# 一种基于密度的空间数据流在线聚类算法

于彦伟 [1]　王 沁 [1]　邝 俊 [1]　何 杰 [1]

**摘　要**　为了解决空间数据流中任意形状簇的聚类问题, 提出了一种基于密度的空间数据流在线聚类算法 (On-line density-based clustering algorithm for spatial data stream, OLDStream), 该算法在先前聚类结果上聚类增量空间数据, 仅对新增空间点及其满足核心点条件的邻域数据做局部聚类更新, 降低聚类更新的时间复杂度, 实现对空间数据流的在线聚类. OLDStream 算法具有快速处理大规模空间数据流、实时获取全局任意形状的聚类簇结果、对数据流的输入顺序不敏感、并能发现孤立点数据等优势. 在真实数据和合成数据上的综合实验验证了算法的聚类效果、高效率性和较高的可伸缩性, 同时实验结果的统计分析显示仅有 4% 的空间点消耗最坏运行时间, 对每个空间点的平均聚类时间约为 0.033 ms.

**关键词**　空间数据挖掘, 聚类数据流, 基于密度的聚类, 在线算法, 噪声处理

**引用格式**　于彦伟, 王沁, 邝俊, 何杰. 一种基于密度的空间数据流在线聚类算法. 自动化学报, 2012, **38**(6): 1051−1058

**DOI**　10.3724/SP.J.1004.2012.01051

## An On-line Density-based Clustering Algorithm for Spatial Data Stream

YU Yan-Wei[1]　WANG Qin[1]　KUANG Jun[1]　HE Jie[1]

**Abstract**　We propose an efficient online density-based clustering algorithm (On-line density-based clustering algorithm for spatial data stream, OLDStream), which is designed for online discovering clusters in spatial data stream. In OLDStream, only the new spatial point and its adjunct points which satisfy core point are processed in clustering update. And the overall clusters results can be accessed instantaneously. The developed algorithm has exhibited many advantages such as its high scalability to online process incremental large-scale spatial data, its capability to discover overall clusters with arbitrary shape instantaneously, its insensitivity to the input sequence of data stream, and its capability to detect all isolated points. An experimental evaluation of the effectiveness, efficiency and scalability of our algorithm was performed by using real data and large synthetic data from Matlab and Thomas Brinkhoff′s network-based generator. Experimental results vividly demonstrated that our algorithm can fast and efficiently cluster new points based on the previous points. The statistics of the results showed that only 4% of the points take the worst case running time, and the average running time is about 0.033 ms for each point process.

**Key words**　Spatial data mining, clustering data stream, density-based clustering, on-line algorithm, handling noise

**Citation**　Yu Yan-Wei, Wang Qin, Kuang Jun, He Jie. An on-line density-based clustering algorithm for spatial data stream. *Acta Automatica Sinica*, 2012, **38**(6): 1051−1058

近年来, 交通导航系统、实时跟踪系统、电子监控系统、基于位置的服务等实时应用的需求越来越大, 在这些应用系统中, 移动目标的实时空间坐标数据一般以快速数据流的形式被接收, 海量递增的空间数据流处理成为了应用的瓶颈[1]. 数据流聚类方法是一种广泛应用于数据流中相关空间知识的在线处理和实时获取的数据处理方法[2]. 基于密度的聚类分析算法可以发现任意形状的聚类, 这对于空间数据分析具有非常重要的作用. 在空间数据流挖掘领域, 基于密度的聚类分析既可以作为一个独立的工具来获得移动目标点实时的空间分布等知识, 也可以作为空间数据流的预处理步骤, 其他挖掘算法在其聚类结果上做进一步的分析[3]. 然而, 对基于空间数据流的实时应用系统而言, 一个必须解决的关键问题是如何对空间数据流进行在线聚类分析. 例如, 在基于车辆 GPS 收集的城市交通监控系统中, 要想获取实时城市交通状态, 需要在线处理几十万辆车的 GPS 数据流.

与传统静态数据相比, 空间数据流中数据变化速度较大, 对算法响应速度要求很高; 而且系统一般以线性扫描接收数据, 要求所有数据尽量处理一次. 因此, 空间数据流在线聚类算法需要具有较高的可伸缩性, 可以处理大规模增量空间数据; 能够发现数据流中存在的任意形状的聚类簇; 对数据流的输入顺序不敏感; 具备较低的时间空间复杂度, 能够处理高速数据流; 并且能够处理数据流中噪声. 而这些需求给空间数据流聚类算法设计带来了巨大挑战.

基于密度的聚类方法, 如 DBSCAN (Density-based spatial clustering of applications with noise)[4]、GDBSCAN (Generalized density-based spatial clustering of applications with noise)[5]、DENCLUE (Density-based clustering)[6] 以及之后一些改进的算法[7−9], 虽然在静态空间数据库中能够获取到较好的任意形状聚类, 却无法适用于空间数据流在线聚类. Ester 等[10] 提出一种用于空间仓库挖掘的增量聚类算法, 该算法在 DBSCAN 的基础上设计增量或减量数据聚类, 详细分析了数据增量或减量时对现有聚类结果的影响, 比较适用于有大规模数据更新的数据仓库挖掘应用. Aggarwal 等[11] 提出了一种双层架构的数据流聚类方法 CluStream, 该双层框架将数据流聚类分为在线微聚类和离线宏聚类两个层次, 前者负责在线统计数据流特征向量, 得出概要数据信息, 后者负责相应用户请求, 根据概要数据生成聚类结果. CluStream 同时使用金字塔时间框架, 可以得到不同粒度的聚类结果. 这种双层的微聚类框架在之后的很多数据流聚类算法中被采用和改进, 如 ACluStream[12]、DenStream[13]、SDStream[14] 和 C-DenStream[15]. 与 K-means[16] 等基于划分的聚类算法相似, CluStream 算法使用特征统计值生成子聚类, 对类球形簇能得到质量较好的聚类效果, 但却很难聚类任意形状的簇. DenStream 算法在 CluStream 双层架构上引入了核心微聚类簇、潜在微聚类簇和孤立点微聚类簇等概念以获取较准确的聚类结果, 但同样需要统计输入数据带时间特性的特征向量和大量密度计算, 其时间复杂度也比较高. SDStream 算法在 DenStream 算法的基础上采用滑动窗口来处理数据流, 只能获取到最近数据流中任意形状的聚类簇. C-DenStream 算法是一个带约束的密度聚类算法, 需要一定的背景知识去指导聚类并将约束反映到微聚类上. 实时数据流聚类 D-Stream[17] 是一种不同于微聚类簇框架的聚类方法, 该算法将输入数据空间划分为若

收稿日期 2011-10-13　录用日期 2012-03-01
Manuscript received October 13, 2011; accepted March 1, 2012
国家高技术研究发展计划 (863 计划) (2011AA040101), 国家自然科学基金 (61172049, 61003251), 教育部博士点基金 (20100006110015) 资助
Supported by National High Technology Research and Development Program of China (863 Program) (2011AA040101), National Natural Science Foundation of China (61172049, 61003251), and Doctoral Fund of Ministry of Education (20100006110015)
本文责任编委 周杰
Recommended by Associate Editor ZHOU Jie
1. 北京科技大学计算机与通信工程学院 北京 100083
1. School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083

干网格, 接收到的空间数据被映射到网格中, 通过设定密度阈值将网格分为密集、稀疏、记忆过渡期 3 类, 每读入一个空间点, 更新该点所在网格单元的密度. 当有聚类查询时, 将相邻的密集网格合并形成网格聚类簇. D-Stream 算法使用网格映射以空间数据点数目表示网格密度, 避免了大量的距离计算, 减少了时间复杂度, 却丢失了数据的空间位置信息, 聚类效果明显受到网格粒度的影响. 为了提高聚类的质量, Tu 等[18] 又在 D-Stream 算法基础上引入了网格间吸引力的概念, 用于解决网格内空间信息的丢失问题, 但是算法在计算网格间吸引力时要进行大量计算, 增加了算法的计算复杂度.

本文为了满足实时空间数据流聚类的应用需求, 提出了一种高效的基于密度的在线聚类算法 OLDStream (On-line density based clustering algorithm for spatial data stream) 该算法采用了 DBSCAN 算法的密度概念, 将输入数据流看作一个整体, 当新空间数据点到达时, 对已有聚类结果做快速的修正处理, 实现对空间数据流的在线聚类处理. 聚类过程中仅对新空间点及其邻域数据做聚类更新, 降低算法时间复杂度, 以此提高在线聚类的执行效率. 同时采用聚类簇核心点列表索引结构和空间 *Eps*-网格存储空间数据点, 当新的数据点到达时, 通过 *Eps*-网格快速找到可能需要更新聚类的邻域点, 并通过核心列表索引迅速完成簇扩展、簇合并等聚类更新操作.

## 1 在线聚类相关定义

### 1.1 基本概念

本文算法在 DBSCAN[4] 的基础上对一些相关术语做出了定义. 以下定义涉及到两个参数: 空间点邻域范围半径 *Eps* (Epsilon neighborhood) 和最小邻居点阈值 $MinPts$ (Minimum number of points).

*Eps*-邻域: 给定一个空间数据点 $p$, 以点 $p$ 为中心, *Eps* 为半径的空间区域范围称为点 $p$ 的 *Eps*-邻域.

*Eps*-邻居点: 在空间数据点 $p$ 的 *Eps*-邻域内的所有空间点称为点 $p$ 的 *Eps*-邻居点, 标示为 $Neighbor_{Eps}(p)$. 其数量标示为 $|Neighbor_{Eps}(p)|$.

*Eps*-网格: 对于空间数据点集合 $D$, 按照给定的 *Eps* 为边长划分数据空间, 每一个划分空间称为一个 *Eps*-网格, 设网格区域中心坐标为 $(x_1, x_2, \cdots, x_d)$, 则该 *Eps*-网格被标示为 $G_{(t_1,t_2,\cdots,t_d)}$, 其中 $t_1 = \lfloor x_1/Eps \rfloor$, $t_2 = \lfloor x_2/Eps \rfloor, \cdots, t_d = \lfloor x_d/Eps \rfloor$. $d$ 为空间维度, $(t_1, t_2, \cdots, t_d)$ 被记为网格的关键字.

核心点: 如果一个空间点 $p$ 的 *Eps*-邻居点数量不少于 $MinPts$, 则点 $p$ 是某个簇的核心点.

边界点: 如果一个空间点 $q$ 的 *Eps*-邻居点数量少于 $MinPts$, 但存在核心点 $p$ 满足 $p \in Neighbor_{Eps}(q)$, 则点 $q$ 是空间点 $p$ 所在簇的边界点.

孤立点: 如果空间数据点 $q$ 的 *Eps*-邻居点数量少于 $MinPts$, 且其任意的 *Eps*-邻居点 $p$ 都不是核心点, 则点 $q$ 被认为是空间数据点集合的孤立点或者噪声.

直接密度可达[4]: 如果一个空间点 $p$ 是核心点 $q$ 的 *Eps*-邻居点, 则称点 $p$ 直接密度可达于点 $q$.

由于空间点 $p$ 不一定是核心点, 因此, 点 $q$ 不一定直接密度可达于点 $p$, 所以直接密度可达是非对称的.

密度可达[4]: 给定一系列的空间点 $p_1, p_2, \cdots, p_n$, 并且 $p_i$ 直接密度可达于 $p_{i+1}$ $(i = 1, 2, \cdots, n-1)$, 若 $p = p_1$, $q = p_n$, 则称空间点 $p$ 密度可达于空间点 $q$.

很明显, 密度可达也是非对称的, 但密度可达具有传递性. 即空间点 $o$ 密度可达于点 $p$, 空间点 $p$ 密度可达于点 $q$, 则空间点 $o$ 密度可达于点 $q$. 另外, 由密度可达的定义可得出: 对于一对密度可达的空间点, 至少有一个是核心点, 且另一空间点密度可达于该核心点. 也就是说, 对于一个空间点来说, 不可能密度可达于一个边界点.

密度相连[4]: 如果空间点 $p$, $q$ 同时密度可达空间点 $o$, 则称点 $p$ 密度相连点 $q$. 由定义可知, 密度相连是对称的.

聚类簇: 将空间数据点集合 $D$ 中所有密度相连的空间点聚集成多个子集合, 每一个非空子集合称为一个聚类簇 $C$. 对每个簇 $C$ 来说, 都可表示由其中的一个核心点及密度可达于该核心点的所有空间点构成.

**推论 1.** 若一个核心点 $p$ 分别属于两个簇 $C_1$ 和 $C_2$, 则 $C_1$ 和 $C_2$ 可密度相连为一个簇 $C$.

**证明.** 设定当前数据集合为 $D$, 由于核心点 $p \in C_1$, 由聚类簇的定义可知, 任意的属于簇 $C_1$ 的空间点 $o$ 都密度相连于点 $p$, 又由于点 $p$ 为簇 $C_1$ 的核心点, 密度相连于点 $p$ 的空间点都密度可达于核心点 $p$, 所以, 任意的空间点 $o$ $(o \in C_1)$ 都密度可达于点 $p$. 同理, 对于任意空间点 $o'$ $(o' \in C_2)$ 也都密度可达于点 $p$. 综上所述, 任意的两个空间点 $o$, $o'$ $(o \in C_1, o' \in C_2)$ 都密度相连, 所以 $C_1$ 和 $C_2$ 可密度相连为一个簇. 因此, 若核心点 $p$ 分别属于两个或两个以上的聚类簇, 则可以合并这些簇为一个簇. 由此也得出核心点最终仅属于一个簇. □

**推论 2.** 边界点至少属于一个簇, 而孤立点不属于任何簇.

**证明.** 推论 2 说明边界点可以同时存在于多个簇中, 而孤立点远离高密度簇. 如图 1 所示的二维数据点分布, 其中, 给定 *Eps* 如圆圈半径和 $MinPts = 4$, 则 $q_1, q_2$ 是核心点, 点 $p$ 是边界点, 并且 $q_1, q_2 \in Neighbor_{Eps}(p)$. 由聚类簇的定义可知, 存在簇 $C_1, C_2$ 满足 $q_1 \in C_1, q_2 \in C_2$. 且由于点 $p$ 直接密度可达于 $q_1, q_2$, 所以点 $p \in C_1$ 同时点 $p \in C_2$. 由图可看出 $distance(q_1, q_2) > Eps$, 并且 $q_1$ 与 $q_2$ 互不密度可达, 所以 $C_1$, $C_2$ 是两个相互独立的簇, 因此, 边界点 $p$ 同时属于两个簇. 而孤立点 $o$ 的所有 *Eps*-邻居点都不是核心点, 即使存在其 *Eps*-邻居点作为边界点被聚类到密度簇中, 如点 $p$. 但由于点 $o$ 不可能密度可达到任何边界点, 孤立点 $o$ 也不可能被聚集到任何簇. □

### 1.2 在线聚类描述

针对空间数据流的在线聚类算法希望最终获取到密度聚集的空间数据点组合. 然而数据点的密度分布是相对于全局数据, 在线聚类算法要在已有数据聚类基础上快速处理增量数据并及时反映到全局数据上. 由于前期处理是在局部数据上的操作, 随着增量数据的急剧增加, 需要不断修正前期处理结果以获取正确的全局聚类结果. 同时为了快速高效地处理呈数据流接收的空间数据, 其修正操作应尽可能少.

在线聚类描述: 对于给定的参数 *Eps* 和 $MinPts$, 当前已获得的聚类簇集合为 $\{C_1, C_2, \cdots, C_m\}$, 在线聚类的过程被描述为: 当一个新的空间数据点 $P$ 被接收时, 如果点 $P$ 满足核心点条件, 则仅需将簇 $C$ ($C$ 为包含任意一个点 $P$ 的满足核心点条件的 *Eps*-邻居点的聚类簇, 若不存在这样的簇, 新建聚类簇 $C$) 更新为 $C = \{o | o \in D_{\text{now}}$ 且点 $o$ 密度可达于点 $P\}$; 否则, 对于所有的点 $q \in Neighbor_{Eps}(P)$, 若点 $q$ 是核心点, 则将聚类簇 $C(q)$ 更新为 $C(q) = \{o | o \in D_{\text{now}}$ 且点 $o$ 密度可达于点 $q\}$. $C(q)$ 表示包含点 $q$ 的簇, $D_{\text{now}}$ 表示当

前数据集合.

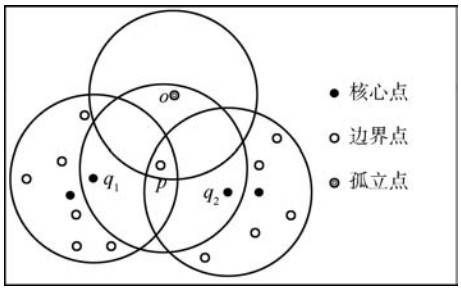



图 1　数据集合 $D$ 分布示例

Fig. 1　An example of points set

上述描述说明当新空间数据点 $P$ 到达后, 仅点 $P$ 或点 $P$ 满足核心点要求的 $Eps$-邻居点的聚类结果被更新, 这是因为一个新空间数据点的到来仅可能影响到本地区域的簇结构, 对远离其 $Eps$-邻域的聚类簇不会产生影响. 若新点 $P$ 本身为孤立点, 不会对本地簇产生任何影响, 因而不做任何的处理.

## 2　OLDStream 算法

### 2.1　算法思想

OLDStream 是解决在线聚类描述问题的一种全局在线的算法实现, 目的是对空间数据流进行基于密度的在线聚类, 在数据流结束时即可获取任意形状的聚类结果. 但是, 数据流何时结束并不知道, 因此必须时刻保证聚类结果能够表征当前数据集合的密度分布.

当一个新空间点 $P$ 到达时, 新点的加入仅会影响部分本地数据点的 $Eps$-邻居点, 因此, 算法仅针对点 $P$ 和其 $Eps$-邻居点做聚类处理. 算法在执行过程中, 每个空间点维护它的 $Eps$-邻居点; 算法维护一个簇结构, 每个簇保存其簇内的核心点, 构成 $Seeds\_list$. 算法的索引结构如图 2 所示, 簇的 $Seeds\_list$ 通过指针索引到数据集合中的核心点, 数据点的 $Near\_list$ 同样通过指针索引连接到其邻居点.

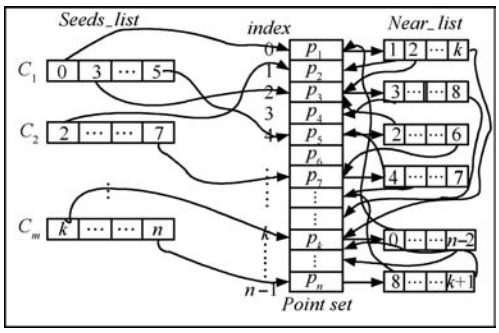



图 2　算法索引结构

Fig. 2　Index structure of OLDStream

OLDStream 算法的思想是: 当一个新数据点 $P$ 到达时, 首先, 通过邻域参数 $Eps$ 和最小邻域点 $MinPts$, 更新数据点的 $Near\_list$; 仅对满足核心点要求的点 $P$ 和它的 $Eps$-邻居点进行聚类处理. 处理的过程就是这些点所在簇的扩展、合并, 或者新建包括这些核心点的簇.

### 2.2　算法描述

OLDStream 算法过程分成三步:

**步骤 1.** 获取新空间点 $P$ 的 $Eps$-邻居点.

为了获取新数据点 $P$ 及其 $Eps$-邻居点, 本文把数据空间划分为若干 $Eps$-网格, 分布在 $Eps$-网格区域中的数据点被存放在网格内, 算法维护一个 $Eps$-网格列表. 查找点 $P$ 的 $Eps$-邻居点的方法如下:

首先, 根据点 $P$ 的位置信息, 获取其所在的 $Eps$-网格 $G_{(t_1,t_2,\cdots,t_d)}$ 的关键字, 根据该 $Eps$-网格的关键字 $(t_1,t_2,\cdots,t_d)$, 判断其邻近 $Eps$-网格是否存在, 若存在, 邻域 $Eps$-网格区域内的点均为点 $P$ 的候选 $Eps$-邻居点. 以二维空间为例, 在图 3 中, 点 $P$ 的候选 $Eps$-邻居点区域范围为 $G_1 \bigcup G_2 \bigcup G_3 \bigcup G_6 \bigcup G_7 \bigcup G_8 \bigcup G_9 \bigcup G_5$. 然后, 根据候选区域范围内的空间点与点 $P$ 的距离计算, 核实是否为点 $P$ 的 $Eps$-邻居点, 图 3 中粗圆区域内的空间点为点 $P$ 的 $Eps$-邻居点. 伪代码如 Step 1 所示.

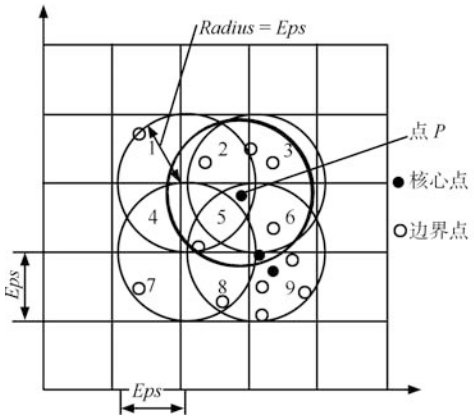



图 3　点 $P$ 的邻近 $Eps$-网格区域

Fig. 3　Near $Eps$-grid regions of points $P$

**Step 1.**

new point $P(x_1, x_2, \cdots, x_d)$;
$nearpoints \longleftarrow$ null;
for $i_1 \longleftarrow \lfloor x_1/Eps \rfloor - 1$ to $\lfloor x_2/Eps \rfloor + 1$ do
 for $i_2 \longleftarrow \lfloor x_2/Eps \rfloor - 1$ to $\lfloor x_2/Eps \rfloor + 1$ do
 $\cdots$ for $i_d \longleftarrow \lfloor x_d/Eps \rfloor - 1$ to $\lfloor x_d/Eps \rfloor + 1$ do
  $Key_{grid} \longleftarrow (i_1, i_2, \cdots, i_d)$;
  if ($KeyList$ include $Key_{grid}$)
   $nearpoints \longleftarrow nearpoints \cup Key_{grid}.points$
$Key_{grid} \longleftarrow (\lfloor x_1/Eps \rfloor, \lfloor x_2/Eps \rfloor, \cdots, \lfloor x_d/Eps \rfloor)$
if ($KeyList$ not include $Key_{grid}$)
 $Key_{grid}.points \longleftarrow \{p\}$;
 $KeyList \longleftarrow KeyList \cap \{Key_{grid}\}$;
 $nearpoints \longleftarrow nearpoints \cup \{p\}$
foreach point $q \in nearpoints$ do
 if (distance$(q, P) \le Eps$)
  $Neighbor(q) \longleftarrow Neighbor(q) \cup \{P\}$;
  $Neighbor(P) \longleftarrow Neighbor(P) \cup \{q\}$;
end

**步骤 2.** 对点 $P$ 的满足核心点条件的 $Eps$-邻居点做聚类处理.

OLDStream 首先对新点 $P$ 的满足了核心点条件的 $Eps$-邻居点做聚类处理, 如 Step 2 描述. 由于点 $P$ 的加入, 其 $Eps$-邻域点可能由边界点或孤立点变成了核心点, 由于孤立点不属于任何簇、边界点可以属于多个簇, 而核心点

只能属于一个簇, 这时必须进行簇更新操作. 对于这类空间点 $q$, 如果点 $q$ 尚未属于任何簇, 则新建簇, 并将点 $q$ 加入其 $Seeds\_list$; 若点 $q$ 已属于一个簇, 则将其 $Eps$-邻居点也加入该簇; 若点 $q$ 为边界点时属于多个簇, 则将其所在的簇合并为一个簇. 对于点 $P$ 加入之前已经满足核心点的 $Eps$-邻居点, 仅需要将点 $P$ 加入到其所在簇中. 伪代码中 $C(q)$ 表示包含点 $q$ 的簇.

**Step 2.**

foreach (point $q$ in $Neighbor_{Eps}(P)$)
 if ($|Neighbor_{Eps}(q)|) = MinPts$
  $q.core$ = true;
  if ($|q.clusters| = 0$)
   new cluster $C \longleftarrow \{q \cup Neighbor_{Eps}(q)\}$;
   $C.seeds \longleftarrow \{q\}$;
  else if ($|q.clusters| = 1$)
   $C(q).seeds \longleftarrow \{q \cup C(q).seeds\}$;
   $C(q) \longleftarrow C(q) \cup \{Neighbor_{Eps}(q)\}$;
  else if ($|q.clusters| > 1$)
   $\{C_1(q) \cup C_2(q) \cdots C_{q.clusters}(q)\}$;
   $C_1(q).seeds \cup C_2(q).seeds \cdots C_{q.clusters}(q).seeds$
 else if ($|Neighbor_{Eps}(q)| > MinPts$)
   $C(q) \longleftarrow C(q) \cup \{p\}$;
end foreach

**步骤 3.** 对新空间点 $P$ 做聚类处理.

在完成步骤 2 之后, 点 $P$ 在修正过程中可能被划分到多个簇中. 若点 $P$ 是边界点, 则无需再处理; 若点 $P$ 是核心点, 则需要对点 $P$ 所在簇再次做聚类更新. 如同对点 $P$ 的 $Eps$-邻域点的修正处理一样, 若点 $P$ 属于多个簇, 则需要对其所在空间簇进行合并操作. 步骤 3 执行过程如 Step 3 描述.

**Setp 3.**

if ($Neighbor_{Eps}(P) \geq MinPts$)
 $P.core$ = true;
 if ($|P.clusters| = 0$)
  new cluster $C \longleftarrow \{P \cup Neighbor_{Eps}(P)\}$;
  $C.seeds \longleftarrow \{P\}$;
 else if ($|P.clusters| = 1$)
  $C(P).seeds \longleftarrow \{P \cup C(P).seeds\}$;
  $C(P) \longleftarrow C(P) \cup \{Neighbor_{Eps}(P)\}$;
 else if ($|P.clusters| > 1$)
  $\{C_1(P) \cup C_2(P) \cdots C_{|P.clusters|}(P)\}$;
  $C_1(P).seeds \cup C_2(P).seeds \cdots C_{|P.clusters|}(P).seeds\}$
end if

OLDStream 执行过程其实就是不断地簇扩展和簇合并的过程, 因此, 簇合并过程的执行效率决定了算法的执行效率. 在 OLDStream 中, 每个簇维护一个核心点索引列表 $Seeds\_list$, 簇的合并仅需要根据一个簇的 $Seeds\_list$, 将其核心点及空间点加入到另一个簇中, 并将该簇的 $Seeds\_list$ 合并到另一个簇的 $Seeds\_list$, 即完成簇的合并. 这种方法有效地减少了聚类簇的搜索空间, 降低了簇合并的时间复杂度. 在大规模数据被接收后, 簇合并的执行时间也不会明显增加.

### 2.3 时间复杂度

设定已接收的数据集合为 $D$, OLDStream 算法主要包括三步: 第一步用于查找新点 $P$ 的 $Eps$-邻居点, 寻找邻近 $Eps$-网格区域为线性扫描, 由于 $Eps$-网格数目不确定, 并且候选 $Eps$-邻居点的核实也为线性扫描, 所以步骤 1 的执行时间最坏情况下为 O($|D|$). 步骤 2 需要对点 $P$ 的满足核心点的 $Eps$-邻居点做聚类处理, 假设点 $P$ 有 $k$ 个 $Eps$-邻居点满足核心点条件 ($0 \leq k \leq MinPts$), 最坏情况下, 这 $k$ 个邻域点都需要进行簇合并操作. 此时, 执行所需要的时间复杂度为 O($k|D|$), 因为一次簇合并最坏情况是扫描整个数据集合. 步骤 3 的处理是在点 $P$ 满足核心点的情况下执行的, 其时间复杂度为 O($|D|$). 因此, 当一个新点被聚类更新时, OLDStream 算法的时间复杂度为 O($k|D|$) ($0 \leq k \leq MinPts + 2$).

## 3 实验测试及分析

我们使用真实数据和大规模合成数据对 OLDStream 进行了聚类效果评估, 同时对 OLDStream 与 DBSCAN、CluStream、D-Stream 的运行效率进行了对比. 所有的算法采用 C# 实现, 整个实验在 vs2008 环境下进行测试, 实验平台配置为 I7 处理器, 2 GB 内存, 操作系统为 Windows 7.

真实数据 Data 1 采用 SEQUOIA 2000 数据库中一个子数据, 该数据集包含多种不规则形状密度聚集的 473 个空间数据点, 约覆盖 100 × 100 的区域, 同样用于聚类效果测试[9].

真实数据 Data 2 为微软亚洲研究院 Geolife 项目收集的轨迹数据的一个子集, 包括了 178 位用户从 2007 年 5 月 7 日至 2007 年 5 月 16 日之间的近 10 000 个地理位置数据, 空间数据大多分布在井字形的道路上, 区域面积约 150 平方千米.

合成数据 Data 3 由 Matlab 生成, 在 1 000 × 1 000 区域内共包含了 4 413 个数据点, 其中有 300 个噪声点. 该数据集设定了多个类球形、线条形、弧形聚集的空间数据点集合.

合成数据测试集 Data 4 ∼ Data 10 采用 Thomas Brinkhoff[19] 的基于网络的移动目标生成器合成, 我们使用生成器的奥尔登堡城市地图和默认参数设置分别生成了包含 1 000, 2 000, 3 000, 5 000, 8 000, 10 000, 20 000 个移动点共 8 组数据集. 各数据集数据分布在较密集的网状道路上, 约覆盖了奥尔登堡 180 平方千米的区域范围.

聚类效果的评估采用了与 DBSCAN 算法、D-Stream 算法对比和 CDbw 有效性指数聚类评估[20] 两种方法. CDbw 有效性评估指数值越高表示聚类效果越好.

### 3.1 聚类效果测试

OLDStream 算法在 Data 1, Data 2, Data 3 和 Data 9 上做了聚类效果测试, 测试的方法是: 算法按一定的速率分别按正序、倒序和随机三种方式读取数据集. OLDStream 能发现所有的任意形状的簇, 并能识别出孤立点. 聚类的效果如图 4 所示, 黑色点为孤立点, 图 4 (a), 4 (f), 4 (k), 4 (p) 为正序输入数据的聚类效果, 图 4 (b), 4 (g), 4 (l), 4 (q) 为倒序输入的效果, 图 4 (c), 4 (h), 4 (m), 4 (r) 为随机输入的效果. 图 4 (a) ∼ 4 (c) 为 OLDStream 在 Data 1 上聚类效果, 参数设置为 $Eps = 5, MinPts = 2$ 辆车. 图 4 (f) ∼ 4 (h) 为 OLDStream 在 Data 2 上的聚类效果, 参数设置为 $Eps = 100$ m, $MinPts = 20$ 辆车, 从聚类结果可以找到哪

些道路交通比较堵塞, 哪些道路交通比较通畅. 图 4 (k) ∼ 4 (m) 为 OLDStream 在 Data 3 上的聚类效果, 参数设置为 $Eps = 20, MinPts = 4$ 辆车. 图 4 (p) ∼ 4 (r) 为 OLDStream 在 Data 9 上的聚类效果, 参数设置为 $Eps = 150$ 米, $MinPts = 10$ 车辆. 实验结果显示数据的输入顺序会影响算法的执行效率, 但不会影响聚类效果. 此外, 还将 OLDStream 的聚类效果与 DBSCAN, D-Stream 的聚类效果进行了对比. 图 4 (d), 4 (i), 4 (n), 4 (s) 分别是 DBSCAN 在 Data 1, Data 2, Data 3 和 Data 9 的聚类效果. 图 4 (e), 4 (j), 4 (o), 4 (t) 分别是 D-Stream 在 Data 1, Data 2, Data 3 和 Data 9 上的聚类效果. 对比结果, OLDStream 和 DBSCAN 在相同参数下能获取到相同的聚类效果, OLDStream 算法以空间数据点为处理对象, 比基于网格划分的 D-Stream 算法有更好的聚类效果.

### 3.2　性能测试

#### 3.2.1　OLDStream 的效率测试

本节分别使用 Data 2 和 Data 9 中 10 000 个空间点的数据测试了 OLDStream 的运行效率, 每处理一个空间点记录其运行时间. 在 Data 2 设定输入参数 $Eps = 100$ m, $MinPts = 20$ 辆车; 在 Data 9 设定 $Eps = 100$ m, $MinPts = 10$ 辆车. 测试实验在 Data 2 和 Data 9 上分别发现 28 个和 39 个聚类簇, 随着数据的接收, 聚类的运行时间如图 5 所示. 从实验分析结果来看, 随着数据的增加, 最坏情况下的运行时间与数据集大小呈线性关系, 并没有急剧增加, 符合算法复杂度的分析, 同时验证了 OLDStream 具有较高的可伸缩性. 在两个数据集上的实验数据统计分析显示仅有 4 % 的空间点聚类消耗了最坏运行时间, 对每个空间点聚类的平均运行时间约为 0.033 ms.

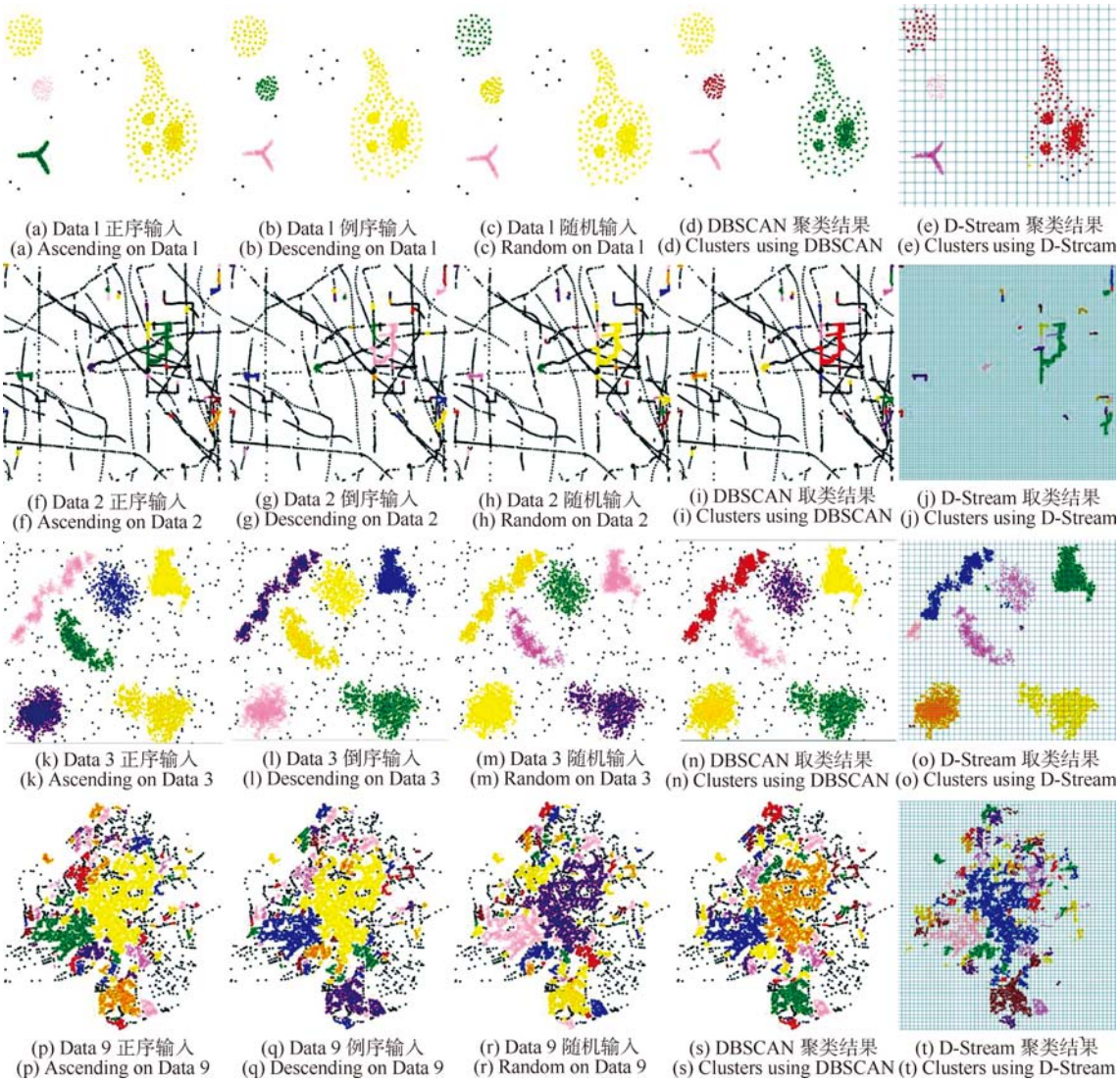
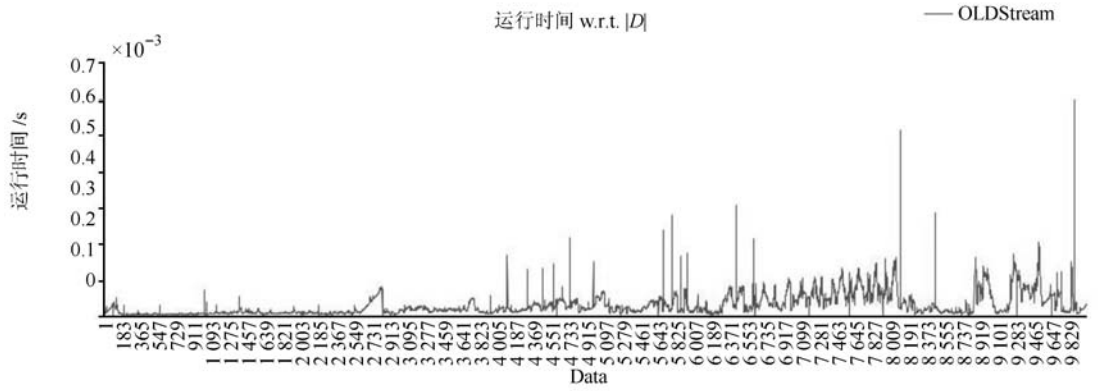
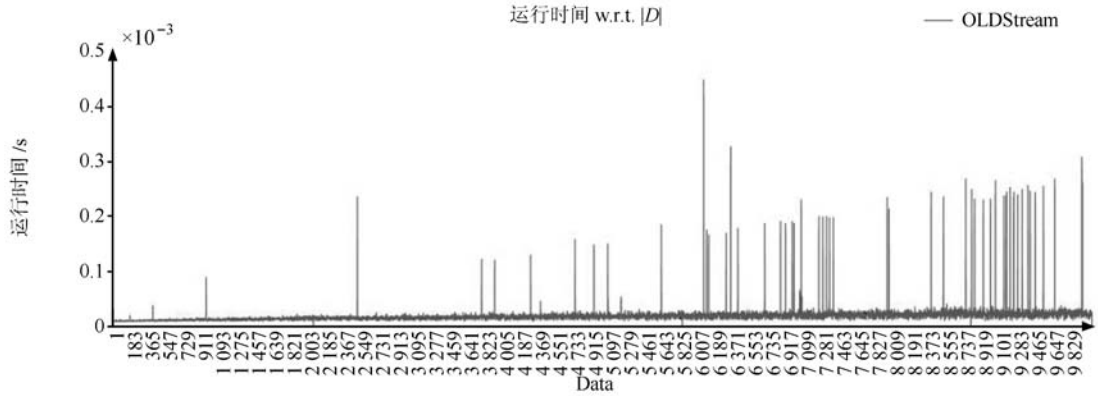



图 4　聚类效果对比

Fig. 4　Comparisons of clustering quality




(a) 在 Data 2 上的运行时间
(a) Running time on Data 2




(b) 在 Data 9 上的运行时间
(b) Running time on Data 9

图 5 OLDStream 在 Data 2 和 Data 9 上的聚类时间 (w.r.t. = with respect to)
Fig. 5 Clustering time of OLDStream on Data 2 and Data 9

表 1 运行时间比较 (s)
Table 1 Comparisons of running time (s)

| Algorithm | Data 2 | Data 4 | Data 5 | Data 6 | Data 7 | Data 8 | Data 9 | Data 10 |
|---|---|---|---|---|---|---|---|---|
| OLDStream (*Eps = 100, MinPts = 10*) | 0.68676 | 0.01853 | 0.02484 | 0.05226 | 0.08316 | 0.17013 | 0.24097 | 0.73709 |
| DBSCAN (*Eps = 100, MinPts = 10*) | 2.63857 | 0.0491 | 0.11606 | 0.24579 | 0.65415 | 1.68331 | 2.55077 | 10.0697 |
| OLDStream (*Eps = 150, MinPts = 10*) | 1.12228 | 0.01866 | 0.02641 | 0.05712 | 0.10441 | 0.20474 | 0.30895 | 0.83321 |
| DBSCAN (*Eps = 150, MinPts = 10*) | 2.70234 | 0.04310 | 0.11533 | 0.24615 | 0.65499 | 1.66096 | 2.57236 | 10.20404 |
| CluStream | 3.52261 | 0.68405 | 1.0776 | 1.62167 | 2.1662 | 2.8199 | 3.21811 | 4.61172 |
| D-Stream | 1.02516 | 0.04296 | 0.08157 | 0.13107 | 0.22064 | 0.44364 | 0.55160 | 0.98274 |

#### 3.2.2 聚类时间比较

本小节在 Data 2, Data 4 ∼ Data 10 数据集上比较了 OLDStream 和 DBSCAN、CluStream、D-Stream 的运行效率. 测试结果如表 1 所示.

OLDStream 算法对新数据点的聚类更新仅仅进行局部空间点的距离计算和簇合并操作, 其计算量较少; 而 DBSCAN 需要计算所有空间数据点两两间的距离; CluStream 需要在线统计带时间信息的输入数据特征信息; D-Stream 算法采用网格密度计算代替大量距离计算以提高算法效率. 实验结果也表明 OLDStream 算法运行时间与数据集大小呈近似线性关系, 相比带 $R$ 树索引的 DBSCAN 和 CluStream 算法具有较高的运行效率, 与 D-Stream 算法具有相近的运行效率, 但 OLDStream 比 D-Stream 算法具有更好的聚类效果.

### 3.3 输入参数敏感度分析

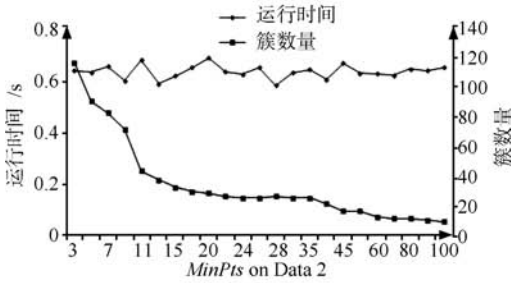
OLDStream 算法采用了两个输入参数: $Eps$ 和 $MinPts$, 用于决定输入空间数据聚类的密度阈值, 输入参数的选择必定会对聚类效率和聚类质量产生一定影响.

本节在 Data 2 和 Data 3 上进行了输入参数对 OLDStream 算法的效率和聚类效果影响的测试. 聚类效果的评估使用 CDbw 指数方法. 测试结果如图 6 和图 7 所示, 图 6 为 $Eps$ 和 $MinPts$ 对算法效率的影响分析, 左侧纵坐标表示聚类时间, 右侧纵坐标表示发现的聚类簇数量. 图 7 为 $Eps$ 和 $MinPts$ 对算法聚类效果的影响分析, 左侧纵坐标表示 CDbw 指数, 右侧纵坐标表示聚类簇数量. 图 6 (a)、图 6 (b)、图 7 (a)、图 7(b) 为在 Data 2 上的测试结果, 图 6 (a) 和图 7 (a) 为设定 $Eps = 100$ m, $MinPts$ 输入变化对算法的影响, 图 6 (b) 和图 7 (b) 为设定 $MinPts = 20$ 辆车, $Eps$ 输入变化对算法的影响. 图 6 (c)、图 6 (d)、图 7 (c)、图 7 (d) 为在 Data 3 上的测试结果, 图 6 (c) 和图 7 (c) 为设定 $Eps = 20$, $MinPts$ 输入变化对算法的影响, 图 6 (d) 和图 7 (d) 为设定 $MinPts = 4$, $Eps$ 输入变化对算法的影响. 在 $Eps$ 确定的情况下, $MinPts$ 仅决定聚类簇密度的最小阈值, 对 OLDStream 算法效率影响都较小, 如图 6 (a) 和图 6 (c); 但对聚类效果具有一定影响, 然而 $MinPts$ 在某一区间变化对聚类效果影响较小且能获取到较好的聚类结果, 如图 7 (a) 和 7 (c) 所示, 在 Data 2 上 $MinPts$ 在 $20 \sim 25$ 之间能获取到较好的聚类效果, 在 Data 3 上 $MinPts$ 在 $4 \sim 5$ 之间能获取到较好的聚类效果. 在 $MinPts$ 确定的情况下, 随着 $Eps$ 的增大, 算法的搜索空间和距离计算、簇合并等计算量都会增加, 算法的效率也随之降低, 如图 6 (b) 和图 6 (d). 当 $MinPts$ 确定时, $Eps$ 的变化同样也对算法聚类效果产生一定影响, 但在一个相当的 $Eps$ 范围内, OLDStream 算法都能获取到效果较好的聚类结果, 如图 7 (b) 中 $Eps$ 在 $80 \sim 100$ 之间、图 7 (d) 中 $Eps$ 在 $20 \sim 28$ 之间都能获取效果较好的聚类簇.

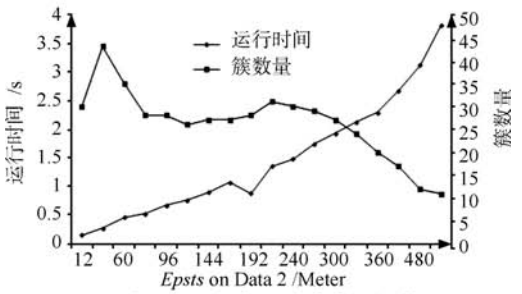
由上述分析可知, OLDStream 算法对输入参数具有一定敏感性, 但输入参数在上述基准参数周围变化对聚类效率和效果影响较小, 因此, 对于输入参数的选择, 要尽量在一个适当的范围内选择. 多数基于密度聚类方法输入参数的选择都要依赖输入数据的特征, 但由于 OLDStream 需要处理的是空间数据流, 无法提前获取全部数据特征. 因此, 本文输入参数的选择既考虑了数据特征又考虑应用背景. 在应用背景下的部分测试数据集上参考 Lee 等[21] 使用的熵理论方法确定最优的输入参数. 图 7 的实验结果也验证了使用该方法选择的输入参数 (例如在 Data 2 上的 $Eps$ 设定为 100, Data 3 的 $Eps$ 设定为 20) 处于聚类效果较好且较稳定的合适范围之内.

## 4　结论

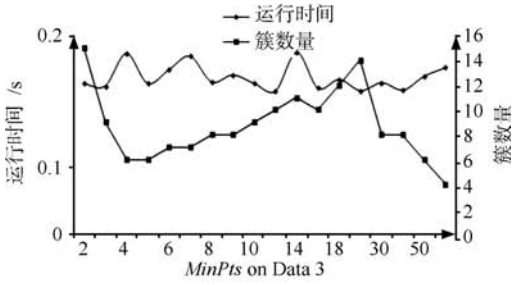
为了解决空间数据流聚类问题, 本文提出了一种基于密度的在线聚类算法以发现空间数据流中任意形状的聚类簇. 根据给出的空间数据流在线聚类描述, 完成了一个高效的基于密度的在线聚类算法的实现, 该实现算法可分为三步, 使用 $Eps$ 网格快速查找新加入点的邻近区域, 仅对新点及其满足核心点条件的 $Eps$-邻居点进行聚类更新处理. 综合实验结果显示了算法具有较高的聚类能力和可伸缩性, 对输入顺序不敏感, 运行效率好于带 R 树索引的 DBSCAN 和 CluStream 算法, 与 D-Stream 算法有相近的运行效率, 但聚类精度要好于 D-Stream 算法. 虽然输入参数对算法效率和聚类效果具有一定的影响, 但可以在应用环境下事先测试部分数据选择最优输入参数来尽量减少算法对输入参数的敏感度. 下一步的工作将重点研究如何降低算法对输入参数的依赖, 实现自适应的在线聚类, 并将算法应用于实时空间数据处理系统.



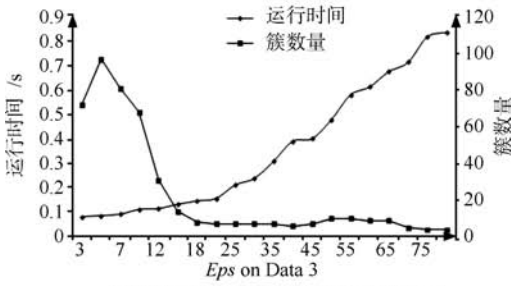
(a) 在 Data 2 上随 *MinPts* 变化的运行时间
(a) Running time vs. *MinPts* on Data 2



(b) 在 Data 2 上随 *Eps* 变化的运行时间
(b) Running time vs. *Eps* on Data 2



(c) 在 Data 3 上随 *MinPts* 变化的运行时间
(c) Running time vs. *MinPts* on Data 3

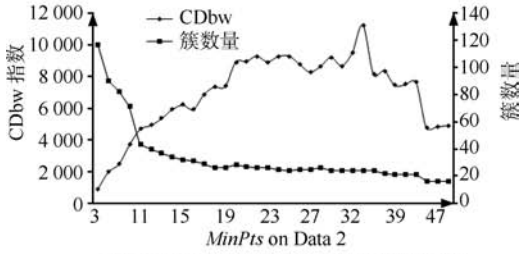


(d) 在 Data 3 上随 *Eps* 变化的运行时间
(d) Running time vs. *Eps* on Data 3

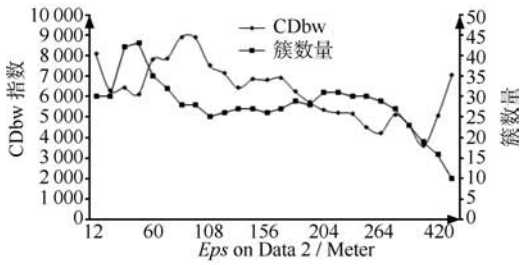
图 6　$Eps$ 和 $MinPts$ 对算法效率的影响分析
Fig. 6　Efficiency analysis of OLDStream w.r.t. $Eps$ and $MinPts$

## 致谢

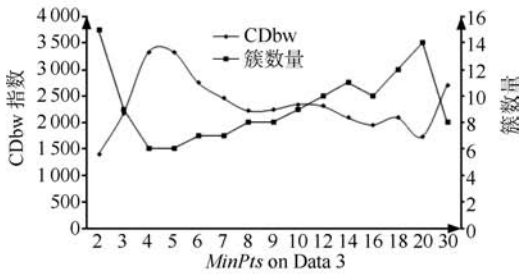
感谢北京科技大学万亚东老师在本文研究工作和稿件修改过程中所给予的帮助.



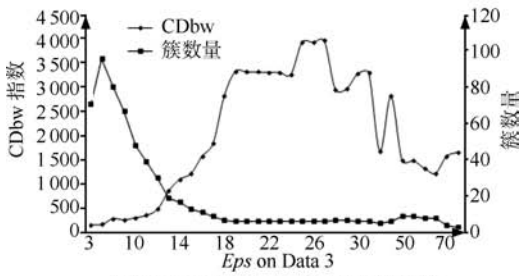

(a) 在 Data 2 上随 *MinPts* 变化的聚类评估
(a) Cluster evaluation vs. *MinPts* on Data 2




(b) 在 Data 2 上随 *Eps* 变化的聚类评估
(b) Cluster Evaluation vs. *Eps* on Data 2




(c) 在 Data 3 上随 *MinPts* 变化的聚类评估
(c) Cluster evaluation vs. *MinPts* on Data 3




(d) 在 Data 3 上随 *Eps* 变化的聚类评估
(d) Cluster evaluation vs. *Eps* on Data 3

图 7 *Eps* 和 $MinPts$ 对聚类效果影响的评估分析
Fig. 7 Clustering evaluation of OLDStream w.r.t. *Eps* and $MinPts$

## References


1 Lu Feng, Duan Ying-Ying, Yuan Wen. Data processing in location-based services. *Communication of the China Computer Federation*, 2010, **6**(6): 38−44
(陆锋, 段滢滢, 袁文. LBS 的数据处理技术. 中国计算机学会通讯, 2010, **6**(6): 38−44)

2 Guha S, Meyerson A, Mishra N, Motwani R, O′Callaghan L. Clustering data streams: theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 2003, **15**(3): 515−528

3 Han J W, Kamber M. *Data Mining Concepts and Techniques*. Beijing: China Machine Press, 2006. 196−211

4 Ester M, Kriegel H P, Sander J, Xu X W. 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. Portland, USA: AAAI Press, 1996. 226−231

5 Sander J, Ester M, Kriegel H P, Xu X W. Density-based clustering in spatial databases: the algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 1998, **2**(2): 169−194

6 Hinneburg A, Keim D A. An efficient approach to clustering in large multimedia databases with noise. In: Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining. New York, USA: AAAI Press, 1998. 58−65

7 Ma Shuai, Wang Teng-Jiao, Tang Shi-Wei, Yang Dong-Qing, Gao Jun. A fast clustering algorithm based on reference and density. *Journal of Software*, 2003, **14**(6): 1089−1095
(马帅, 王腾蛟, 唐世渭, 杨冬青, 高军. 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)

8 Chen Zhuo, Meng Qing-Chun, Wei Zhen-Gang, Ren Li-Jie, Dou Jin-Feng. A fast clustering algorithm based on grid and density condensation point. *Journal of Harbin Institute of Technology*, 2005, **37**(12): 1654−1657
(陈卓, 孟庆春, 魏振刚, 任丽婕, 窦金凤. 一种基于网格和密度凝聚点的快速聚类算法. 哈尔滨工业大学学报, 2005, **37**(12): 1654−1657)

9 Duan L, Xiong D Y, Lee J, Feng G. A local density based spatial clustering algorithm with noise. In: Proceedings of the 2006 IEEE International Conference on Systems, Man, and Cybernetics. Taipei, China: IEEE, 2006. 4061−4066

10 Ester M, Kriegel H P, Sander J, Wimmer M, Xu X W. Incremental clustering for mining in a data warehousing environment. In: Proceedings of the 24th Very Large Data Bases (VLDB) Conference. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 1998. 323−333

11 Aggarwal C C, Han J W, Wang J Y, Yu P S. A framework for clustering evolving data streams. In: Proceedings of the 29th Very Large Data Bases (VLDB) Conference. Berlin, USA: VLDB Endowment, 2003. 81−92

12 Zhu Wei-Heng, Yin Jian, Xie Yi-Huang. Arbitrary shape cluster algorithm for clustering data stream. *Journal of Software*, 2006, **17**(3): 379−387
(朱蔚恒, 印鉴, 谢益煌. 基于数据流的任意形状聚类算法. 软件学报, 2006, **17**(3): 379−387)

13 Cao F, Ester M, Qian W N, Zhou A Y. Density-based clustering over an evolving data stream with noise. In: Proceedings of the 2006 SIAM Conference on Data Mining. Bethesda, USA: SIAM Press, 2006. 326−337

14 Ren J D, Ma R Q. Density-based data streams clustering over sliding windows. In: Proceedings of the 6th International Conference on Fuzzy systems and Knowledge Discovery. Piscataway, USA: IEEE Press, 2009. 248−252

15 Ruiz C, Menasalvas E, Spiliopoulou M. C-DenStream: using domain knowledge on a data stream. In: Proceedings of the 12th International Conference on Discovery Science. Berlin, Heidelberg: Springer-Verlag, 2009. 287−301

16 MacQueen J. Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability. Berkeley, USA: University of California Press, 1967. 281−297

17 Chen Y X, Tu L. Density-based clustering for real-time stream data. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, USA: ACM, 2007. 133−142

18 Tu L, Chen Y X. Stream data clustering based on grid density and attraction. *ACM Transactions on Knowledge Discovery from Data*, 2009, **3**(3): 1−27

19 Thomas Brinko. Network-based Generator of Moving Object [Online], available: http://iapg.jadehs.de/ personen/brinkho?/generator/, April 19, 2005

20 Halkidi M, Vazirgiannis M. Clustering validity assessment using multi representatives. In: Proceedings of the 2nd Hellenic Conference on Artificial Intelligence (SETN), SETN 2002. Thessaloniki, New York: Springer, 2002. 237−248

21 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, China: ACM, 2007. 593−604

**于彦伟** 北京科技大学计算机与通信工程学院博士研究生. 主要研究方向为时空数据挖掘, 无线传感器网络. 本文通信作者.
E-mail: yuyanwei0530@gmail.com
(**YU Yan-Wei** Ph. D. candidate at the School of Computer and Communication Engineering, University of Science and Technology Beijing. His research interest covers spatio-temporal data mining and wireless sensor networks. Corresponding author of this paper.)

**王 沁** 北京科技大学计算机与通信工程学院教授. 主要研究方向为无线传感器网络, 无线定位, 数据处理, 计算机系统结构.
E-mail: wangqin@ies.ustb.edu.cn
(**WANG Qin** Professor at the School of Computer and Communication Engineering, University of Science and Technology Beijing. Her research interest covers wireless sensor networks, wireless localization, data mining, and computer architecture.)

**邝 俊** 北京科技大学计算机与通信工程学院硕士研究生. 主要研究方向为轨迹数据挖掘. E-mail: kuangjun0616@gmail.com
(**KUANG Jun** Master student at the School of Computer and Communication Engineering, University of Science and Technology Beijing. His main research interest is trajectory data mining.)

**何 杰** 北京科技大学计算机与通信工程学院博士研究生. 主要研究方向为室内定位技术, 无线传感器网络. E-mail: hejie1983@gmail.com
(**HE Jie** Ph. D. candidate at the School of Computer and Communication Engineering, University of Science and Technology Beijing. His research interest covers indoor positioning techniques and wireless sensor networks.)