direction-based error. Proceedings of the VLDB Endowment, 2014, 8(1): 49–60. [doi: 10.14778/2735461.2735466]

14 李升宏, 耿生玲, 田立勤, 等. 矢量轨迹有损压缩余弦垂距判别法. 西安邮电大学学报, 2021, 26(6): 72–81.

15 邬群勇, 王祥健. 出租车轨迹数据的频繁轨迹识别. 测绘通报, 2021, (11): 70–75.

16 Yuan G, Xia SX, Zhang L, *et al*. An efficient trajectory-clustering algorithm based on an index tree. Transactions of the Institute of Measurement and Control, 2012, 34(7): 850–861. [doi: 10.1177/0142331211423284]

17 田智慧, 马占宇, 魏海涛. 基于密度核心的出租车载客轨迹聚类算法. 计算机工程, 2021, 47(2): 133–138.

18 Potamias M, Patroumpas K, Sellis T. Sampling trajectory streams with spatiotemporal criteria. Proceedings of the 18th International Conference on Scientific and Statistical Database Management. Vienna: IEEE, 2006. 275–284.

19 Lin CY, Hung CC, Lei PR. A velocity-preserving trajectory simplification approach. Proceedings of the 2016 Conference on Technologies and Applications of Artificial Intelligence. Hsinchu: IEEE, 2016. 58–65.

20 张甜, 杨智应. 基于分段的移动对象轨迹简化算法. 计算机应用研究, 2019, 36(7): 2044–2048.

21 Rousseeuw PJ. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics, 1987, 20: 53–65. [doi: 10.1016/0377-0427(87)90125-7]

22 杨家轩, 马令琪. 基于信息熵的船舶轨迹自适应分段压缩算法. 上海海事大学学报, 2022, 43(2): 7–13, 73.

23 Leiva LA, Vidal E. Warped K-means: An algorithm to cluster sequentially-distributed data. Information Sciences, 2013, 237: 196–210. [doi: 10.1016/j.ins.2013.02.042]

24 郭炜强, 赵卓峰, 李征宇, 等. REGRASP: 一种反应式 GRASP 的无监督轨迹分段方法. 西北师范大学学报 (自然科学版), 2020, 56(6): 44–52, 62.

25 Zheng Y, Zhang LZ, Xie X, *et al*. Mining interesting locations and travel sequences from GPS trajectories. Proceedings of the 18th International Conference on World Wide Web. Madrid: ACM Press, 2009. 791–800.

26 Zheng Y, Li QN, Chen YK, *et al*. Understanding mobility based on GPS data. Proceedings of the 10th International Conference on Ubiquitous Computing. Seoul: ACM Press, 2008. 312–321.

27 Zheng Y, Xie X, Ma WY. GeoLife: A collaborative social networking service among user, location and trajectory. IEEE Data Engineering Bulletin, 2010, 33(2): 32–39.

(校对责编: 牛欣悦)