高技术通讯 2010 年 第 20 卷 第 1 期:99 ~ 105

doi:10.3772/j.issn.1002-0470.2010.01.017

一种快速移动对象轨道聚类算法①

陶运信② 皮德常

(南京航空航天大学信息科学与技术学院 南京 210016)

摘 要 针对已有轨道聚类(TRACLUS)算法的线段聚类模块需要对划分后的每条线段进行邻域查询的问题,将取样技术引入轨道聚类,提出一种快速移动对象轨道聚类(FTCS)算法。FTCS 算法根据基于极大连通子图的合并原理,对核心线段的 *Eps* 邻域以及与该 *Eps* 邻域相重叠的所有轨道聚类进行合并,避免了 TRACLUS 算法中核心线段 *Eps* 邻域内线段的不必要邻域查询操作。在真实和合成轨道数据集上的大量实验结果表明,FTCS 算法显著降低了邻域查询操作次数,在保持 TRACLUS 算法轨道聚类质量的同时,成倍提高了轨道聚类的时间效率。

关键词 数据挖掘,聚类,轨道,邻域,密度

0 引 言

聚类是一个将数据库中相似数据划分到一组以便提供数据分布模式概要的过程[1]。随着海量移动对象轨道数据被收集、存储在数据库中,迫切需要对这些数据进行有效分析。一个典型的数据分析任务是寻找以相同方式运动的对象[2],聚类技术可以满足这项任务的需要,这使得移动对象轨道聚类这一研究方向应运而生。移动对象轨道聚类在台风登陆预测、交通堵塞预报和动物移动分析等领域具有广阔应用前景。

目前,轨道数据分析的研究大多将轨道视为一个整体,没有考虑子轨道。例如,1999 年,Gaffney 等人[3]最早对轨道聚类做出研究,聚类的基本单元是整条轨道。2000 年,Knorr 等人[4]在离群点检测的一个实例研究中,将基于距离的离群点检测算法用于检测轨道离群,但是离群检测的基本单元也是整条轨道。2006 年,Nanni 等人[5]提出一种移动对象轨道数据聚类算法,该算法试图将以全局相似方式运动的移动对象划分到一组,还是针对整条轨道。不幸的是,将轨道作为一个整体进行聚类并不能发现轨道的相似部分。

2007 年,Lee 等人[6]设计了一个轨道聚类的划分和分组框架,基于该框架提出一种轨道聚类(trajectory clustering,TRACLUS)算法,用于发现公共子轨道。由于 TRACLUS 的线段聚类模块需要对划分后的每条线段进行邻域查询,这直接导致整个轨道聚类算法的时间复杂度为 $O(n^2)$。但是为了提高轨道聚类算法在大型数据库上的可伸缩性,迫切需要提高算法的时间效率。

由于 TRACLUS 中线段聚类模块的基本思想来源于著名的基于密度的从带有噪声的空间数据库中发现聚类(density-based spatial clustering of applications with noise,DBSCAN)算法[7],本文先概述有关对 DBSCAN 算法时间效率进行改进的研究,并分析这些改进思想无法扩展到轨道聚类的线段聚类模块中。相关改进研究[8,9]通过选择核心对象邻域中种子对象或参考点,以避免对每个对象都进行邻域查询,这对于二维或多维空间中有规则的对象比较有效。但是,在轨道聚类中,一条核心线段邻域中线段的分布并不像二维或多维空间中有规则的对象那样分布在一个圆形或(超)球形区域内,因此很难找到一种有效的方法来选择种子线段,使得一条核心线段邻域中的线段能被位于该邻域中的种子线段的邻域覆盖,以降低邻域查询操作次数。为此,本文提出了一种快速移动对象轨道聚类(fast moving object trajectory clustering,FTCS)算法。该算法能显著降低邻域查询操作的次数,提高轨道聚类时间效率。

① 863 计划(2007AA01Z404)资助项目。
② 男,1985 年生,硕士;研究方向:移动对象数据挖掘;联系人,E-mail: taoycs@yahoo.cn
(收稿日期:2009-01-16)

1 轨道聚类框架

目前,通过轨道聚类来发现公共子轨道问题主要基于划分和分组框架[6],本文对其进行扩展,由 3 步组成,如图 1 所示。给定一个轨道数据集 $TR = \{TR_1, \cdots, TR_{num_{tra}}\}$,首先,近似轨道划分算法使用最小描述长度(minimum description length, MDL)原理,将轨道划分问题转换为 MDL 优化,使每条轨道被划分成一个轨道划分(划分后的线段)的集合;其次,对轨道划分的集合,线段聚类算法根据线段之间的相似性度量对其进行聚类,输出一个轨道聚类集合 $C = \{C_1, \cdots, C_{num_{clus}}\}$,这是轨道聚类问题中很关键的一步;最后,代表轨道生成算法通过沿与聚类方向向量相垂直的方向扫描一条直线,为每个轨道聚类 C_i 生成一条代表轨道——公共子轨道。

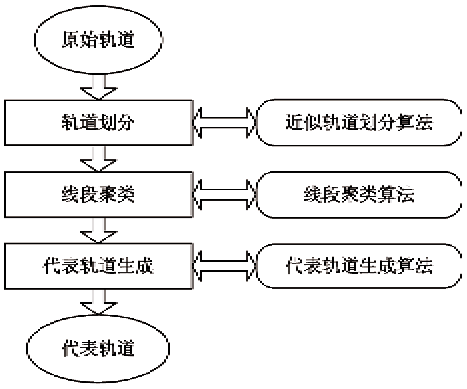


图 1 轨道聚类框架

2 快速轨道聚类算法 FTCS

2.1 相关概念

Chen 等人[10]提出了一种改进的线段 Hausdorff 距离,用于 Logo 识别。Lee 等人[6, 11, 12]对该距离进行改进,定义轨道聚类中线段之间的距离函数,共包括垂直距离、水平距离和角距离这三个要素。为了让读者理解后续内容,本节将对该距离函数做简单介绍。

假设有两条 d 维($d = 2$)有向线段 $L_i = s_i e_i$ 和 $L_j = s_j e_j$,线段方向根据轨道数据采样点的时间先后来确定,其中 s_i, e_i, s_j, e_j 代表 d 维点,如图 2 所示。不妨设 L_i 比 L_j 长,s_j 和 e_j 在 L_i 上的投影分别为 p_s 和 p_e。

定义 1 假设 $l_{\perp 1}$ 为 s_j 和 p_s 之间的欧氏距离,$l_{\perp 2}$ 为 e_j 和 p_e 之间的欧氏距离,则线段 L_i 与 L_j 之间的垂直距离定义为

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

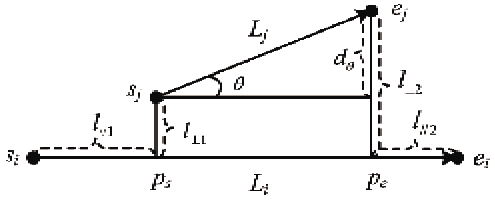


图 2 线段之间的距离函数解析($d = 2$)

定义 2 线段 L_i 与 L_j 之间的平行距离定义为

$$d_{//}(L_i, L_j) = \min(l_{//1}, l_{//2}) \tag{2}$$

其中:$l_{//1} = \min(\mathrm{dist}(p_s, s_i), \mathrm{dist}(p_s, e_i))$,$l_{//2} = \min(\mathrm{dist}(p_e, s_i), \mathrm{dist}(p_e, e_i))$。

定义 3 线段 L_i 与 L_j 之间的角距离定义为

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

其中:θ($0° \leqslant \theta \leqslant 180°$)取有向线段 L_i 与 L_j 之间的夹角。

定义 4 设 L 和 S 是两条线段,L 和 S 之间的距离函数 $\mathrm{dist}(L, S)$ 由 3 个分量组成:

$$\mathrm{dist}(L, S) = \omega_{\perp} \cdot d_{\perp}(L, S) + \omega_{//} \cdot d_{//}(L, S) + \omega_{\theta} \cdot d_{\theta}(L, S) \tag{4}$$

其中权值 $\omega_{\perp}$、$\omega_{//}$ 和 ω_{θ} 由应用背景来决定。默认情况下,将这 3 个权值均设为 1。

定义 5 线段 L 的 Eps 邻域定义为 $N_{Eps}(L) = \{S \in D \mid \mathrm{dist}(L, S) \leqslant Eps\}$。

如果一条线段的 Eps 邻域足够稠密,例如它包含至少 $MinLns$ 条线段,那么该条线段为核心线段。如果一条线段本身不是核心线段,并且也不包含在其它核心线段的 Eps 邻域中,那么该线段为边界线段。轨道划分集合中剩下的既不是核心线段,又不是边界线段的那些轨道划分称为离群线段。

定义 6 轨道聚类 C_i 的划分轨道(participating trajectories)集合定义为:$\mathrm{PTR}(C_i) = \{\mathrm{TR}(L_j) \mid \forall L_j \in C_i\}$,其中,$\mathrm{TR}(L_j)$表示 C_i 中线段 L_j 由哪条轨道被划分得到,$|\mathrm{PTR}(C_i)|$称为 C_i 的轨道势。

2.2 FTCS 算法思想

与 TRACLUS 类似,快速轨道聚类算法 FTCS 亦包括第 1 节描述的轨道划分、线段聚类和代表轨道

生成这3个计算模块,其中线段聚类模块为本文研究的重点,下面主要讲述线段聚类模块。FTCS使用基于取样的快速线段聚类算法FTCS_LS,FTCS_LS的基本思想是:一个轨道聚类可以被视为最少的核心线段和它们的*Eps*邻域组成,为了发现一个轨道聚类,仅需对那些代表性的核心线段进行*Eps*邻域查询。例如,图3所示为一个轨道划分的集合,它包含一个由浅灰色轨道划分表示的轨道聚类,并用粗的虚拟轨道表示该轨道聚类的代表轨道,该轨道聚类大致可以视为由虚线圈表示的*Eps*邻域组成(作为一个解析性例子,*Eps*邻域的数目并不一定是最少的)。

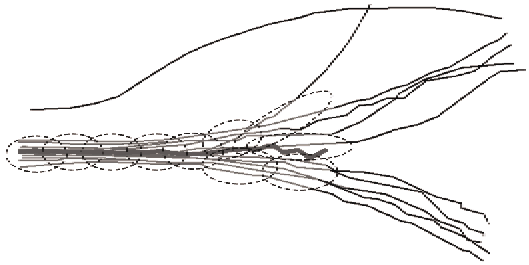


图3 FTCS_LS的基本思想解析

不幸的是,我们不可能在扫描线段集合D之前确定这些核心线段,再加上线段的*Eps*邻域也不像二维或多维空间中有规则对象的*Eps*邻域那样具有对称性,因此很难找到一种有效的方法来根据当前核心线段的*Eps*邻域确定种子线段。为此,FTCS_LS引入了如下基于取样的解决方法:顺序选择首次发现的未标号线段,然后对其进行*Eps*邻域查询,最后判断该线段是否为核心线段,如果是核心线段,那么按照基于极大连通子图的合并原理,将该*Eps*邻域以及与该*Eps*邻域相重叠的所有轨道聚类进行合并。尽管通过上述取样方法选择的核心线段并不一定是相对整个轨道划分集合全局意义上最少的核心线段,但是该方法仍可以显著地减少*Eps*邻域查询操作的次数,实验结果也表明该方法是可行的、有效的。

基于取样的解决方法中*Eps*邻域的合并是根据下述基于极大连通子图的合并原理来进行。如果两条核心线段CL_1和CL_2之间的距离小于等于$2\times Eps$,则CL_1和CL_2是邻接核心线段。这样我们就可以用无向图来描述算法执行过程中取样的核心线段,图的顶点是核心线段,邻接核心线段之间有一条边,处于同一个极大连通子图中的核心线段组成一类,该类中核心线段的*Eps*邻域就组成了一个轨道聚类。FTCS_LS把求上述无向图的极大连通子图融入到发现轨道聚类过程中。另外,由于线段之间的距离函数不满足三角不等式,会存在两个*Eps*邻域之间存在重叠而它们的核心线段之间距离却大于$2\times Eps$的情况,这会导致轨道聚类的分裂。例如,一个*Eps*邻域合并的实例如图4所示,核心线段$CL_1,\cdots,CL_n$的*Eps*邻域组成了轨道聚类C_i,L_i同时位于核心线段CL_1和CL_j的*Eps*邻域中,尽管$\mathrm{dist}(CL_1,L_i)<Eps$,$\mathrm{dist}(CL_j,L_i)<Eps$,而$\mathrm{dist}(CL_1,CL_j)>\mathrm{dist}(CL_1,L_i)+\mathrm{dist}(CL_j,L_i)>2\times Eps$(线段之间的距离函数违背三角不等式),则$CL_j$的*Eps*邻域会作为一个单独的聚类而未合并到$C_i$中。所以当一条核心线段与一个轨道聚类中所有*Eps*邻域的核心线段之间距离都大于$2\times Eps$时,再判断该核心线段的*Eps*邻域是否与轨道聚类中某一条核心线段的*Eps*邻域存在重叠。

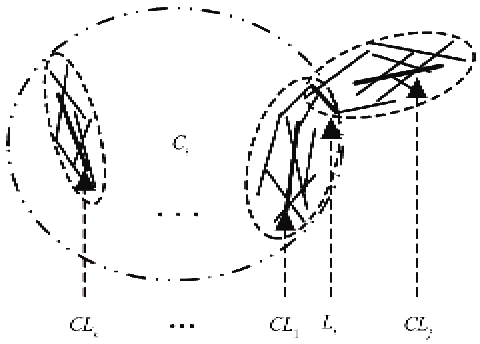


图4 *Eps*邻域合并的实例

2.3 FTCS_LS算法描述

FTCS_LS的描述如算法1所示,该算法由两步组成。与传统的基于密度的聚类算法如DBSCAN[7]、CURD[9]和DBRS[13]等不同的是,并不是所有密度连接的集合都会成为轨道聚类,FTCS_LS算法通过第二步检查每个轨道聚类的轨道势来排除轨道势低于给定阈值的轨道聚类,这与"公共"子轨道这一目的相吻合。

算法1 基于取样的快速线段聚类算法FTCS_LS

输入:轨道划分后的线段集合D,参数*Eps*和*MinLns*

输出:轨道聚类

01. 将线段聚类列表LSClusterList置为空;

//第一步:发现轨道聚类

02. WHILE(! D.IsAllClassified()) DO

```
03.  顺序扫描 D,从中选择首次发现的未标
     号线段 L;
04.  EpsSeeds = EpsNeighborhood(D, L, Eps);
     //对 L 进行 Eps 邻域查询
05.  IF |EpsSeeds| < MinLns THEN
06.    L.SetClusterId(OUTLIER);
       //L 暂时视为离群线段或边界线段
07.    continue;
08.  END IF
09.  EpsSeeds.SetClusterId(CLASSIFIED);
     //将 EpsSeeds 以及已发现的与
     //EpsSeeds 相重叠的所有聚类进行合并
10.  bFirstMerge 置为 TRUE, newCi 置为空;
11.  Ci = LSClusterList.GetFirstCluster();
12.  WHILE (! Ci.IsEmpty()) DO
13.      IF Intersection(L, EpsSeeds, Ci)
         THEN//若 Eps 邻域与 Ci 存在重叠
14.        IF bFirstMerge THEN
15.          newCi = Merge(Ci, EpsSeeds);
             //Ci 与 EpsSeeds 合并,结果保
             存到 newCi
16.          将核心线段 L 和 Ci 对应的核心
             线段表加入到 newCi 对应的核
             心线段表;
17.          LSClusterList.Replace(Ci, new-
             Ci);   //用 newCi 代替 LSClus-
             terList 中的 Ci
18.          bFirstMerge = FALSE;
19.        ELSE
20.          Add(newCi, Ci);
             //将 Ci 添加到 newCi
21.          LSClusterList.DeleteCluster(Ci);
             //从 LSClusterList 中删除 Ci
22.          将 Ci 对应的核心线段表加入到
             newCi 对应的核心线段表;
23.        END IF
24.    END IF //IF Intersection (EpsSeeds, Ci)
25.    Ci = LSClusterList.GetNextCluster();
26.  END WHILE //WHILE (! Ci.IsEmpty())
27.  IF bFirstMerge THEN
     //若 EpsSeeds 与已发现的聚类均不重叠
28.    LSClusterList.AddCluster(EpsSeeds);
       //EpsSeeds 作为新聚类加入聚类列表
29.    将核心线段 L 加入到聚类 EpsSeeds 对
       应的核心线段表;
30.  END IF
31.  END WHILE //WHILE (! D.IsAllClassified())
32.  将线段聚类列表 LSClusterList 转换为 C 中
     轨道聚类;
     //第二步:检查每个轨道聚类的轨道势
33.  FOR each trajectory cluster Ci in C DO
34.    IF (|PTR(Ci)| < thTrajCard) THEN
       //thTrajCard 为轨道势的阈值
35.      从轨道聚类集合中移除 Ci;
36.  END IF
37.  END FOR
```

2.4 时间复杂性分析

FTCS算法共包括轨道划分、线段聚类和代表轨道生成3个计算模块,由于线段聚类是本文研究的重点,再加上轨道划分和代表轨道生成的时间复杂度相对于线段聚类可以忽略不计[6],所以复杂性分析主要针对线段聚类。类似地,若没有特别指明,时间效率评价中所有算法也是主要考虑线段聚类的时间效率。

邻域查询操作是FTCS_LS算法中最费时的操作,由于没有合适的索引结构支持,对某条线段进行一次邻域查询就需要扫描 D 中每条线段一次,该操作时间复杂度为 $O(n)$。而FTCS_LS引入了取样机制选择核心线段,通过基于极大连通子图的合并原理对核心线段的 Eps 邻域以及与该 Eps 邻域相重叠的所有轨道聚类进行合并,避免了位于核心线段的 Eps 邻域内线段的邻域查询操作,这是成倍提高算法性能的本质所在。FTCS_LS能避免不必要的邻域查询操作的次数取决于参数的选择和轨道的具体分布,因此FTCS_LS需要进行的邻域查询次数 m 为一个不确定的值,但是满足不等式 $m \leqslant n$ 且绝大多数情况下 $m \ll n$。在没有索引结构支持下,FTCS的时间复杂度为 $O(m \cdot n)$,而TRACLUS的时间复杂度为 $O(n^2)$,所以FTCS比TRACLUS快 n/m 倍。

3 实验结果与分析

本节为FTCS算法的实验评价,选择的对比算法为TRACLUS。实验中,算法在VC6.0下用C++实现,所有实验是在CPU为Pentium4 3.0G、主存为512M、硬盘为80G 7200r/min和操作系统为Windows XP Professional的机器上进行。实验评价中所用到的数据集如表1所示,Test是一个合成轨道数据集,

表 1　轨道聚类实验数据集

数据集的名称	Test	Best Track	Deer95	Elk93
原始轨道数目	26	570	32	33
原始点的数目	1635	17736	20065	47204
轨道划分数目	457	1873	2007	3852

Best Track、Deer95 和 Elk93 既是轨道聚类算法 TRACLUS[6]中使用的测试数据集,也是轨道离群检测[11]和轨道分类[12]中用到的测试数据集,所以对比实验选用了这 3 个真实数据集。

3.1　轨道聚类质量比较

使用基于引力概念的聚类质量评估方法[14],来定量比较两个算法的轨道聚类质量,对比结果见表 2。该评估方法通过分析轨道聚类结果中线段之间的引力关系来评估轨道聚类质量,质量越高的聚类结果,其评估值 F_{aggl}/F_{noise} 就越大。

表 2　FTCS 和 TRACLUS 的轨道聚类质量比较

算法	Test (*Eps* = 24, *MinLns* = 6)		Best Track (*Eps* = 24, *MinLns* = 6)		Deer95 (*Eps* = 22, *MinLns* = 6)		Elk93 (*Eps* = 22, *MinLns* = 6)	
	聚类个数	F_{aggl}/F_{noise}	聚类个数	F_{aggl}/F_{noise}	聚类个数	F_{aggl}/F_{noise}	聚类个数	F_{aggl}/F_{noise}
TRACLUS	2	8.765	7	58.013	2	7.401	10	53.660
FTCS	2	8.493	5	58.881	2	11.184	9	64.516

实验结果表明,在离群线段所占比例很小的合成轨道数据集 Test 上, FTCS 的聚类质量比 TRACLUS 低,但是在 3 个真实轨道数据集上,FTCS 的聚类质量比 TRACLUS 都要高。这是因为,FTCS 的基本思想是:一个轨道聚类可以被视为最少的核心线段和它们的 *Eps* 邻域组成,以达到用尽量少的核心线段和这些核心线段的 *Eps* 邻域来覆盖轨道聚类。FTCS 使用基于取样的解决方法并根据基于极大连通子图的合并原理来进行 *Eps* 邻域合并,这对位于聚类中心的线段被划分到哪个类影响较小,FTCS 和 TRACLUS 轨道聚类结果的区别主要在于位于聚类边缘的线段。随着搜索粒度的变大,FTCS 不对位于聚类边缘核心线段 *Eps* 邻域内每条线段进行判断,所以会在聚类边缘比 TRACLUS 多包含一些离群线段,即 TRACLUS 中位于聚类边缘的真正离群线段比 FTCS 多。根据上述聚类质量评估方法,这些被 FTCS 聚类结果多包含的离群线段会导致 TRACLUS 离群线段所受到的引力 F_{noise} 较大,从而引起在真实数据集上 TRACLUS 的评估值 F_{aggl}/F_{noise} 反而比 FTCS 低。因此在实际应用中,应根据待聚类数据集的规模、聚类精确性要求来选择相应的轨道聚类算法,以取得最佳聚类效果或时间效率。

3.2　影响效率的邻域查询操作次数评价

Eps 邻域查询操作是 FTCS 算法和 TRACLUS 算法的线段聚类模块中最费时的操作,该操作的次数是决定整个轨道聚类算法运行时间的关键因素,所以在进行时间效率评价之前本节先对两个算法的 *Eps* 邻域查询操作次数进行比较。

首先,通过实验对 FTCS 和 TRACLUS 在 3 个真实轨道数据集上的 *Eps* 邻域查询操作次数进行比较,影响算法 *Eps* 邻域查询操作次数的参数 *Eps* 和 *MinLns* 根据文献[6]中描述的启发方法来选择。FTCS 和 TRACLUS 的参数设置相同,在 Best Track 上的参数设置为 *Eps* = 25 和 *MinLns* = 6,在 Deer95 上的参数设置为 *Eps* = 22 和 *MinLns* = 8,在 Elk93 上的参数设置为 *Eps* = 22 和 *MinLns* = 6。实验结果如图 5 所示。从该图可以看出,FTCS 算法较大程度地减少了 TRACLUS 所需的 *Eps* 邻域查询操作次数。

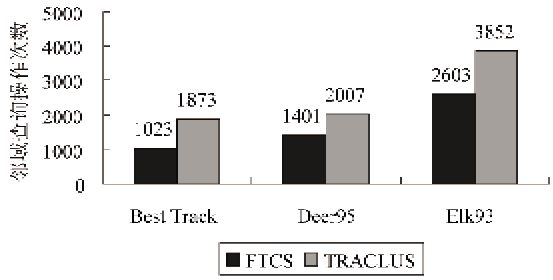
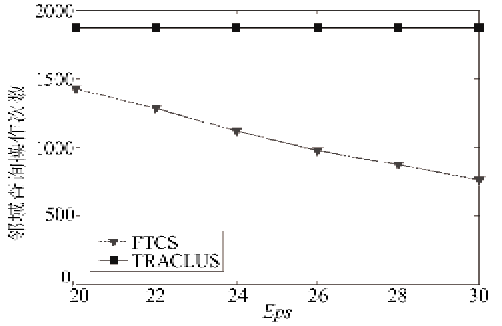
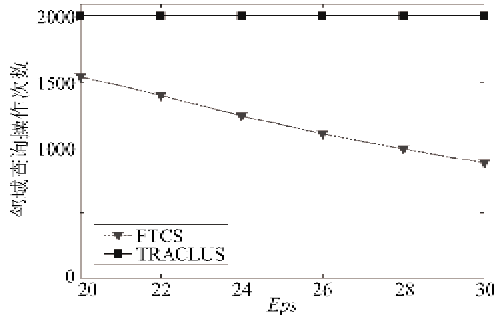


图 5　FTCS 和 TRACLUS 的 *Eps* 邻域查询操作次数比较

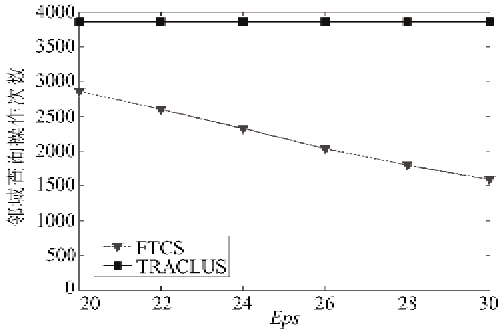
然后,通过实验研究了随着 *Eps* 取值的变化,FTCS 和 TRACLUS 在 3 个真实轨道数据集上的 *Eps* 邻域查询操作次数相应的变化情况,*MinLns* 保持为 6 不变,实验结果如图 6 所示。对于 TRACLUS 算法,由于它需要对每条线段进行邻域查询,所以不论 *Eps* 取值如何变化,*Eps* 邻域查询操作次数始终保持不变。但是对于 FTCS 算法,随着 *Eps* 取值的增大,*Eps* 邻域查询操作次数线性减少,可伸缩性比 TRACLUS 好。



(a) Best Track



(b) Deer95



(c) Elk93

图6 *Eps* 邻域查询操作次数与 *Eps* 取值的变化关系

3.3 时间效率比较

实验使用3个真实数据集,对TRACLUS、NBTC和FTCS的运行时间进行比较。对于3个数据集,TRACLUS和FTCS输入参数分别设置如下:Best Track上设置为:*Eps* = 25和*MinLns* = 6,Deer95上设置为:*Eps* = 22和*MinLns* = 8,Elk93上设置为:*Eps* = 22和*MinLns* = 6。实验结果如图7所示,该图表明

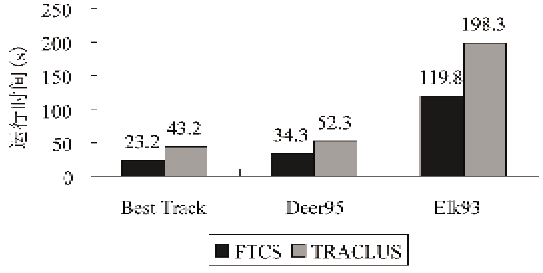


图7 FTCS和TRACLUS的运行时间比较

FTCS算法的运行时间明显少于TRACLUS,速度为TRACLUS的约2倍。

4 结 论

移动对象轨道数据聚类分析是一个新兴研究方向。由于已有的轨道聚类算法效率不高,因此面对低粒度的大型轨道数据集,算法通常不能有效工作。本文在分析TRACLUS算法不足的基础上,提出一种快速移动对象轨道聚类算法FTCS。该算法能显著地降低邻域查询操作的次数,提高轨道聚类速度。使用基于极大连通子图的合并原理,对核心线段的*Eps*邻域以及与该*Eps*邻域相重叠的所有已发现轨道聚类进行合并,FTCS避免了对核心线段*Eps*邻域中的线段进行不必要邻域查询操作,从而降低轨道聚类时间和I/O开销。分别用合成轨道数据集和真实轨道数据集对快速算法FTCS的性能进行测试,结果表明FTCS在保持轨道聚类质量的同时提高了轨道聚类的时间效率。

参考文献

[1] Xu R, Wunsch D. Survey of clustering algorithms. *IEEE Trans on Neural Networks*, 2005, 16(3): 645-678

[2] Vlachos M, Kollios G, Gunopulos D. Discovering similar multidimensional trajectories. In: Proceedings of the 18th International Conference on Data Engineering. San Jose: IEEE Computer Society, 2002. 673-684

[3] Gaffney S, Smyth P. Trajectory clustering with mixtures of regression models. In: Proceedings of the 5th ACM SIGKDD International Conference on Knowledge and Data Mining. San Diego: ACM Press, 1999. 63-72

[4] Knorr E M, Ng R T, Tucakov V. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 2000, 8(3-4): 237-253

[5] Nanni M, Pedreschi D. Time-focused clustering of trajectories of moving objects. *Journal of Intelligent Information Systems*, 2006, 27(3): 267-289

[6] Lee J G, Han J W, Whang K Y. Trajectory clustering: a partition-and-group framework. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. Beijing: ACM Press, 2007. 593-604

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

[8] Zhou S G, Zhou A Y, Jin W, et al. FDBSCAN: a fast DBSCAN algorithm. *Journal of Software*, 2000, 11(6): 735-744

[9] 马帅, 王腾蛟, 唐世渭等. 一种基于参考点和密度的快速聚类算法. 软件学报, 2003, 14(6): 1089-1095

[10] Chen J Y, Leung M K H, Gao Y S. Noisy logo recognition using line segment Hausdorff distance. *Pattern Recognition*, 2003, 36(4): 943-955

[11] Lee J G, Han J W, Li X L. Trajectory outlier detection: a partition-and-detect framework. In: Proceedings of the 24th International Conference on Data Engineering. Cancun: IEEE Computer Society, 2008. 140-149

[12] Lee J G, Han J W, Li X L, et al. TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering. In: Proceedings of the 34th International Conference on Very Large Data Bases. Auckland: ACM Press, 2008. 1081-1094

[13] Wang X, Hamilton H J. DBRS: a density-based spatial clustering method with random sampling. In: Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining. Seoul: Springer Verlag, 2003. 563-575

[14] 于勇前, 赵相国, 陈衡岳等. 基于引力概念的聚类质量评估算法. 东北大学学报(自然科学版), 2007, 28(8): 1109-1112

A fast moving objects trajectory clustering algorithm

Tao Yunxin, Pi Dechang

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

Abstract

Considering that the existing trajectory clustering (TRACLUS) algorithm needs neighborhood query for each line segment after partition, the paper introduces a sampling technique into trajectory clustering and proposes a fast moving objects trajectory clustering (FTCS) algorithm. The FTCS algorithm merges the *Eps*-neighborhood of core line segments with trajectory clusters that intersect with those *Eps*-neighborhoods according to the merging principle based on maximum connected subgraph, so it avoids the TRACLUS algorithm's unnecessary neighborhood query of line segments that lie in *Eps*-neighborhood of core line segments. The experimental results on real and synthetic trajectory data demonstrate that the FTCS algorithm reduces the number of neighborhood query remarkably and improves the efficiency of trajectory clustering while keeps the quality of trajectory clustering.

Key words: data mining, clustering, trajectory, neighborhood, density