iBTC:一种基于独立森林的移动对象轨迹聚类算法

张怀峰 皮德常 董玉兰

(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 移动对象轨迹聚类在城市规划、公共空间设计、移动对象行为预测等领域具有重要的理论指导意义和实际应用价值。针对传统聚类算法(如k-means,DBSCAN)在移动对象轨迹方面聚类效果不佳的问题,提出一种新的轨迹聚类算法iBTC。该算法首先对轨迹进行分段,根据最小描述长度原理,将轨迹分段问题转换为求无向图的最短路径问题,使用Dijkstra算法求得轨迹的最佳分段;然后将轨迹聚类问题转换为一种特殊的异常检测问题,并基于独立森林的思想,使用细分-合并过程对轨迹数据进行聚类;最后在模拟数据集和监控视频记录的行人轨迹公开数据集上进行实验,结果表明该算法能够取得较好的聚类效果。

关键词 移动对象,聚类,独立森林,轨迹分段

中图法分类号 TP309 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.01.039

iBTC:A Trajectory Clustering Algorithm Based on Isolation Forest

ZHANG Huai-feng PI De-chang DONG Yu-lan

(College of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 210016,China)

Abstract The clustering of moving object trajectories has important theoretical and practical value in the fields of urban planning,public space design and mobile object behavior prediction. In this paper,aiming at the poor clustering effect of traditional clustering algorithms (e.g.,k-means,DBSCAN) on moving object trajectories,a new trajectory clustering algorithm named iBTC was proposed. The algorithm segments the trajectory firstly,and based on the principle of minimum description length,the trajectory segmentation problem is modeled as the shortest path problem for undirected graphs. Then,Dijkstra algorithm is used to find the optimal segmentation. Next,based on the idea of Isolation Forest,the trajectory clustering problem is transformed into a special anomaly detection problem,and the algorithm uses partition-merging procedure to cluster trajectory data. Finally,experiments were performed on the simulated data set and pedestrian track data recorded by monitor. The results show that the algorithm can achieve good clustering effect.

Keywords Moving object,Clustering,Isolation forest,Trajectory segmentation

1 引言

聚类是将一组实际的或抽象的相似对象划分为一类的过程[1],被广泛用于市场调查、模式识别、数据分析和图像处理等领域。聚类既能作为一个单独过程,用于寻找数据内在的分布结构,也可作为分类等其他学习任务的前驱过程[2]。代表性的聚类算法有k-means[3],BIRCH[4],DBSCAN[5],OPTICS[6]等。

GPS技术、物联网、卫星和无线通信技术的发展使得人们可以追踪并获取各类移动对象的轨迹,如飞机、车辆、行人和船只等。这些轨迹数据蕴藏着丰富的信息,需要一种有效的方法来进行分析。通过对这些数据进行聚类分析,可以获得很多有用的信息,如飞机飞行轨迹的聚类是空中交通管理领域的一个重要课题[7],通过对飞机飞行轨迹的聚类可以对机场设计、航线规划提供建设性的意见;通过对城市车辆轨迹的聚类来发现车流量大的交通路线,帮助交通管理部门改善市内交通状况;通过对行人轨迹进行聚类,预测行人的移动轨迹或发现其中的异常行为。

本文提出了一种新的轨迹聚类算法,该算法的主要过程如下:首先改进了基于最小描述长度原理(Minimum Description Length,MDL)的轨迹分段算法[8],将轨迹分段问题转换为求无向图的最短路径问题,然后使用Dijkstra算法求得最佳分段(即最优的轨迹分段方式),对待分类的所有轨迹进行分段。若轨迹不属于某一类别,则对于该类别而言,该轨迹相当于一条异常轨迹。本文参考基于独立森林的异常轨迹检测算法[9](isolation-Based Anomalous Trajectory detection,

到稿日期:2017-11-27 返修日期:2018-02-01 本文受国家自然科学基金(U1433116),南京航空航天大学研究生创新基地(实验室)开放基金(kfjj20171603)资助。

张怀峰(1994-),男,硕士生,主要研究方向为数据挖掘,E-mail:zhanghuaifeng@nuaa.edu.cn;**皮德常**(1971-),男,博士,教授,博士生导师,CCF会员,主要研究方向为数据挖掘、大数据管理与分析,E-mail:nuaacs@126.com(通信作者);**董玉兰**(1994-),女,硕士生,主要研究方向为数据挖掘。

iBAT),提出了基于独立森林的轨迹聚类算法(isolation-Based Trajectory Cluster,iBTC)。

本文第 2 节介绍相关研究工作;第 3 节对提出的 iBTC 轨迹聚类算法进行详细的讨论和分析;第 4 节通过模拟数据集和监控视频记录的行人轨迹数据集,将所提算法与几种现有的聚类算法进行对比实验分析,验证了 iBTC 算法的有效性;最后总结全文,并给出进一步的研究方案。

2 相关工作

现有的轨迹聚类算法主要有 3 类研究方向:1)提取轨迹的特征(如时空信息、速度、方向、加速度和其他特征)并找出轨迹的移动模式;2)找到合适的轨迹间的距离度量方法,从而可以有效地找到轨迹间的差异;3)设计高效的轨迹聚类算法,以更短的时间和更小的空间处理大量的轨迹数据。轨迹聚类算法的聚类依据可大致分为 3 类。

1)基于空间的轨迹聚类算法。空间特征是移动对象轨迹最基本的特征,基于空间特征的轨迹聚类非常直观,并且有助于发现移动对象的行为特征。Palma 等[10]提出 Eps 线性近邻距离函数来衡量轨迹间的差异,然后利用 DBSCAN 算法对轨迹进行聚类,以发现轨迹中的移动和停止行为。Mascuari[11]提出了基于空间划分的轨迹聚类算法,该算法根据移动对象的轨迹位置将空间划分为合适粒度的若干区域,从而使轨迹可以用符号进行表示。Hung 等[12]提出了名为轨迹线索聚合的轨迹聚类框架,用于发现用户最频繁的移动模式和能代表该移动模式的移动轨迹。Florian 等[13]提出了一种基于相对持久同源性的拓扑轨迹聚类法,该算法的时间复杂度为 O(n)。Gaffney 等[14]提出使用回归混合模型和最大期望的方法解决相似轨迹的聚类问题。Izakia 等[15]提出了一种基于粒子群优化算法的轨迹聚类算法,该算法可以自动聚类出最佳的类别数目。基于空间的轨迹聚类算法常用于与空间相关的分析中,如密集区域检测、热点区域挖掘等。

2)基于时间的轨迹聚类算法。轨迹通常由若干采样点组成,每个采样由该点坐标和采样时间等信息组成,因此时间特征对于移动对象的分析也非常重要。Nanni 等[16]提出了改进的 OPTICS 算法——T-OPTICS,该算法通过利用时间维度上的内在语义信息来提高轨迹聚类的质量。Mitsch 等[17]指出很多研究旨在寻找时间维度上的线性模式,而大量的周期性模式的情形并没有得到很好的处理。Yasodha 等[18]讨论了多种基于时间特征的轨迹相似性度量方法和基于时间特征的轨迹聚类算法,然后提出了一种度量聚类效果的方法。该类算法提供了一种非常有效的挖掘轨迹数据内在结构和压缩轨迹数据的方法。

3)基于划分-分组框架的轨迹聚类算法。该类算法通过轨迹分段算法先将轨迹划分为若干轨迹段,然后对这些轨迹段进行聚类,从而可以发现轨迹数据的公共子模式。Lee 等[8]提出了一个划分-分组的轨迹聚类框架,该方法首先利用 MDL 原理对轨迹进行分段,然后提出了 TRACLUS 算法对轨迹段进行聚类。为了简化轨迹分段过程,Yuan 等[19]提出了一种基于拐角检测的分段算法。Li 等[20]提出了一种增量式轨迹聚类算法,旨在减少计算新增轨迹时的时间开销和空间开销。该类轨迹聚类算法能够有效地处理长度较长、不同轨迹长度相差较大的轨迹数据,并且能够有效地发现轨迹的局部模式,但是这类算法受轨迹分段的影响较大。

综合上述算法的优点和不足,本文提出了基于独立森林的轨迹聚类算法,该算法属于基于空间的聚类算法,同时吸收了划分-分组框架中轨迹分段的思想。对本文启发较大的有文献[8]提出的轨迹分段、文献[9]提出的异常轨迹检测及 Liu 等[21]提出的独立森林。

3 基于独立森林的轨迹聚类算法

本节由 5 个小节组成,3.1 节介绍数据预处理;iBTC 算法借鉴了 iBAT 算法的思想,因此在介绍 iBTC 算法之前,3.2 节先对 iBAT 算法做简单介绍;3.3 节和 3.4 节分别对 iBTC 算法的细分和合并两个过程做介绍;3.5 节给出了 iBTC 算法的整体流程。iBTC 算法的 3 个主要步骤如下:

1)数据预处理。首先使用改进的 MDL 轨迹分段算法将轨迹数据分段,即将原始轨迹数据化简为由若干轨迹段组成的数据。

2)细分。将预处理后的数据细分为若干类别,其中每个类别自成一类,但是不能保证不同类别之间的轨迹不属于同一类。

3)合并。对于细分过程中得到的类别,将可能属于同一类的类别合并为一类,经过若干次细分和合并的迭代后得到最终的聚类结果。

3.1 数据预处理

首先定义不同轨迹段之间的距离,然后形式化地描述轨迹分段问题,最后提出改进的 MDL 轨迹分段算法对轨迹数据进行分段。

根据文献[8]中的定义,不同轨迹段之间的距离由垂直距离、平行距离和角度距离 3 部分组成,如图 1 所示,其中,$l_{\perp 1}$ 是点 s_j 到线段 L_i 的垂直距离,$l_{\perp 2}$ 是点 e_j 到线段 L_i 的垂直距离,$l_{\parallel 1}$ 是点 s_i 到点 s_j 的水平距离,$l_{\parallel 2}$ 是点 e_i 到点 e_j 的水平距离。

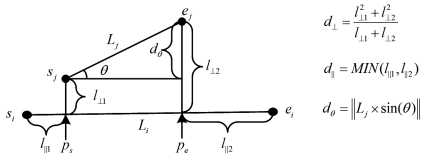


图 1 轨迹段之间的距离的 3 个组成部分

Fig. 1 Three components of distance for line segments

垂直距离为:

$$d_\perp(L_i, L_j) = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}} \tag{1}$$

平行距离为:

$$d_\parallel(L_i, L_j) = MIN(l_{\parallel 1}, l_{\parallel 2}) \tag{2}$$

角度距离为:

$$d_\theta(L_i, L_j) = \begin{cases} \|L_j\| \times \sin(\theta), & \text{if } 0° \leqslant \theta \leqslant 90° \\ \|L_j\|, & \text{if } 90° < \theta < 180° \end{cases} \tag{3}$$

角度距离的定义考虑了有方向的轨迹,当处理无方向的轨迹数据时,角度距离可直接定义为 $\|L_j\| \times \sin(\theta)$。

根据上述定义，得出两条轨迹段之间的距离如下：

$$dist(L_i,L_j)=w_\perp\cdot d_\perp(L_i,L_j)+w_\parallel\cdot d_\parallel(L_i,L_j)+w_\theta\cdot d_\theta(L_i,L_j) \quad (4)$$

其中，系数 $w_\perp$，$w_\parallel$，w_θ 通常取 1。

轨迹分段的形式化描述如下：将轨迹在某一点处变化很大的点称为特征点；从一条轨迹 $t_i=p_1p_2\cdots p_j\cdots p_{len_i}$ 中找出其特征点的集合，从而轨迹 t_i 被这些特征点划分为若干段，每两个连续特征点之间的线段称为轨迹段，从而轨迹 t_i 被划分为 par_i-1 个轨迹段，即 $\{p_{c_1}p_{c_2},p_{c_2}p_{c_3},\cdots,p_{c_{par_i-1}}p_{c_{par_i}}\}$。这个过程称为对轨迹的分段，简称轨迹分段。图 2 给出了轨迹分段的一个例子，其中原始轨迹 T_i 有 p_1-p_8 8 个点，经过分段后，可以由 p_1，p_4，p_6，p_8 这 4 个关键点表示。

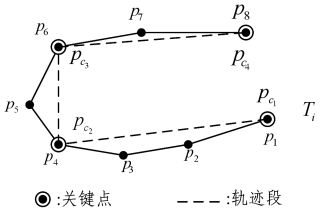


图 2　轨迹分段的一个示例

Fig. 2　Example of trajectory segementation

最优的轨迹分段应该具有两个性质：精确性和简洁性。精确性要求原始轨迹和轨迹分段之间的差异越小越好，简洁性要求划分出的轨迹段越少越好，精确性和简洁性是互相矛盾的。例如，在轨迹分段中，如果一条轨迹的所有点都被选择为特征点，即 $par_i=len_i$，则这种情况下精确性最高，但简洁性最低；反之，如果只有轨迹的起点和终点被选择为兴趣点，则这种情况下的简洁性最高，而精确性最低。

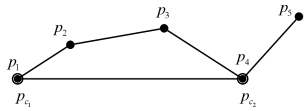
文献[8]提出使用 MDL 原理来寻找最佳的分段方案。MDL 原理包括两个部分：$L(H)$ 和 $L(D|H)$，H 是假设，D 是数据。$L(H)$ 用来描述假设的长度，单位为比特；$L(D|H)$ 是在假设为 H 的情况下数据 D 编码的长度，单位也是比特。当 $L(H)$ 与 $L(D|H)$ 的和最小时，H 是能够解释数据 D 的最佳假设。

在轨迹分段问题中，原始轨迹相当于 MDL 原理中的数据，对原始轨迹的分段相当于 MDL 原理中的假设 H，我们的目标是找到最佳的轨迹分段 H，使得 $L(H)$ 与 $L(D|H)$ 的和最小。

由文献[8]得到在轨迹分类问题中 $L(H)$ 和 $L(D|H)$ 的形式化描述，如图 3 所示。给定一条轨迹 $t_i=p_1p_2\cdots p_j\cdots p_{len_i}$ 和特征点的集合 $\{p_{c_1},p_{c_2},\cdots,p_{c_{par_i}}\}$ $(c_1<c_2<\cdots<c_{par_i})$，$L(H)$ 如式(5)所示，其中 $len(p_{c_j}p_{c_{j+1}})$ 是 $p_{c_j}p_{c_{j+1}}$ 轨迹段的长度，从而 $L(H)$ 表示在一个轨迹分段中所有轨迹段的长度之和。$L(D|H)$ 如式(6)所示，$L(D|H)$ 表示原始轨迹和轨迹分段之间的差异。

$$L(H)=\sum_{j=1}^{par_i-1}\log_2(len(p_{c_j}p_{c_{j+1}})) \quad (5)$$

$$L(D|H)=\sum_{j=1}^{par_i-1}\sum_{k=c_j}^{c_{j+1}-1}\{\log_2(d_\perp(p_{c_j}p_{c_{j+1}},p_kp_{k+1}))+\log2(d_\theta(p_{c_j}p_{c_{j+1}},p_kp_{k+1}))\} \quad (6)$$



$$L(H)=\log_2(len(p_1p_4))$$

$$L(D|H)=\log_2(d_\perp(p_1p_4,p_1p_2)+d_\perp(p_1p_4,p_2p_3)+d_\perp(p_1p_4,p_3p_4))+\log_2(d_\theta(p_1p_4,p_1p_2)+d_\theta(p_1p_4,p_2p_3)+d_\theta(p_1p_4,p_3p_4))$$

图 3　$L(H)$ 和 $L(D|H)$ 的形式化描述

Fig. 3　Formal specification of $L(H)$ and $L(D|H)$

根据以上描述，我们需要找到一个轨迹分段，使得 $L(H)+L(D|H)$ 最小，则该轨迹分段就是最佳分段。文献[8]提出了一种寻找最佳分段的近似算法，但是该算法只是找到了一个近似分段来代替最佳分段。此外，实际应用中对精确性和简洁性有不同的要求。譬如在轨迹数据量较小时，我们对轨迹进行分段时往往要求简洁性低一些而精确性高一些，使分段更能够表示数据的真实情况；而当数据量较大时，我们往往要求简洁性高一些而精确性低一些，简化数据，以减少数据处理时间。基于以上分析，我们提出了参数化轨迹分段策略，并对问题进行了建模，使其转化为无向图最短路径问题，并使用 Dijkstra 算法进行求解，最终得到最佳分段。

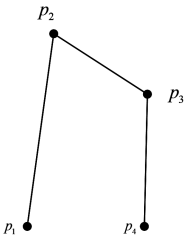
以 $uL(H)+vL(D|H)$ 代替 $L(H)+L(D|H)$，即找到一个分段，使得 $uL(H)+vL(D|H)$（$u,v\geqslant 0$，且 u 和 v 为实数）最小，则该分段就是最佳分段。当 $u/v>1$ 时，我们得到的最佳分段的简洁性会高一些，而精确性低一些，考虑极端情况，当 $u/v\to\infty$，即 $v=0$ 时，得到的轨迹分段即为原始轨迹起点和终点的连线，这种情况下的简洁性最高而精确性最低；当 $u/v<1$ 时，我们得到的最佳分段精确性会高一些，而简洁性低一些，考虑极端情况，当 $u/v=0$，即 $u=0$ 时，得到的轨迹分段即为原始轨迹，这种情况下的精确性最高而简洁性最低。根据上述分析，我们可以根据具体需要来调整 u 和 v 的大小，使轨迹分段的简洁性和精确性满足需求。另外，为了便于计算，对式(5)和式(6)进行了修改，使得 $L(H)$ 和 $L(D|H)$ 为非负数，如式(7)和式(8)所示：

$$L(H)=\sum_{j=1}^{par_i-1}\log_2(len(p_{c_j}p_{c_{j+1}})+1) \quad (7)$$

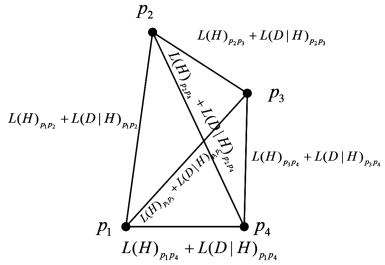
$$L(D|H)=\sum_{j=1}^{par_i-1}\sum_{k=c_j}^{c_{j+1}-1}\{\log_2(d_\perp(p_{c_j}p_{c_{j+1}},p_kp_{k+1})+1)+\log_2(d_\theta(p_{c_j}p_{c_{j+1}},p_kp_{k+1})+1)\} \quad (8)$$

考虑由轨迹 t_i 中所有点组成的一个无向完全图 G_i，其中边 p_ip_j 的权重 $w(p_ip_j)$ 由 $uL(H)+vL(D|H)$ 计算得出，则求最佳分段相当于求图 G_i 中以 p_1 为起点、以 p_{len_i} 为终点的最短路径，即该最短路径就是对该轨迹的最佳分段。因此，可以使用 Dijkstra 算法求得最佳分段。使用堆优化后，该算法的时间复杂度为 $O(n\log n)$[22]，其中 n 为图 G_i 中边的数量。图 4 给出了将最佳分段问题建模为最短路径问题的一个示例，其中，图 4(a)为原始轨迹，图 4(b)为该轨迹最佳分段问题对应的最短路径问题。图 4(a)为一条包含了 4 个点的轨迹，由于给定该轨迹任意的两个点，我们都可以求出这两点的 $L(H)$ 和 $L(D|H)$，因此对于图 4(b)中的无向完全图，每条边上的权重 $L(H)+L(D|H)$ 都可以求出。从而，根据 MDL 原理求出的最佳分段，也就是图 4(b)中无向完全图从点 p_1 到

点 p_4 的最短路径。



(a)原始轨迹



(b)转换为完全图最短路径问题,$u=v=1$

图 4 最佳分段问题以及对应的最短路径问题的示例

Fig. 4 Example of modeling optimal trajectory segmentation problem and corresponding shortest path problem

3.2 iBAT 算法

由于 iBTC 算法吸收了 iBAT 算法的主要思想,因此在介绍 iBTC 算法之前,首先介绍 iBAT 算法的主要思想。

iBAT 算法主要用于发现异常轨迹,该问题的形式化描述如下:

问题 1 记一条轨迹为 $t_i=p_{i1}\,p_{i2}\cdots p_{ik}$ $(k\geqslant 2)$,p_{ij} $(j\in[1,k])$是轨迹所经过的点。给定一个由若干条轨迹数据组成的集合:

$T=\{t_1,t_2,\cdots,t_n\}$,$n\in Z^+$

找出集合 $A=\{t_i\mid t_i$ 与 $T-\{t_i\}$中的元素差异很大 $\wedge\, t_i\in T\}$且 $|A|\ll n$。

问题 1 中的集合 A 就是所有异常轨迹组成的集合。考虑到异常轨迹有如下两条内在的性质:

1)异常轨迹的数量较少;

2)异常轨迹和大多数轨迹是不同的,具体来说,异常轨迹经过了不同的地点或以不同顺序经过了相同的地点。

iBAT 算法利用了独立森林的思想。独立森林是一种新的异常检测算法,该算法利用了异常对象“数量少且与众不同”的内在属性,使用随机树对所有对象进行划分,直到所有对象都与其他对象独立,得到一棵独立树(Isolation Tree,iTree),最终每个实例都是 iTree 的一个叶子结点。通过上述划分,异常对象通过较少的划分就可以与其他对象独立,而正常对象则需要较多划分才能与其他对象独立,即在 iTree 上表现为:异常对象所在的叶子结点的深度较小,正常对象所在的叶子结点的深度较大。因此,在 iTree 中,深度较小的叶子结点所对应的对象很有可能是异常对象[20]。

iBAT 算法对独立森林进行了改进,使其可以应用到轨迹数据上。算法的工作过程如下:记集合 $C_i=T-\{t_i\}$,首先从 T 中选取一条轨迹 t_i,称其为测试轨迹,然后随机选择 t_i 中的一个点 p_{ij},将所有不经过点 p_{ij} 的轨迹从集合 C_i 中移除,这个过程称为选点。重复上述过程,直到满足条件 1 或条件 2。

条件 1 集合 C_i 为空;

条件 2 集合 C_i 中的所有轨迹都包含 t_i 所经过的点。

通过该算法,异常轨迹在经过较少的选点后就可以与其他轨迹分离开(即满足条件 1 或条件 2),而正常轨迹则需要较多的选点才可以与其他轨迹分离开来,甚至无法与其他轨迹分离。由此可知,若一条轨迹在经过较少的选点后就和其他轨迹分离开,那么可以认定该轨迹为异常轨迹。

举例说明 iBAT 算法:图 5(a)包含了 9 条简化的轨迹;这些轨迹在图 5(b)中列出,其中 t_0 和 t_8 与其他轨迹明显不同,是异常轨迹;图 5(c)给出了 iBAT 算法的过程,每一步的集合中仅保留含有选择点的轨迹。可以发现,t_0 经过 1 次选点后就与其他轨迹分离开;t_8 经过 2 次选点后与其他轨迹分离开;而 t_3 在经过 4 次选点后仍然没有与其他轨迹分离开。

S	1	2	3	4	5
	6			7	8
	9				10
	11	12	13	14	D

(a)简化了的轨迹图像

t_0:1→6→9→11→12→13→14

t_1:1→2→3→4→5→8→10

t_2:1→2→3→4→8→10

t_3:1→2→3→4→7→8→10

t_4:1→2→3→4→7→10

t_5:1→2→3→4→5→8→10

t_6:1→2→3→7→8→10

t_7:1→2→3→7→10

t_8:1→2→3→4→5→8→7→4→5→8→7→4→5→8→10

(b)图 5(a)中包含的轨迹

$t_0:\{t_1,\cdots,t_8\}\xrightarrow{12}\varnothing$

$t_8:\{t_0,\cdots,t_7\}\xrightarrow{5}\{t_1,t_5\}\xrightarrow{7}\varnothing$

$t_3:\{t_0,t_1,t_2,t_4,\cdots,t_8\}\xrightarrow{3}\{t_1,t_2,t_4,\cdots,t_8\}\xrightarrow{2}\{t_1,t_2,t_4,\cdots,t_8\}\xrightarrow{7}\{t_4,t_6,t_7,t_8\}\xrightarrow{8}\{t_6,t_8\}$

(c)iBAT 算法过程示例

图 5 iBAT 算法示例

Fig. 5 Example of iBAT algorithm

3.3 iBTC 算法的细分过程

本文提出的 iBTC 算法用于解决轨迹聚类问题,该问题的形式化描述如下。

问题 2 记一条轨迹为 $t_i=p_{i1}\,p_{i2}\cdots p_{ik}$ $(k\geqslant 2)$,p_{ij} $(j\in[1,k])$是轨迹所经过的点。给定一个由若干条轨迹数据组成的集合:

$T=\{t_1,t_2,\cdots,t_n\}$,$n\in Z^+$

找出集合$\mathbb{C}=\{C_1,\cdots,C_m\}$,$1\leqslant m<n$,其中 C_i 是轨迹的集合,即聚类出的一个类别。$\forall\, i,j\in[1,m]$,$i\neq j$,$C_i\cap C_j=\varnothing$ 且 C_i 内部元素差异较小,C_i 中的元素与 $T-\{C_i\}$中的元素差异较大。

不妨假设 T 中有两个非空类别 C_1 和 C_2,则 C_1 中的轨迹

相对于 C_2 中的轨迹的差异与两个集合内部轨迹的差异中，必然是前者更大。若从 C_1 中取一个轨迹 t_c，则 t_c 相对于 C_2 中的轨迹必然满足“少且不同”这个条件，因此，可以认为 t_c 相对于 C_2 中的轨迹是一个异常轨迹。因此，若对 $\{t_c\}\cup C_2$ 应用 iBAT 算法，则 iBAT 算法可找出异常轨迹 t_c，即 t_c 在经过较少的选点后就与 C_2 中的轨迹分离开来。若对 $C_1\cup C_2$ 应用 iBAT 算法，不妨设选中的轨迹仍为 t_c，由上述分析可知，C_2 中的轨迹经过较少的选点后就与 t_c 分离开，而 C_1 中的轨迹需要经过更多次选点才能与 t_c 分离，甚至无法分离。推而广之，若 T 中有 $C_1,C_2,\cdots,C_m$ 个类别，应用 iBAT 算法选中的轨迹仍为 t_c，则 $C_2,\cdots,C_m$ 中的轨迹经过较少的选点后就与 t_c 分离开来，而 C_1 中的轨迹需要经过更多次选点才能与 t_c 分离，甚至无法分离。显然，我们可以认为，在经过一定次数的选点后仍然没有分离开的轨迹属于同一类，记为 C_i。这是因为与 C_i 中的轨迹差异较大的轨迹，在经过较少次选点后就与 C_i 中的轨迹分离开，所以在经过一定次数的选点后，C_i 中保留的是较为相似的轨迹。

对上述分析过程稍加改造即为 iBTC 算法的初步分类过程，该过程可以看作是将初始类别细分为若干子类别的过程，我们将该过程称为细分过程，伪代码如算法 1 所示。

算法 1　轨迹细分算法 partition(T,δ)

输入：轨迹数据集 T，选点次数阈值 δ

输出：细分结果$\mathbb{C}'=\{C_1',\cdots,C_w'\}$

```
1. 初始化：
   root_node=Create_node(T,0);
2. 随机选取一条轨迹 t_c∈root_node.cla;
3. Tree=Partition(root_node,δ,t_c);
4. Tree 的每个叶子结点中的 cla 字段的值即为𝕮′ 中的一个元素，即
   聚类出的一个类别；
5. return 𝕮′;
6. Partition(node,δ,t){
7.    初始化集合 C_acc,C_neg为空;
8.    if node.depth≥δ or node.cla 只有一条轨迹
9.       return;
10.   endif
11.   从 t 中随机选取一个点 p_c;
12.   for each t_i∈node.cla
13.      if p_c∈t_i
14.         将 t_i加到 C_acc中;
15.      else
16.         将 t_i 加到 C_neg中;
17.      endif
18.   endfor
19.   if C_acc非空
20.      depth=node.depth+1;
21.      right_node=Create_node(C_acc,depth);
22.   endif
23.   if C_neg非空
24.      left_node=Create_node(C_neg,0);
25.   endif
26.   Partition(right_node,δ,t);
27.   随机选取一条轨迹 t_lc∈left_node.cla;
28.   Partition(left_node,δ,t_lc);
29.   node.right=right_node;
30.   node.left=left_node;
31.   return;
32.}
33.Create_node(cla,depth){
34.   创建一个 node;
35.   node.cla=cla,
      node.depth=depth,
      node.left=null,
      node.right=null;
36.   return node;
37.}
```

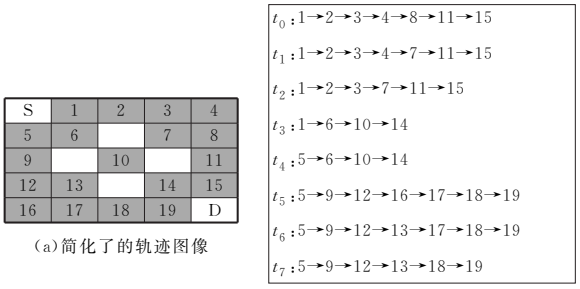
上述算法实现了对轨迹数据的初步细分，细分过程用一棵树来表述，该树的每个叶子结点就是一个类别。算法 1 中，第 33－36 行是创建结点的过程，每个结点包含 4 个字段，分别是该结点对应的轨迹数据的类别（*cla*）、该结点的深度（*depth*）（深度也就是选中某一轨迹后，在该轨迹上选点的次数，这与传统树的深度不同）、该结点的左孩子结点（*left*）和右孩子结点（*right*）。第 1 行初始化树的根结点 *root_node*，根结点的深度为 0，在根结点处原始轨迹数据集 T 被视为一类。第 3 行调用 Partition 过程构建树，即细分过程。第 4 行找到了树的所有叶子结点，也即找到了细分的结果。第 6－32 行是细分过程，该过程递归地构建了一个树状结构。第 7－10 行产生了一个叶子结点 *node*，当在某一条轨迹上的选点次数（*node.depth*）大于给定的选点次数阈值 δ 时，*node.cla* 中的轨迹仍未能分离开，则可以认为 *node.cla* 中的轨迹为一个类别，不需要再对 *node.cla* 进行细分，即对应树的一个叶子结点。第 11－25 行是对 *node.cla* 中的轨迹数据进行再次细分，与所选轨迹 t 有公共点 p_c 的所有轨迹被划分到 *right_node.cla* 中，反之，不包含 p_c 的所有轨迹被划分到 *left_node.cla* 中，可以说 *right_node.cla* 中包含的是与 t 相似的轨迹，*left_node.cla* 中包含的是与 t 不同的轨迹。第 26－28 行分别递归地对 *right_node.cla* 和 *left_node.cla* 再进行分类，由于 *right_node.cla* 包含的是与 t 相似的轨迹，因此对 *right_node.cla* 的再细分仍然选取轨迹 t 上的点，从而需将选点次数加 1，即：

$$right\_node.depth=node.depth+1$$

由于 *left_node.cla* 中包含的是与 t 不同的轨迹，不能再以 t 上的点进行细分，因此需要重新选取一条测试轨迹，从而需要将 *left_node.depth* 重新置为 0。对于选点次数的阈值 δ，细分出的类别的数量是 δ 的非递增函数，当 $\delta\to 0$ 时，细分出的类别的数量为 1，即细分出的类别为 T；当 $\delta\to\infty$ 时，由于通常任何轨迹之间并不会完全相同，因此细分出的类别会接近 $|T|$。图 6 给出了该算法的一个简单示例。其中 t_0-t_2，t_3-t_4 和 t_5-t_7 分别属于 3 个不同的类别。图 6(a)给出了简化后的轨迹图像；图 6(b)给出了图 6(a)中的 8 条轨迹；图 6(c)中，每个结点用一个三元组$\langle t_c,cla,depth\rangle$表示，其中 t_c 为选中的测试轨迹，父子结点之间连线上的数字是每次从 t_c 中选出的点的标号，$\delta=3$。

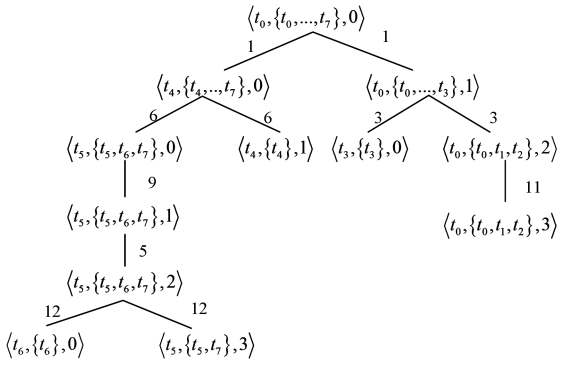
通过图 6(c)可以发现，树共有 5 个叶子结点，即算法将示例中的 8 条数据细分为 5 个类别。该算法将$\{t_0,t_1,t_2\}$类别正确分出，但是将$\{t_3,t_4\}$分成了两类，即$\{t_3\}$和$\{t_4\}$，将$\{t_5,t_6,t_7\}$分成了两类，即$\{t_6\}$和$\{t_5,t_7\}$。实际上，由于选点的随

机性，$\{t_0, t_1, t_2\}$也有可能会被划分成若干个类，这个问题将在 3.4 节解决。



(a)简化了的轨迹图像

(b)根据图 6(a)构造的 8 条数据



(c)初步分类算法的过程实例

图 6 轨迹细分算法的一个简单示例

Fig. 6 Example of trajectory partition algorithm

在解决算法 1 细分出的类别不准确的问题之前，需要先对该算法加以修改，使其可以用于分段后的轨迹数据。对于 3.1 节中分段后的轨迹，用 t_p 表示，则 $t_p = p_{c_1} p_{c_2} \cdots p_{c_{par_p}}$；以 s_i 表示轨迹段 $p_{c_i} p_{c_{i+1}}$，则 $t_p = s_1 s_2 \cdots s_{c_i} \cdots p_{c_{par_p-1}}$，将算法 1 中的第 11 行改为随机选择一个轨迹段 s_c，由于不同轨迹的轨迹段通常并不会重合，因此无法像判断点那样判断一条轨迹段是否属于某一条轨迹。因此我们采用近似的解决方案，即当两条轨迹段之间的距离小于某一阈值 ε 时，认为这两条轨迹段重合，即二者可以看作“同一”条轨迹段，将算法 1 中第 13 行的判断条件修改为“t_i 中存在与 s_c 的距离小于 ε 的轨迹段”即可。修改后的算法的伪代码如算法 2 所示。算法 1 与算法 2 仅第 11 行和第 13 行不同，其余全部相同。

算法 2 修改后的轨迹细分算法 iBTC_partition(T,δ,ε)

输入：分段后的轨迹数据集 T，选段次数阈值 δ，轨迹段之间的距离阈值 ε

输出：细分结果$\mathbb{C}' = \{C_1', \cdots, C_w'\}$

```
1. 初始化：
   root_node=Create_node(T,0);
2. 随机选取一条轨迹 t_c∈root_node.cla;
3. Tree=Partition(root_node,δ,t_c);
4. Tree 的每个叶子结点中的 cla 字段的值即为𝕮′ 中的一个元素，即一个类别;
5. return 𝕮′;
6. Partition(node,δ,t){
7.    初始化集合 C_acc,C_neg为空;
8.    if node.depth≥δ or node.cla 只有一条轨迹
9.       return;
10.   endif
11.   从 t 中随机选取一个轨迹段 s_c;
12.   for each t_i∈node.cla
13.      if t_i 中存在轨迹段与 s_c 的距离小于 ε
14.         将 t_i加到 C_acc中;
15.      else
16.         将 t_i 加到 C_neg中;
17.      endif
18.   endfor
19.   if C_acc非空
20.      depth=node.depth+1;
21.      right_node=Create_node(C_acc,depth);
22.   endif
23.   if C_neg非空
24.      left_node=Create_node(C_neg,0);
25.   endif
26.   Partition(right_node,δ,t);
27.   随机选取一条轨迹 t_lc∈left_node.cla;
28.   Partition(left_node,δ,t_lc);
29.   node.right=right_node;
30.   node.left=left_node;
31.   return;
32.}
33.Create_node(cla,depth){
34.   创建一个 node;
35.   node.cla=cla,
      node.depth=depth,
      node.left=null,
      node.right=null;
36.   return node;
37.}
```

考虑最坏情况，即所有轨迹自成一类（当然这几乎不可能，但是为了分析算法的时间复杂度，我们不妨这样假设），假设平均每条轨迹被分为 k 个轨迹段，所有轨迹分段后共有 n 条轨迹段，则每一条选中的轨迹段都要计算与其他轨迹的轨迹段的距离 δ 次。由于每个轨迹自成一类，因此每条轨迹都需要选取一条轨迹段计算其与所有不在该轨迹中的轨迹段的距离，则共需要计算约 $\delta \cdot (n^2/k)$ 次。考虑到 δ 和 k 比较接近，可以认为算法 2 的时间复杂度为 $O(n^2)$，在实际中每个轨迹自成一类的情况几乎不可能出现，因此算法的实际运行时间应该远小于 $O(n^2)$。

3.4 iBTC 算法的合并过程

将同一类别划分为若干不同类别的原因为：1)轨迹数据是序列化数据，同一类别的轨迹也并不是完全相同的，如图 6(b)中的 t_5 和 t_6 经过的点就不完全相同；2)选点是随机的，所以很有可能在选点次数到达 δ 之前就已经选出了同一类轨迹中不同的点，如图 6(c)中选出了点 13，导致$\{t_5, t_6, t_7\}$分成了两类，即$\{t_6\}$和$\{t_5, t_7\}$。因此，细分过程得到的类别具有如下性质。

性质 1 每个叶子结点中的轨迹数据是相似的，可以认为其属于同一类；

性质 2 不同的叶子结点中的轨迹数据也可能属于同一类。

基于性质 1 和性质 2，我们对细分过程得到的子类进行

合并，即将属于同一类但是被细分为不同类的类别合并为同一类，我们将该过程称为合并过程。该过程的伪代码描述如算法 3 所示。

算法 3　合并算法 iBTC_merge($\mathbb{C}'$,λ)

输入：前一次的细分结果$\mathbb{C}'$，迭代次数 λ

输出：最终分类结果$\mathbb{C}=\{C_1,\cdots,C_m\}$

1. for i=1 to λ
2.　初始化集合 T′为空；
3.　for each $C'_w\in\mathbb{C}'$
4.　　随机选取一条轨迹 $t_r\in C'_w$，将其加入到 T′中；
5.　　将 t_r 加上标记 w，即 t_r.label=w；
6.　endfor
7.　$\mathbb{C}''$=iBTC_partition(T′,δ,ε)；
8.　令集合$\mathbb{TC}'$为空；
9.　for each $C_j''\in\mathbb{C}''$
10.　　初始化集合 Tc 为空；
11.　　$\forall t_a\in C_j''$，将 $C'_{t_a.label}$中的元素加入到 Tc 中；
12.　　将 Tc 加入到集合$\mathbb{TC}'$中；
13.　endfor
14.　$\mathbb{C}'=\mathbb{TC}''$；
15. endfor
16. for each $C'_w\in\mathbb{C}'$
17.　if $|C'_w|<$MinLns
18.　　将 C'_w从$\mathbb{C}'$中移除，并标记 C'_w中的轨迹为异常轨迹；
19.　endif
20. endfor
21. $\mathbb{C}=\mathbb{C}'$；
22. return $\mathbb{C}$.

当 $i=1$ 时，算法 3 中，第 3－6 行根据前面的细分算法得出的细分结果$\mathbb{C}'$，从每一个类别 $C_i'\in\mathbb{C}'$中随机选出一条轨迹 t_r，并用 $t_r.label$ 记录 t_r 来自于$\mathbb{C}'$中的哪个类别。选取的所有轨迹组成了一个新的轨迹数据集 T'。第 7 行对 T'应用细分算法，得到新的细分类别的集合$\mathbb{C}''$。第 8－15 行，对于新的类别 $C_j''\in\mathbb{C}''$，其中的元素来自于$\mathbb{C}'$中的不同类别，若 $t_a,t_b\in C_j''$，即 t_a 和 t_b 被划分为同一类，换言之 t_a 和 t_b 较为相似，由 t_a 和 t_b 在$\mathbb{C}'$中的所在类别分别为 $C'_{t_a.label}$和 $C'_{t_b.label}$，即 t_a 与 $C'_{t_a.label}$中的元素较为相似，t_b 与 $C'_{t_b.label}$中的元素较为相似，可以推知 $C'_{t_a.label}$和 $C'_{t_b.label}$较为相似，从而 $C'_{t_a.label}$和 $C'_{t_b.label}$极有可能属于同一类别，因此将 $C'_{t_a.label}$和 $C'_{t_b.label}$合并为同一类。由于一次合并不一定能够满足要求，因此将上述过程迭代执行多次，λ 即为迭代执行的次数。第 16－20 行得到迭代执行 λ 次后的类别集合$\mathbb{C}'$，对于 $C'_w\in\mathbb{C}'$，若类别 C'_w中的元素过少，即$|C'_w|<MinLns$，则将其中的轨迹标记为异常轨迹并从$\mathbb{C}'$中移除，最终得到分类结果$\mathbb{C}$。图 7 给出了该算法的一个示例。

图 7 的输入数据为图 6(c)细分算法的细分结果，$\lambda=2$，$\delta=3$，最终合并后的聚类结果为$\mathbb{C}=\{\{t_0,t_1,t_2\},\{t_3,t_4\},\{t_5,t_6,t_7\}\}$。

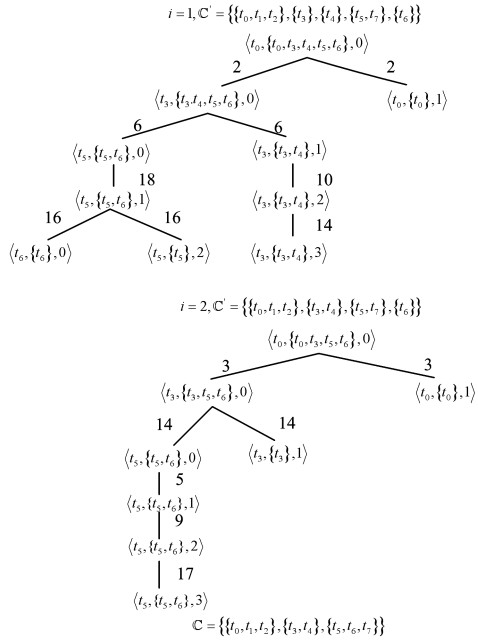


图 7　合并算法的示例

Fig. 7　Example of merging algorithm

3.5　iBTC 算法的整体流程

综合前文的算法，算法 4 给出 iBTC 算法的整体流程，算法的时间复杂度为 $O(\lambda\cdot n^2)$。

算法 4　基于独立森林的轨迹聚类算法 iBTC(T,δ,λ,ε)

输入：原始轨迹数据集 T，选点次数阈值 δ，迭代次数 λ，轨迹段之间的距离阈值 ε

输出：聚类结果$\mathbb{C}$

1. 使用 Dijkstra 算法对 T 中的轨迹进行分段，得到分段后的轨迹数据集 T_p；
2. $\mathbb{C}'$=iBTC_partition(T,δ,ε)；
3. $\mathbb{C}$=iBTC_merge($\mathbb{C}'$,λ).

4　实验结果与分析

4.1　模拟数据集上的实验

基于函数 $y=f(x)$，在函数值中加入高斯噪声得到模拟的轨迹数据，这些数据由 3 个函数产生：$y=120+4x$，$y=10+2x+0.1x^2$ 和 $y=250-0.75x$，共生成了 150 条轨迹数据，每条轨迹数据由 10 个点组成，每个函数产生 50 条轨迹，即共 3 个类别，每个类别包含 50 条轨迹。图 8 给出了这些模拟数据。

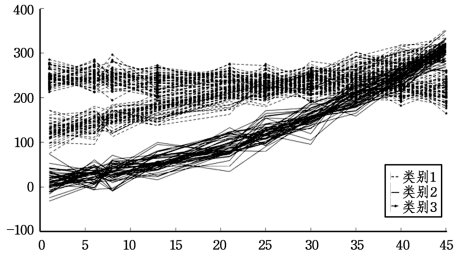


图 8　模拟轨迹数据

Fig. 8　Simulated trajectory data

实验软件为 Matlab R2016a，硬件为四核 Intel i5-6400

3.2GHz,4GB 内存,取参数 $\varepsilon=16.7$,$\delta=5$,$\lambda=40$。我们使用 Rand 指数[2](Rand Index,RI)来评价算法在模拟数据集上的聚类效果,RI 的值越大,说明聚类效果越好。重复运行算法 100 次,得到如图 9 所示的 RI-次数曲线。可以发现,算法聚类的结果与实际情况非常相符,但是由于算法内部有随机过程,因此 RI 的取值范围为 0.77～1,RI 的平均值为 0.98。

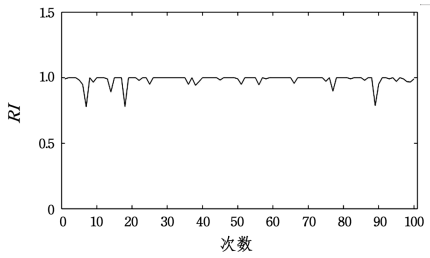


图 9 多次运行算法得到的 RI 曲线

Fig. 9 RI curves obtained by running algorithm with multiple times

4.2 聚类视频中的轨迹

采用爱丁堡信息论坛行人数据库中的数据(Edinburgh Informatics Forum Pedestrian Database)[24]。该数据库记录了监控探头拍摄的行人轨迹数据,我们取其中 200 条轨迹数据进行实验,包含 6919 个点,其图像如图 10 所示。

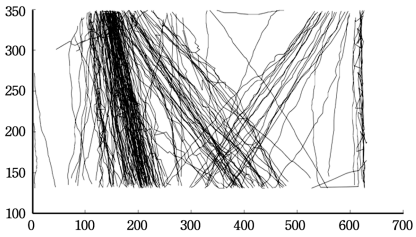


图 10 行人轨迹数据

Fig. 10 Pedestrian track data

对原始数使用 Dijkstra 算法进行轨迹分段,分段后的数据轨迹点数明显下降至 3835 个点。图 11 给出了分段后的轨迹图像。可以发现,分段后有效降低了数据量并且保留了原始数据的特征。

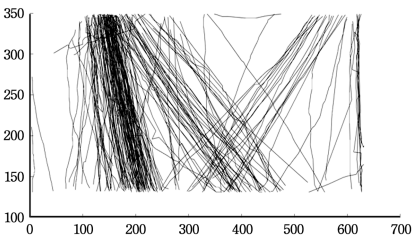


图 11 分段后的轨迹图像

Fig. 11 Trajectory after segmentation

将本文算法与基于 Fréchet 距离的 k-means 聚类算法、DBSCAN 聚类算法以及文献[15]提出的基于粒子群优化的轨迹聚类算法(Automated Clustering of Trajectory data using a Particle Swarm Optimization,ACTPSO)在行人轨迹数据集上进行实验对比,其中 k-means 聚类算法中计算类别中心的方法参考了文献[8]中提出的代表轨迹生成算法(Representative Trajectory Generation Algorithm)。表 1 列出了各种算法的时间复杂度,表 2 列出了各种算法在行人轨迹数据集上实验时的相关参数和运行时间,图 12 给出了 3 种算法的聚类效果。

表 1 各种算法的时间复杂度对比

Table 1 Time complexity of different algorithms

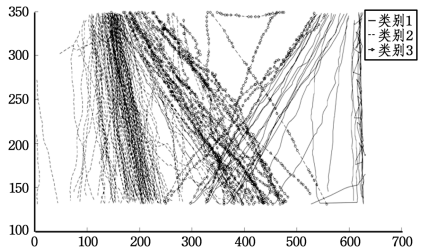
算法	时间复杂度
k-means	$O(n\cdot k\cdot t)$
DBSCAN	$O(n^2)$
ACTPSO	$O(n\cdot t\cdot p\cdot c_{\max})$
iBTC	$O(\lambda\cdot n^2)$

注:k 为类别数目,t 为迭代次数,p 为粒子数目,$c_{\max}$ 为最大类别数目

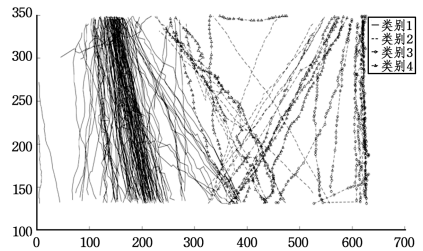
表 2 参数设置和算法运行时间

Table 2 Parameter setting and running time of algorithms

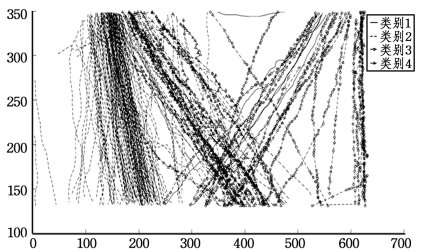
算法	参数	运行时间/s
k-means	$k=3$,$t=50$	47.23
DBSCAN	$MinPts=10$,$Eps=245.5$	32.08
ACTPSO	$p=30$,$v_{\min}=-4$,$v_{\max}=4$ $c_1=c_2=2$,$c_{\max}=15$,$m=2$	470.32
iBTC	$\delta=29$,$\lambda=45$,$\varepsilon=70$	11.89



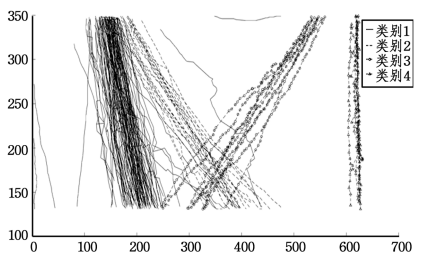
(a)k-means 聚类结果



(b)DBSCAN 聚类结果



(c)ACTPSO 聚类结果



(d)iBTC 聚类结果

图 12 各种算法的聚类结果

Fig. 12 Clustering results of different algorithms

由图 12 可以看出，iBTC 算法的聚类效果明显优于 k-means，DBSCAN 和 ACTPSO，即类内相较于其他两个算法更加紧致，而类间相较于其他算法更加松散。

结束语　本文主要做了以下两方面的工作：1）基于 MDL 原理，在文献[8]的基础上提出了参数化轨迹分段策略，通过更改参数使分段结果满足实际应用中对精确性和简洁性的要求。对轨迹分段问题进行建模，使其转化为求无向最短路径问题，使用 Dijkstra 算法进行求解，得到最佳分段。2）提出了基于独立森林的轨迹聚类算法，该算法从同一类别内的轨迹之间差异较小而不同类别的轨迹之间差异较大的角度出发，首先对原始轨迹进行分段，然后对分段后的轨迹数据通过迭代细分-合并过程，聚集差异较小的轨迹，分离差异较大的轨迹，从而达到聚类的目的；并且通过在模拟数据集上的实验和行人轨迹数据上的实验验证了算法的有效性。在后续的工作中，我们将进一步研究细分和合并过程中参数对聚类效果的影响，以获得更好的聚类效果。

参 考 文 献

[1] ACHTERT E，KRIEGEL H P，ZIMEK A. Deriving quantitative models for correlation clusters[C]// ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM，2006：4-13.

[2] 周志华. 机器学习[M]. 北京：清华大学出版社，2016.

[3] LLOYD S. Least squares quantization in PCM[J]. IEEE Transactions on Information Theory，1982，28(2)：129-137.

[4] ZHANG T，RAMAKRISHNAN R，LIVNY M. BIRCH：An Effcient Data Clustering Method for Very Large Databases[C]// ACM SIGMOD Int'l Conference on Management of Data. ACM，1996：103-114.

[5] ESTER M，KRIEGEL H P，XU X. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise[C]// International Conference on Knowledge Discovery and Data Mining. AAAI Press，1996：226-231.

[6] WANG W，YANG J，MUNTZ R R. STING：A Statistical Information Grid Approach to Spatial Data Mining[C]// International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc，1997：186-195.

[7] NICOL F，PUECHMOREL S. Trajectories clustering：a minimum entropy approach[C]// The Second International Conference on Big Data，Small Data. Linked Data and Open Data，2016：35-41.

[8] LEE J G，HAN J，WHANG K Y. Trajectory clustering：a partition-and-group framework[C]// ACM SIGMOD International Conference on Management of Data. ACM，2007：593-604.

[9] ZHANG D，LI N，ZHOU Z H，et al. iBAT：detecting anomalous taxi trajectories from GPS traces[C]// International Conference on Ubiquitous Computing. ACM，2011：99-108.

[10] PALMA A T，BOGORNY V，KUIJPERS B，et al. A clustering-based approach for discovering interesting places in trajectories[C]// ACM Symposium on Applied Computing. DBLP，2008：863-868.

[11] MASCIARI E. A Framework for Trajectory Clustering[C]// International Conference on Geosensor Networks. Springer-Verlag，2009：102-111.

[12] HUNG C C，PENG W C，LEE W C. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes[J]. Vldb Journal，2015，24(2)：169-192.

[13] FLORIAN T，POKORNY F T，GOLDBERG K，et al. Topological trajectory clustering with relative persistent homology[C]// IEEE International Conference on Robotics and Automation. IEEE，2016：16-23.

[14] GAFFNEY S，SMYTH P. Trajectory clustering with mixtures of regression models[C]// ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM，1999：63-72.

[15] IZAKIAN Z，MESGARI M S，ABRAHAM A. Automated clustering of trajectory data using a particle swarm optimization[J]. Computers Environment & Urban Systems，2016，55：55-65.

[16] NANNI M，PEDRESCHI D. Time-focused clustering of trajectories of moving objects[J]. Journal of Intelligent Information Systems，2006，27(3)：267-289.

[17] MITSCH S，MÜLLER A，RETSCHITZEGGER W，et al. A Survey on Clustering Techniques for Situation Awareness[M]// Web Technologies and Applications. Springer Berlin Heidelberg，2013：815-826.

[18] YASODHA M，PONMUTHURAMALINGAM P. A survey on temporal data clustering[J]. International Journal of Advanced Research in Computer and Communication Engineering，2016，1(9)：768-772.

[19] YUAN G，XIA S，ZHANG L，et al. An efficient trajectory-clustering algorithm based on an index tree[J]. Transactions of the Institute of Measurement & Control，2012，34(7)：850-861.

[20] LI Z，JAEGIL L，LI X，et al. Incremental Clustering for Trajectories[C]// International Conference on Database Systems for Advanced Applications. Springer-Verlag，2010：32-46.

[21] LIU F T，KAI M T，ZHOU Z H. Isolation Forest[C]// Eighth IEEE International Conference on Data Mining. IEEE Computer Society，2008：413-422.

[22] CHEN M，CHOWDHURYR A，RAMACHANDRAN V，et al. Priority Queues and Dijkstra's Algorithm[R]. USA：University of Texas，2007：1-25.

[23] FISHER B. Edinburgh informatics forum pedestrian database[OL]. http://homepages. inf. ed. ac. uk/rbf/FORUMTRACKING.