

## 基于聚类的快速数据流匿名方法<sup>\*</sup>

郭 昆<sup>1</sup>, 张岐山<sup>2</sup>

<sup>1</sup>(福州大学 数学与计算机科学学院, 福建 福州 350108)

<sup>2</sup>(福州大学 管理学院, 福建 福州 350108)

通讯作者: 张岐山, E-mail: zhangqs@fzu.edu.cn, http://glxy.fzu.edu.cn/teachers.asp

**摘 要:** 为了防止敏感信息的泄漏, 保护用户隐私, 常采用概化和抑制等技术在共享数据前对其准标识符进行匿名化. 与静态数据集不同, 数据流具有潜在无限、高度动态等特性, 使得数据流匿名需要解决更加复杂的问题, 不能直接应用静态数据集的匿名方法. 在分析现有数据流匿名方法的基础上, 提出一种采用聚类思想进行数据流匿名的方法, 通过单遍扫描数据识别和重用满足匿名条件的簇, 以实现数据流的快速匿名. 真实数据集上的实验结果表明, 该方法在满足匿名要求的同时能够降低概化和抑制处理带来的信息损失, 并且具有较低的时间和空间复杂度.

**关键词:** 数据匿名; 数据流; 聚类

中图法分类号: TP311 文献标识码: A

中文引用格式: 郭昆, 张岐山. 基于聚类的快速数据流匿名方法. 软件学报, 2013, 24(8): 1852-1867. <http://www.jos.org.cn/1000-9825/4330.htm>

英文引用格式: Guo K, Zhang QS. Fast clustering-based anonymization algorithm for data streams. Ruan Jian Xue Bao/Journal of Software, 2013, 24(8): 1852-1867 (in Chinese). <http://www.jos.org.cn/1000-9825/4330.htm>

### Fast Clustering-Based Anonymization Algorithm for Data Streams

GUO Kun<sup>1</sup>, ZHANG Qi-Shan<sup>2</sup>

<sup>1</sup>(College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China)

<sup>2</sup>(College of Management, Fuzhou University, Fuzhou 350108, China)

Corresponding author: ZHANG Qi-Shan, E-mail: zhangqs@fzu.edu.cn, http://glxy.fzu.edu.cn/teachers.asp

**Abstract:** In order to prevent the disclosure of sensitive information and protect users' privacy, the generalization and suppression of technology is often used to anonymize the quasi-identifiers of the data before its sharing. Data streams are inherently infinite and highly dynamic which are very different from static datasets, so that the anonymization of data streams needs to be capable of solving more complicated problems. The methods for anonymizing static datasets cannot be applied to data streams directly. In this paper, an anonymization approach for data streams is proposed with the analysis of the published anonymization methods for data streams. This approach scans the data only once to recognize and reuse the clusters that satisfy the anonymization requirements for speeding up the anonymization process. Experimental results on the real dataset show that the proposed method can reduce the information loss that is caused by generalization and suppression and also satisfies the anonymization requirements and has low time and space complexity.

**Key words:** data anonymization; data stream; clustering

随着人们对个人隐私的日益关注, 如何在共享信息的同时保护个人和组织的隐私, 已成为近年来数据挖掘研究领域的重要研究方向<sup>[1]</sup>. 在共享数据前, 对敏感属性进行匿名处理是保护用户隐私的一种重要方法. 目前, 在静态数据的隐私保护方面已经有许多成熟的研究成果<sup>[2-7]</sup>.  $K$  匿名<sup>[8]</sup>、 $l$  多样性<sup>[9]</sup>等设计准则被广泛使用. 但是, 现实生活中的数据通常表现为动态变化的数据流, 如企业的销售记录、医疗机构的就诊信息、社交网络中的亲友信

\* 基金项目: 国家自然科学基金(70871024); 福建省自然科学基金(2010J01358); 福州大学科技发展基金(201-xy-16)

收稿时间: 2011-07-29; 修改时间: 2012-03-23; 定稿时间: 2012-10-19

息、传感器网络中的感应信号等.在发布数据流信息时,同样需要考虑隐私保护问题.

数据流具有潜在无限、流动迅速、变化频繁等不同于静态数据集的特点<sup>[10]</sup>,对算法的时间和空间复杂度提出了较高的要求:首先,数据量的无限性要求算法的空间复杂度不超过  $O(\log n)$ ;其次,数据的高流动性要求在时间上只能对数据进行单遍扫描,有时甚至需要丢弃部分数据以匹配速率;最后,数据变动的频繁性要求算法能够跟踪数据特征的变化,否则将带来较大的信息损失.因此,在数据流上进行匿名处理需要考虑更加复杂的问题.

本文提出了一种基于聚类的数据流匿名方法,与现有数据流匿名方法相比,其时间复杂度为线性.在真实数据集上的大量实验结果表明,该方法在满足匿名要求的同时,运行时间显著降低,且匿名后数据仍保持较高的可用性.

## 1 相关工作

目前,数据流匿名方法主要有 Li 等人提出的基于扰动的方法<sup>[11]</sup>、Cao 等人提出的 CASTLE 算法<sup>[12]</sup>、Zhou 等人提出的随机化方法<sup>[13]</sup>、Li 等人提出的 SKY 算法<sup>[14]</sup>、Wang 等人提出的 SWAF 算法<sup>[15]</sup>、Zhang 等人提出的 KIDS 算法<sup>[16]</sup>、Wang 等人提出的 B-CASTLE 算法<sup>[17]</sup>、Hessam 等人提出的 FAANST 算法<sup>[18]</sup>等.

基于扰动的方法通过向数据中加入随机噪声实现数据流匿名,但仅适用于数值型数据.CASTLE 算法、B-CASTLE 算法和 FAANST 算法基于聚类思想,通过从数据流中识别出簇结构并发布满足匿名要求的簇包含的元组实现数据流匿名,若用  $S$  表示数据流,则这一类算法的时间复杂度为  $O(|S|^2)$ ,不适用于大规模数据流.此外,这类算法的空间复杂度为  $O(|S|)$ ,根据数据流潜在无限的特点,这意味着算法需要无限的空间,这是不可接受的.文献[13]中的随机化方法、SKY 算法、SWAF 算法和 KIDS 算法等需要维护一棵以元组集合及其概化信息为节点的树,不断用新到达元组更新树,并发布满足匿名要求节点包含的元组以实现数据流匿名.若对树的节点数量做限制,则需要定期删除已发布节点.设  $\delta$  为发布时限,则算法的平均时间复杂度为  $O(|S|\delta \log \delta)$ ,最坏时间复杂度为  $O(|S|\delta^2)$ (当树退化时),算法的空间复杂度为  $O(\delta)$ ,运行速度快于基于聚类的方法.此外,其空间复杂度不超过一个限值.但是,删除节点使节点包含的元组概化信息不能被新到达元组所利用,输出元组的信息损失较大.若不限制树的节点数量,则通过重用已发布节点中的元组概化信息可以减小信息损失,但此时算法的平均时间复杂度为  $O(|S|^2 \log |S|)$ ,最坏时间复杂度达到  $O(|S|^3)$ ,空间复杂度为  $O(|S|)$ ,已不适用于数据流应用.

## 2 数据流匿名

### 2.1 基本概念

将  $K$  匿名概念扩展到数据流可以定义  $K$  匿名数据流.

**定义 1(K 匿名数据流).** 设数据流为  $S(pid, a_1, \dots, a_m, q_1, \dots, q_n, ts)$ ,其中,  $pid$  为用户身份标识,  $q_1, \dots, q_n$  为准标识符属性(quasi-identifier,简称 QI,是指能够联合额外获得的信息唯一确认用户身份的属性<sup>[8]</sup>),  $a_1, \dots, a_m$  为其余非 QI 属性,  $ts$  为元组到达时刻.设  $S'$  为由  $S$  生成的不包含  $pid$  和  $ts$  属性的匿名数据流.若  $S'$  满足以下条件,则称  $S'$  为  $K$  匿名数据流:

- (1) 对  $\forall t \in S, \exists t' \in S'$  与  $t$  对应;
- (2) 对  $\forall t' \in S', |DP(EQ(t'))| \geq K, EQ(t')$  为  $S'$  中与  $t'$  在 QI 上取值相同的元组组成的等价类,定义为

$$EQ(t') = \{t^* | t^* \in S', q_i = t'.q_i, i=1, \dots, n\},$$

$DP(EQ(t'))$  为  $EQ(t')$  中元组对应的用户集.

与静态数据集不同,匿名数据流发布受到时延约束.

**定义 2(时延约束).** 设有数据流  $K$  匿名方法  $F$ ,若  $F$  输出的  $K$  匿名数据流  $S'$  满足  $\forall t' \in S', t'.ts - t.ts < \delta, t$  为原数据流  $S$  中与  $t'$  对应的元组,  $\delta$  为给定正整数,则称  $F$  满足时延约束  $\delta$ .

概化(generalization)和抑制(suppression)等常用静态数据匿名技术同样可以应用于数据流匿名.概化将元组具体的属性值替换成较抽象概括的值,如将具体年龄 34,40,50 替换成区间[34-50],将小学、中学、大学替换

成学校等.概化使同一属性值对应的元组数增加,从而降低元组被攻击者识别的概率.抑制是一种特殊的概化,用最抽象的值代替元组具体的属性值,使所有元组在该属性上的值都相同,例如,将所有年龄都替换成 $[-\infty, +\infty]$ ,从而最大程度地保护用户隐私.在此基础上定义元组的概化.

**定义 3(元组的概化).** 设元组  $t(pid, v_{a1}, \dots, v_{am}, v_{q1}, \dots, v_{qn}, ts)$  的概化为  $g(g_1, \dots, g_n) = (f_1(v_{q1}), \dots, f_n(v_{qn}), f(v_{qi}))$ , 则

- (1) 当  $q_i$  为数值型属性时,  $f_i(v_{qi}) = [r_{li}, r_{ui}]$ ,  $v_{qi} \in [r_{li}, r_{ui}]$ ;
- (2) 当  $q_i$  为分类型属性时,  $f_i(v_{qi}) = H_i$ ,  $H_i$  为与  $q_i$  对应的域概化结构(domain generalization hierarchy, 简称 DGH)中  $v_{qi}$  的上层节点.

数据流匿名就是将原始数据流  $S$  中的元组进行概化或抑制处理后,输出满足匿名要求的数据流  $S'$ ,并保证  $S'$  中元组的输出时延不超过  $\delta$ .然而,静态数据匿名方法不能直接应用于数据流匿名,主要有以下几个原因:(1) 静态数据匿名对实时性要求不高,由于已经证明实现最优  $K$  匿名为 NP 难问题<sup>[19]</sup>,寻找最优  $K$  匿名方案的算法需要指数复杂度时间,寻找近似最优  $K$  匿名方案的算法也需要较高阶多项式时间,不适用于实时性要求较高且存在时延约束的数据流匿名;(2) 为了达到较低的信息损失,静态数据匿名方法通常需要重复访问数据集,数据流潜在无限、流动迅速的特点使得这一点难以实现;(3) 静态数据匿名方法要求每个元组具有唯一性,即元组的  $pid$  属性值应互不相同,但数据流中经常存在  $pid$  值相同的元组,如一家商店的销售记录可能包含同一位顾客的多次购物信息,直接应用静态数据匿名方法可能生成不满足  $K$  匿名要求的数据;(4) 一些算法为了加快  $K$  匿名处理速度,采用弱  $K$  匿名<sup>[20]</sup>代替  $K$  匿名作为隐私保护要求.弱  $K$  匿名不要求等价类包含至少  $K$  个相同 QI 值的元组,只要求每个元组的 QI 值关联到不少于  $K$  个用户即可,这在数据流环境下将导致问题.例如,表 1 为一家医院的病人名单, QI 为性别和年龄和邮政编码.表 2 为与其对应的 3-匿名表.医院还同时发布经过匿名处理的药品销售记录.若一个攻击者发现某条药品销售记录为  $t(M, [35-40], [300000-350000], \text{cancer medicine})$ ,且该记录的 QI 值唯一.当不知道被攻击者 John 是否在该医院购买过药品时,他只能得出该记录可能与表 2 中的前 3 个病人有关的结论,这样就实现了弱 3-匿名, John 的信息没有被泄露.但是,一旦他通过其他途径得知 John 在该医院购买过药品,立即就能断定记录  $t$  与 John 有关,导致 John 的隐私被泄露.由此可见,数据流匿名需要处理比静态数据集匿名更加复杂的情况.

**Table 1** Original patient information table

**表 1** 原始病人信息表

<i>pid</i>	Name	Gender	Age	Zipcode
1	Tom	M	30	300001
2	Samuel	F	40	200010
3	Mike	M	50	320005
4	Alice	F	23	100006
5	John	M	35	310000
6	Danny	F	36	230007

**Table 2** 3-Anonymized patient information table

**表 2** 3-匿名病人信息表

Gender	Age	Zipcode
M	[30-50]	[300001-320005]
M	[30-50]	[300001-320005]
M	[30-50]	[300001-320005]
F	[23-40]	[100006-230007]
F	[23-40]	[100006-230007]
F	[23-40]	[100006-230007]

## 2.2 信息损失

对数据进行概化抑制等操作必然造成信息损失,从而降低数据可用性.目前,已有若干衡量信息损失大小的指标被提出来.分辨度指标(discernibility metric,简称 DM)<sup>[4]</sup>用等价类大小表示概化信息损失,等价类包含的元组越少则信息损失越小,但 DM 指标没有考虑数据分布的影响,同样大小的等价类,元组分散的等价类的信息损失显然应当大于元组分布集中的等价类.分类指标(classification metric,简称 CM)<sup>[21]</sup>用等价类中与多数元组的类别不同的元组(称为离群元组)所占的比例来表示概化信息损失,离群元组越少则信息损失越小,但 CM 指标需要事先知道元组的类别信息.概化损失指标(generalized loss metric,简称 GLM)<sup>[21]</sup>在计算信息损失时将等价类大小和数据分布结合考虑,是比较理想的信息损失指标.针对数据流应用将其做如下修改:

**定义 4(元组概化信息损失).** 元组  $t(pid, v_{a1}, \dots, v_{am}, v_{q1}, \dots, v_{qn}, ts)$  概化为  $g(g_1, \dots, g_n)$  后的信息损失定义为

$$Infloss(t, g) = \frac{1}{n} \sum_{i=1}^n Infloss(g_i) \quad (1)$$

$$Infloss(g_i) = \begin{cases} \frac{r_{ui} - r_{li}}{R_{ui} - R_{li}}, & g_i = [r_{li}, r_{ui}] \\ \frac{|H_i| - 1}{|DGH_i| - 1}, & g_i = H_i \end{cases} \quad (2)$$

公式(2)首先计算元组在每个 QI 上的信息损失:当 QI 为数值型属性时,信息损失为元组在该 QI 上概化后的区间 $[r_{li}, r_{ui}]$ 的长度与该 QI 的值域 $[R_{li}, R_{ui}]$ 的比值;当 QI 为分类型属性时,信息损失为元组在该 QI 上概化后对应的节点 $H_i$ 所覆盖的叶子节点数 $|H_i|$ 与该 QI 的 DGH 的所有叶子节点数 $|DGH_i|$ 的比值,节点 $H_i$ 所覆盖的叶子节点是指 DGH 中以 $H_i$ 为根的子树的叶子节点.然后再求所有 QI 信息损失的均值,作为元组概化后的信息损失.在此基础上,数据流概化平均信息损失定义为

**定义 5(数据流概化平均信息损失).** 设 $g_i$ 为元组 $t_i$ 的概化,则数据流 $S$ 到时刻 $t_p$ 为止的平均信息损失为

$$AvgInfloss(S, t_p) = \frac{1}{t_p} \sum_{t_i \in S, t_i \leq t_p} Infloss(t_i, g_i) \quad (3)$$

本文提出的方法采用上述公式来计算概化信息损失.

### 3 基于聚类的快速数据流匿名方法

#### 3.1 算法设计思想

##### 3.1.1 簇的概化及相关定义

从数据挖掘的角度来看,数据流匿名可被看成一种特殊的带有约束的聚类问题.首先,聚类生成的簇不能太小, $K$ 匿名要求簇的大小至少为 $K$ ;其次,需要为每个簇定义与其关联的概化,并以此为基础进一步定义元组间及元组与簇的距离,作为元组划分的依据;最后,元组间距离的定义需要考虑概化信息损失.下面给出簇的概化及相关概念的定义.

**定义 6(簇的概化).** 设簇 $C$ 的概化为 $g_c(g_1, \dots, g_n)$ ,则

- (1) 当 $q_i$ 为数值型属性时, $g_i = [r_{\min, i}, r_{\max, i}]$ , $r_{\min, i}$ 和 $r_{\max, i}$ 分别为簇内元组在 $q_i$ 上的最小值和最大值;
- (2) 当 $q_i$ 为分类型属性时, $g_i = H_{\text{lowest}, i}$ 为簇内元组在 $q_i$ 上的值所对应的 DGH 中节点的最低共同父节点.

**定义 7(元组间距离).** 设元组 $t_1$ 和 $t_2$ 均概化为 $g$ ,则两个元组间的距离为

$$Distance(t_1, t_2) = |Infloss(t_1, g) - Infloss(t_2, g)| \quad (4)$$

容易证明,该距离的定义满足以下 4 个公理:(1) 非负性, $Distance(t_1, t_2) \geq 0$ ;(2) 同一性, $Distance(t_1, t_1) = 0$ ;(3) 对称性, $Distance(t_1, t_2) = Distance(t_2, t_1)$ ;(4) 三角不等性, $Distance(t_1, t_3) \leq Distance(t_1, t_2) + Distance(t_2, t_3)$ .因此,该定义能够恰当地描述元组的相似程度.

**定义 8(元组到簇的距离).** 设 $g$ 为簇 $C$ 的概化,则元组 $t$ 到簇 $C$ 的距离为

$$Distance(t, C) = Infloss(t, g) \quad (5)$$

元组到簇的距离反映采用簇的概化发布元组的信息损失程度.概化程度越高的簇,在发布元组时的信息损失越大,元组到簇的距离也越大.因此,通过计算元组到簇的距离,选择距离元组最近的簇,能够最小化元组发布的信息损失.

随着簇的增长,簇内元组距离及元组到簇的距离均增大,从而簇的信息损失也增大.因此,簇也不能太大.此外,数据流潜在无限、流动迅速的特点要求算法的时间和空间复杂度不能太高.

##### 3.1.2 现有数据流匿名方法分析

目前,基于聚类的数据流匿名方法主要有 CASTLE 算法、B-CASTLE 算法和 FAANST 算法等,CASTLE 算法是其中的典型代表.该算法的主要设计思想是:(1) 维护两个集合:候选 $K$ 匿名簇集合 $Set_{mwc}$ 和已发布 $K$ 匿名簇集合 $Set_{wc}$ ;(2) 对于新到达元组 $t$ ,由信息损失限值 $\tau$ 和簇数限值 $\beta$ 决定是将 $t$ 加入某个候选 $K$ 匿名簇,还是创建一个包含 $t$ 的新簇并将其加入 $Set_{mwc}$ ;(3) 在发布元组 $t'$ 时,首先尝试发布 $t'$ 所在候选 $K$ 匿名簇或使用覆盖 $t'$ 的已

发布  $K$  匿名簇的概化发布  $t'$ , 再检查是否需要抑制  $t'$ , 最后通过簇合并发布  $t'$ ; (4) 在发布簇中元组时, 为了减小信息损失, CASTLE 算法将所有大于  $2K$  的簇分割成不小于  $K$  的簇后再发布. 已发布  $K$  匿名簇的信息损失若小于限值  $\tau$ , 则将其加入  $Set_{wc}$ . CASTLE 算法是一种连续数据匿名方法. 它并不缓存到达的元组, 而是从发布时限  $\delta$  开始连续发布到达时间超过  $\delta$  的元组.

文献[12]分析得到的 CASTLE 算法的时间复杂度为  $O(|Q||S|\delta^2/K)$ . 但实际上, CASTLE 算法在分割大于  $2K$  的簇为子簇集并将其中信息损失小于  $\tau$  的簇加入  $Set_{wc}$  时, 对  $Set_{wc}$  的大小没有限制(簇数限值  $\beta$  只限制  $Set_{wc}$  的大小). 随着新元组的不断到达,  $Set_{wc}$  不断增大, 其大小与  $|S|$  成正比. 由于每次发布元组  $t'$  时都需要从  $Set_{wc}$  中取出每个已发布  $K$  匿名簇判断是否覆盖  $t'$ , 这个判断过程需要的时间为  $O(|Q||S|^2)$ . 因此, CASTLE 算法的时间复杂度应为  $O(|Q||S|\delta^2/K + |Q||S|^2) = O(|Q||S|^2)$ . 此外,  $Set_{wc}$  的大小随  $|S|$  增大, 也导致算法的空间复杂度达到  $O(|Q||S|)$ , 这意味着算法占用的空间随着数据的不断到达而不断增加.

B-CASTLE 算法针对 CASTLE 算法产生的候选  $K$  匿名簇分布不均匀问题(即一个大簇包含绝大部分元组, 其他簇只包含少量元组), 通过限制候选  $K$  匿名簇的大小使簇分布均匀化, 减少了簇的分割操作, 使算法运行速度得到提高. 但由于  $K$  匿名簇集合的大小仍未受到限制, 该算法的时间和空间复杂度与 CASTLE 算法相同.

FAANST 算法摒弃了 CASTLE 算法复杂的簇合并和分割操作, 也不使用候选  $K$  匿名簇集合, 只维护已发布  $K$  匿名簇集合  $Set_{wc}$ . 其基本的设计思想是, 先缓存新到达的元组, 当发布时限到时再一次性发布缓存中的全部元组. 因此, FAANST 算法是一种批量数据匿名方法. 在发布元组  $t$  时, 先在  $Set_{wc}$  中查找覆盖  $t$  且信息损失最小的簇, 若找到则直接用找到的簇的概化发布  $t$ ; 否则, 用  $K$ -Means 算法将所有未被覆盖的元组划分成  $|S_r|/K$  个簇,  $S_r$  为剩余元组集合, 并发布大小不小于  $K$  的簇. 若已发布簇的信息损失小于限值  $\tau$ , 则将其加入  $Set_{wc}$ . 由于不对簇进行合并或分割, 且每隔  $\delta$  时间才执行一次发布, FAANST 算法的运行速度比 CASTLE 算法有较大提高. 但该算法也没有限制  $K$  匿名簇集合  $Set_{wc}$  的大小, 在  $Set_{wc}$  中查找覆盖  $t$  且信息损失最小的簇, 需要的时间仍为  $O(|Q||S|^2)$ , 因此其时间复杂度为  $O(|Q||S|^2/K)$ , 空间复杂度为  $O(|Q||S|)$ , 与 CASTLE 算法的阶相同, 但不受发布时限  $\delta$  的影响.

### 3.1.3 算法基本原则及框架

由前述分析可总结出在设计新的数据流匿名算法时需要考虑的基本原则:

- (1) 针对数据流潜在无限、快速到达的特点, 算法的时间复杂度为不能超过  $O(|S|)$ ;
- (2) 针对数据流潜在无限的特点, 算法的空间复杂度应与  $|S|$  无关;
- (3) 为了尽可能地减小信息损失, 应当充分利用已发布的  $K$  匿名簇.

满足原则(1)的关键是要对已发布  $K$  匿名簇集合  $Set_{wc}$  的大小设置上限  $C_s$ , 使查找覆盖待发布元组  $t$  的时间从  $O(|S|)$  减小为  $O(C_s)$ ,  $C_s$  为常量. 此时, 算法的时间复杂度下降为  $O(C_s|S|)$ ; 另一方面, 当  $Set_{wc}$  大小不超过  $C_s$  时,  $Set_{wc}$  占用的空间也从  $O(|S|)$  减小为  $O(C_s)$ , 原则(2)同时被满足. 原则(3)可直接体现在算法框架的设计中. 下面讨论常量  $C_s$  的取值.

一种简单的取值方案是将  $C_s$  作为算法参数留给用户决定, 这将增加算法参数数量, 且用户不易理解该值的含义. 因此, 考虑根据已有算法参数和  $C_s$  之间的关系来确定  $C_s$  的取值. 前述算法的共同参数是发布时限  $\delta$ 、信息损失限值  $\tau$  以及匿名参数  $K$ . 为简化讨论, 考虑每时刻到达 1 个元组的情况(若每时刻到达多个元组, 可以通过细化时刻使每时刻到达元组数不超过 1), 此时,  $\delta$  同时也是待发布元组数.

从减小信息损失角度考虑, 要求  $C_s$  尽可能地大. 因为  $C_s$  越大, 可供选择的  $K$  匿名簇越多, 找到满足条件的簇的概率越高. 从减少运行时间和内存占用角度考虑, 要求  $C_s$  尽可能地小, 从而减少查找次数和用于存储  $K$  匿名簇的空间. 如果能够控制  $C_s$  的大小, 使  $C_s \ll |S|/K$  (数据流中全部元组产生的簇不超过  $|S|/K$  个), 则  $C_s$  的增加对算法时间和空间的影响就不大. 因此, 在保证  $C_s \ll |S|/K$  的前提下,  $C_s$  的取值优先考虑减小信息损失, 其次考虑减小运行时间和内存占用.

考察算法参数与  $C_s$  间的关系: 首先, 当  $\delta$  增大时, 需要发布的元组数量增加, 为了减小信息损失,  $C_s$  也要增大, 以便从更多的  $K$  匿名簇中选择满足条件的簇, 因此,  $C_s \propto \delta$ ; 其次, 当  $\tau$  增大时, 信息损失小于  $\tau$  的  $K$  匿名簇增加, 也要增大  $C_s$ , 因此,  $C_s \propto \tau$ ; 最后, 当  $\delta$  不变时,  $K$  的增大将使产生的新簇减少,  $C_s$  可以取较小值, 因此,  $C_s \propto 1/K$ . 由于

$\tau \in [0, 1]$ , 可不考虑  $\tau$  对  $C_s$  的影响. 综上所述, 为了减小信息损失,  $C_s$  应满足  $C_s \propto \tau \delta / K$ . 另一方面, 每发布  $\delta$  个元组, 可能产生的新簇数最多可为  $\delta / K$ , 为了容纳这些簇,  $C_s$  不应小于  $\delta / K$  的倍数. 最后得到  $C_s$  的计算公式为

$$C_s = c_0 \delta / K \quad (6)$$

其中,  $c_0 \geq 1.0$  为倍率系数.

进一步考虑减小算法的时间和空间代价. 为了保证  $C_s \ll |S| / K$ , 要求  $c_0 \ll |S| / \delta$ . 因此, 对于数据分布变化较均匀的数据流,  $c_0$  可以取 1.0. 对于数据分布变化较剧烈的数据流,  $c_0$  可以取大于 1.0 的值. 通过实验我们发现,  $c_0$  的变化对算法的信息损失和运行时间影响很小(实验结果见第 5 节), 因此可以统一取  $c_0 = 1.0$ .

基于上述基本原则, 可以设计一种基于聚类的快速数据流匿名算法 FADS (fast clustering-based anonymization algorithm for data stream). 受 FANNST 算法的启发, 在保证信息损失较小的条件下, 为了加快新算法的运行速度并减少空间占用, 只需维护一个大小为  $C_s$  的已发布  $K$  匿名簇集合  $Set_{wc}$  即可. 当产生的新簇达到  $K$  匿名时即发布簇内所有元组, 避免簇的合并和分割操作. 由此得到 FADS 算法的基本框架如下:

- (1) 将新到达的元组保存至元组集合  $Set_{tp}$ ;
- (2) 当发布时限  $\delta$  到时, 对  $Set_{tp}$  中的每个元组  $t$ , 从  $Set_{wc}$  查找覆盖  $t$  且信息损失最小的簇, 并用该簇的概化发布  $t$ ;
- (3) 根据距离远近, 不断从  $Set_{tp}$  剩余元组中划分中大小至少为  $K$  的簇, 将信息损失小于  $\tau$  的簇加入  $Set_{wc}$ , 若  $Set_{wc}$  中簇已达  $C_s$  个, 则删除最早加入的簇;
- (4) 当不再有新数据到达时, 对  $Set_{tp}$  中的剩余元组重复执行步骤(2)和步骤(3).

### 3.2 算法实现

本节描述 FADS 算法的具体实现.

**算法 1.** FADS 算法.

输入: 数据流  $S$ 、QI 集合、匿名参数  $K$ 、发布时延  $\delta$ 、信息损失下限  $\tau$ .

输出: 满足  $K$  匿名要求的数据流.

步骤:

1.  $Set_{tp}$  为待发布元组集合, 初始化为空集.
2.  $Set_{wc}$  为已发布  $K$  匿名簇集合, 初始化为空集.
3. **while**  $S \neq \emptyset$  **do**
4.   从  $S$  中读取一个元组  $t$  加入  $Set_{tp}$ ;
5.   **if**  $|Set_{tp}| \geq \delta$  **then**
6.      $outputWithWorkCluster()$ ;
7.     **if**  $|Set_{tp}| \geq K$  **then**
8.        $outputWithCondensation()$ ;
9.     **else**
10.       将  $Set_{tp}$  中的所有元组抑制后输出, 并从  $Set_{tp}$  中删除已输出元组;
11.     **end if**
12.   **end if**
13. **end while**
14. **if**  $|Set_{tp}| > 0$  **then**
15.    $outputWithWorkCluster()$ ;
16. **end if**
17. **if**  $|Set_{tp}| \geq K$  **then**
18.    $outputWithCondensation()$ ;
19. **else**

20. 将  $Set_p$  中的所有元组抑制后输出;

21. **end if**

算法首先从数据流  $S$  中读取一个元组并加入  $Set_p$ (步骤 4),然后判断  $Set_p$  中的元组是否已经达到或超过  $\delta$  个(步骤 5).若判断成立,则说明已经有元组的存在时间达到发布时延限值,需要发布这些元组.在发布元组时,算法首先调用过程  $outputWithWorkCluster()$ ,试图将元组加入已发布的  $K$  匿名簇后输出(步骤 6).由于已发布的  $K$  匿名簇的大小至少为  $K$ ,加入新元组后簇的大小必定超过  $K$ ,因此不会违反  $K$  匿名要求.接着,若  $Set_p$  中还有未发布的元组,则算法调用过程  $outputWithCondensation()$ ,将这些元组聚合成  $K$  匿名簇后再发布(步骤 8).若元组数量不足  $K$  个,则将其抑制后输出(步骤 10).最后,当不再有新元组到达时,算法对  $Set_p$  中的剩余未发布元组再做一次相同处理:先调用过程  $outputWithWorkCluster()$ ,尝试将其加入已发布的  $K$  匿名簇后输出(步骤 15),再调用过程  $outputWithCondensation()$ ,尝试将剩余元组聚合成新簇输出(步骤 18),最后将剩余的不足  $K$  个元组抑制输出(步骤 20).过程  $outputWithWorkCluster()$  的具体实现如下:

过程 1.  $outputWithWorkCluster()$ .

```

1. foreach  $t_i \in Set_p$  do
2.    $Set_{min} = \emptyset$ ;
3.   在  $Set_{wc}$  中查找覆盖  $t_i$  且信息损失增量最小的簇(可能有多个),加入  $Set_{min}$ ;
4.   if  $Set_{min} \neq \emptyset$  then
5.     从  $Set_{min}$  中随机选择一个簇  $C_{min}$ ,用  $C_{min}$  的概化输出  $t_i$ ;
6.     从  $Set_p$  中删除  $t_i$ ;
7.   end if
8. end foreach

```

对于  $Set_p$  中的每一个元组  $t_i$ ,过程  $outputWithWorkCluster()$  在所有已发布的  $K$  匿名簇中查找能够覆盖  $t_i$  且信息损失最小的簇,将  $t_i$  加入该簇后输出.若满足条件的簇有多个,则从中随机选择一个.这样,在保证元组  $t_i$  概化后的信息损失最小的同时也满足了  $K$  匿名要求.过程  $outputWithCondensation()$  的具体实现如下:

过程 2.  $outputWithCondensation()$ .

```

1.  $Set_c = greedyCondense()$ ;
2. foreach  $C_i \in Set_c$  do
3.   用  $C_i$  的概化输出  $C_i$  内所有元组;
4.   if  $C_i$  的信息损失小于  $\tau$  then
5.     while  $|Set_{wc}| \geq c_0 \delta / K$  do
6.       删除  $Set_{wc}$  中最早加入的簇;
7.     end while
8.     将  $C_i$  加入  $Set_{wc}$ ;
9.   end if
10. end foreach

```

当  $Set_p$  中的元组不能被任一已发布的  $K$  匿名簇覆盖时,过程  $outputWithCondensation()$  调用函数  $greedyCondense()$ ,将这些元组划分成若干个  $K$  匿名簇并输出.接着,将信息损失小于  $\tau$  的  $K$  匿名簇加入  $Set_{wc}$ ,以便为下一轮的元组发布提供更多的选择.若  $Set_{wc}$  大小已达到  $c_0 \delta / K$ ,则先删除最早加入的簇后再添加新簇.函数  $greedyCondense()$  在划分元组时,采用与 kNN 算法相似的思想,但针对数据流匿名应用做了一些修改,其具体实现如下:

函数 1.  $greedyCondense()$ .

```

1.  $Set_c = \emptyset$ ;
2. while  $|Set_p| \geq K$  do

```

3. 从  $Set_{tp}$  中随机选择一个元组  $t$ ;
4. 计算剩余  $|Set_{tp}|-1$  个元组与  $t$  的距离,并按距离从小到大排序,选出前  $K-1$  个和  $t$  的  $pid$  不同的元组,与  $t$  组成新簇  $C_{new}$ ;
5. 将  $C_{new}$  加入  $Set_c$ ;
6. 从  $Set_{tp}$  中删除  $C_{new}$  包含的元组;
7. **end while**
8. **if**  $|Set_{tp}|>0$  **then**
9.   **foreach**  $t_i \in Set_{tp}$  **do**
10.     在  $Set_c$  中查找加入  $t_i$  后信息损失的增量最小的簇  $C_{min}$ ;
11.     将  $t_i$  加入  $C_{min}$ ;
12.     从  $Set_{tp}$  中删除  $t_i$ ;
13.   **end foreach**
14. **end if**
15. 返回  $Set_c$ ;

函数 *greedyCondense()* 首先从  $Set_{tp}$  中随机选择一个元组  $t$ , 找出其  $K-1$  个近邻. 由于数据流中可能存在多个  $pid$  相同的元组, 为了满足  $K$  匿名要求, 在搜索  $t$  的  $K-1$  个近邻时, 要求它们的  $pid$  不能与  $t$  相同. 然后, 将元组  $t$  与其  $K-1$  个近邻聚合成新簇  $C_{new}$ , 并将  $C_{new}$  加入簇集合  $Set_c$ . 这个过程不断迭代进行, 直到  $Set_{tp}$  中剩下的元组少于  $K$  个为止. 剩余元组依次分配到加入该元组后信息损失增量最小的簇. 最后返回  $Set_c$ .

### 3.3 算法复杂度分析

首先分析算法的时间复杂度. 先求算法调用的 3 个过程的时间复杂度:

1. 过程 *outputWithWorkCluster()* 的时间复杂度.  $Set_{tp}$  中共有  $\delta$  个待处理元组, 所以 **foreach** 循环的次数为  $\delta$ . 每次循环需要在  $Set_{wc}$  中查找能够覆盖元组  $t_i$  的簇, 设  $|Set_{wc}|=n_{wc}$ , 则查找时间为  $O(n_{wc})$ . 判断一个簇是否覆盖  $t_i$  需要时间  $O(|QI|)$ , 判断簇的信息损失是否最小的时间为  $O(1)$ , 输出元组  $t_i$  的概率化的时间为  $O(|QI|)$ , 因此, 过程 *outputWithWorkCluster()* 的时间复杂度为  $O(\delta n_{wc} |QI|)$ .
2. 函数 *greedyCondense()* 的时间复杂度. 步骤 2~步骤 7 的 **while** 循环次数不超过  $\delta/K$ . 在每次循环中, 求元组  $t$  的  $K-1$  个近邻的时间为  $O((K-1)(n_{tp}-K/2))$ ,  $n_{tp}$  为每次循环时  $Set_{tp}$  中的元组数. 由于每次循环都会生成一个大小为  $K$  的新簇,  $n_{tp}$  从最大值  $\delta$  开始每循环一次就减小  $K$ , 直至  $n_{tp} < K$ . 计算元组间距离需要的时间为  $O(|QI|)$ . 因此, 步骤 2~步骤 7 的 **while** 循环的时间复杂度为  $O(\delta^2 |QI|)$ . 步骤 9~步骤 13 的 **foreach** 循环次数小于  $K$ , 集合  $Set_c$  的大小不超过  $\delta/K$ . 因此, 该循环的时间复杂度为  $O(\delta)$ . 综合可得, 函数 *greedyCondense()* 的时间复杂度为  $O(\delta^2 |QI|)$ .
3. 过程 *outputWithCondensation()* 的时间复杂度. 集合  $Set_c$  的大小不超过  $\delta/K$ . 输出簇  $C_i$  内元组概率化的时间为  $O(K|QI|)$ , 则步骤 2~步骤 10 的 **foreach** 循环的时间复杂度为  $O(|QI|\delta)$ . 结合函数 *greedyCondense()* 的时间复杂度可知, 过程 *outputWithCondensation()* 的时间复杂度为  $O(\delta^2 |QI|)$ .

再求 FADS 算法的时间复杂度. 算法步骤 3~步骤 11 的 **while** 循环次数不超过  $|S|/\delta$ , 而  $n_{wc}$  从 0 开始每次循环最坏情况下的增量为  $\delta/K$ , 当增加到  $c_0\delta/K$  后停止增长. 因此, 该循环中过程 *outputWithWorkCluster()* 的总时间复杂度为

$$O\left(\delta |QI| \left( \sum_{i=0}^{c_0} i\delta/K + (|S|/\delta - c_0) \cdot c_0\delta/K \right)\right) = O(\delta |QI| (|S| + \delta/2 - c_0\delta/2)/K) = O(c_0\delta |QI| |S|/K).$$

循环中另一过程 *outputWithCondensation()* 的总时间复杂度为  $O(\delta |QI| |S|)$ . 每次循环步骤 10 输出的元组数少于  $K$ . 综合可得, 算法步骤 3~步骤 11 的 **while** 循环的时间复杂度为  $O(c_0\delta |QI| |S|/K)$ , 步骤 15 的时间复杂度为  $O(c_0\delta^2 |QI|/K)$ , 步骤 18 的时间复杂度为  $O(\delta^2 |QI|)$ , 步骤 20 的时间复杂度为  $O(K)$ . 因此, 算法总的复杂度为



$O(c_0\delta|QI|S/K)$ . 由于数据流  $|S| \gg \delta, K, |QI|$ , 算法总时间复杂度可简化为  $O(|S|)$ .

接下来分析算法的空间复杂度. 集合  $Set_{ip}$  占用的空间为  $O(\delta|QI|)$ . 集合  $Set_{wc}$  占用的空间为  $O(c_0\delta|QI|/K)$ . 函数  $greedyCondense()$  中, 存储新生成的簇需要的空间为  $O(\delta|QI|/K)$ . 因此, 算法总的空间复杂度为  $O(\delta|QI|)$ .

#### 4 实验结果

在这一节中, 我们通过实验对 FADS 算法的性能进行分析, 并将其与 CASTLE 算法和 FAANST 算法进行比较. 实验采用的数据集是在隐私保护文献中广泛使用的 UCI 的 Adult 数据集<sup>[22]</sup>. 在删除信息不完整的元组后, 实际用于实验的数据集包含 30 162 个元组. 实验采用的 QI 集合从属性集合  $\{age, final-weight, education-number, capital-gain, capital-loss, hours-per-week, education, marital-status, occupation, nation\}$  中选取, 前 6 个为数值型属性, 后 4 个为分类型属性. 对分类型属性采用与文献[3]相同的 DGH. 所有算法均采用 Java 语言实现. 实验的硬件配置为 Intel Core2 Duo 1.66GHz CPU, 2048MB RAM, 软件配置为 Microsoft Windows XP SP2, JDK 6.0.

实验比较了各种算法在数据量变化和参数变化条件下的性能. 由于 FAANST 算法只能处理数值型数据, 为了充分测试算法的性能, 设计了两组实验: 一组测试 FADS 算法和 CASTLE 算法, QI 集合从全部 10 个属性中选取; 另一组测试 FADS 算法、CASTLE 算法和 FAANST 算法, QI 集合从 6 个数值型属性中选取.

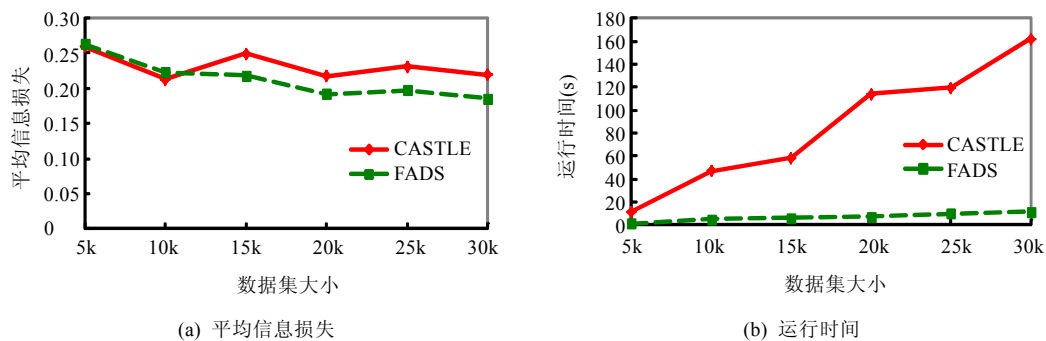
##### (1) FADS 算法和 CASTLE 算法的对比实验结果

首先测试了数据量变化对算法性能的影响, 算法参数按表 3 设置. 实验结果如图 1 所示.

**Table 3** Settings of the parameters of the algorithms running on the 1st experimental group

**表 3** 第 1 组实验算法参数设置

Algorithm	Parameter settings
FADS	$K=100,  QI =10, \delta=10000, \tau=0.5, c_0=1.0$
CASTLE	$K=100,  QI =10, \delta=10000, \beta=50, \mu=100$



**Fig.1** Average information loss and runtime with varying data size (the 1st experimental group)

**图 1** 不同数据量的平均信息损失和运行时间(第 1 组实验)

从图 1(a)可以看出, 随着数据量的增加, 两种对比算法的平均信息损失均逐渐降低. 这是因为随着新数据的不断到达, 新簇不断产生并加入已发布  $K$  匿名簇集合, 使后来到达的元组被某个  $K$  匿名簇覆盖的概率逐渐上升, 从而减小了信息损失. 另外, FADS 算法的平均信息损失在数据量小于等于 10k 之前与 CASTLE 算法基本相同, 当数据量超过 10k 后开始低于 CASTLE 算法, 且降低的速度逐渐加快. 这主要是因为 FADS 算法只保留已发布  $K$  匿名簇集合, 所有满足条件的新簇都加入该集合以供后续查找, 而 CASTLE 算法还有一部分可能覆盖新元组的簇存于待发布  $K$  匿名簇集合, 不能被充分利用. 因此, FADS 算法能够在更大的范围内查找满足条件的簇, 从而得到更低的信息损失. 到达的元组越多, 两种算法之间的差异就越明显. 图 1(b)显示: FADS 算法的运行时间远少于 CASTLE 算法, 即使当数据量达到 30k 时, 也只需要 10s 左右的时间; 相反地, CASTLE 算法在数据量超过 10k 时已经需要 1min 左右的时间. 由第 3.1 节的分析可知, CASTLE 算法时间复杂度与数据量成平方关系, 因此随着

数据量的增加,其运行时间迅速增加.FADS 算法的线性时间复杂度使其运行时间仅随数据量线性增加,且由于  $|S| \gg \delta, |Q|$ , 增加速度较缓慢,符合第 3.3 节的分析.

接着测试了参数值变化对算法性能的影响.每次实验只改变 1 个参数的值,其他参数按表 3 取值.测试时使用全部 30 162 个元组. $\delta$ 值、 $K$ 值和  $QI$ 数量变化时的实验结果分别显示在图 2~图 6 中.

$\delta$ 对于两特困户算法都是很重要的参数,这一点从图 2(a)可以反映出来.随着 $\delta$ 值的增大,两个算法的平均信息损失均较快下降,但当降低到一定程度时( $\delta=10000$ 附近),减速趋缓.FADS 算法的平均信息损失始终小于 CASTLE 算法,其  $K$ 匿名簇集合的设计在这里起主要作用.图 2(b)表明, $\delta$ 值的变化对 CASTLE 算法的运行时间有显著影响,因为其时间复杂度与 $\delta^2$ 成正比.由于 CASTLE 算法不限制  $K$ 匿名簇数量,随着 $\delta$ 值的增大,发布元组时需要查找的  $K$ 匿名簇增多,从而进一步增加了运行时间.FADS 算法的时间复杂度与 $\delta$ 成正比,且对  $K$ 匿名簇集合大小的限制使其在发布元组时需要查找的  $K$ 匿名簇不超过一个上限,因此,运行时间仅随 $\delta$ 值的增大线性增加,且增加的速度远小于 CASTLE 算法.

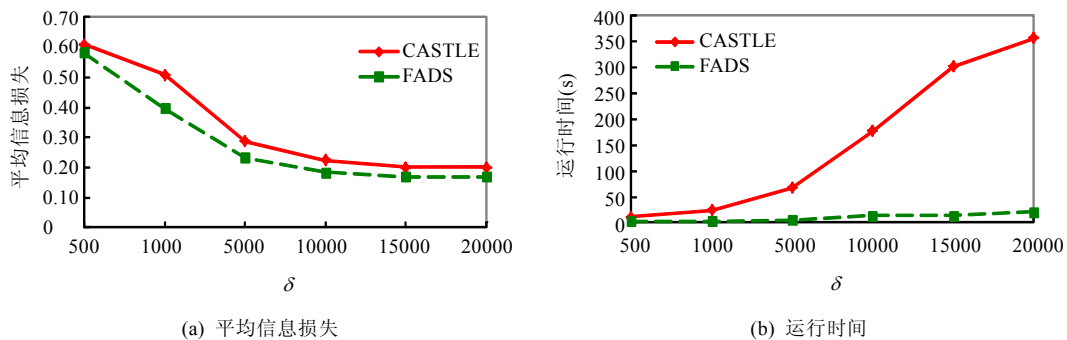


Fig.2 Average information loss and runtime with varying  $\delta$  (the 1st experimental group)

图 2 不同 $\delta$ 值的平均信息损失和运行时间(第 1 组实验)

从图 3(a)可以看出,平均信息损失随  $K$  值的增大而增加.这是由于  $K$  值越大,一个簇在发布时需要包含的元组就越多,从而使簇的信息损失增加.FAD 算法由于能够充分利用已发布  $K$ 匿名簇发布元组,减少新簇生成,获得比 CASTLE 算法更低的平均信息损失.从时间复杂度上分析,CASTLE 算法和 FADS 算法的运行时间应当随  $K$  值的增大而减少.图 3(b)显示,FADS 算法基本符合这一预测,而 CASTLE 算法的运行时间在  $K \geq 50$  时随  $K$  变化不大,这是因为其时间复杂度主要受  $|S|^2$  项控制, $K$  值的变化对其运行时间影响不大.但是当  $K=10$  时,CASTLE 算法的运行时间却明显减小.为了研究这一现象的原因,我们对算法在  $K$  从 10 增大至 50 时的运行时间进行了实验,结果如图 4 所示.

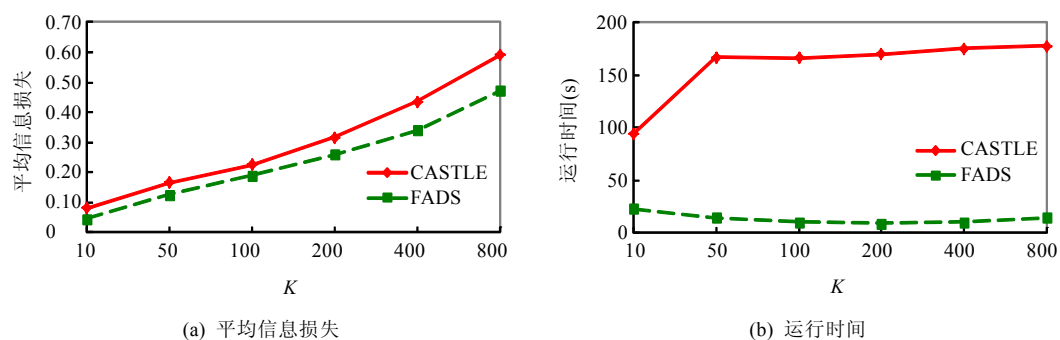


Fig.3 Average information loss and runtime with varying  $K$  (the 1st experimental group)

图 3 不同  $K$  值的平均信息损失和运行时间(第 1 组实验)

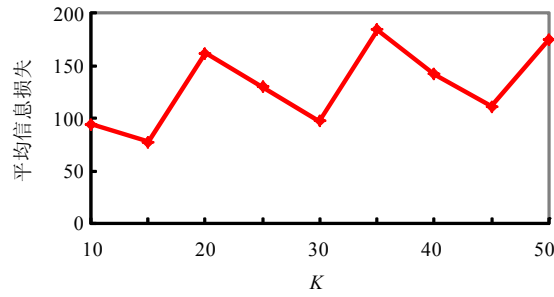


Fig.4 Runtime of CASTLE when K varies from 10 to 50

图4 CASTLE 算法当 K 在 10~50 之间变化时的运行时间

由图 4 可知,算法的运行时间当 K 在 10~50 之间变化时呈波动变化,并非线性增加,结合后面图 8 显示的实验结果,我们认为,这主要是由 K 值较小时分类型属性值对 CASTLE 算法的影响所致.

从图 5(a)可以看出,QI 数量的增加使算法的平均信息损失增大.这主要是由于 QI 数量越多,需要概化的属性也越多,元组概化后的信息损失也就越大.FADS 算法能够更好地利用已发布 K 匿名簇,因此得到比 CASTLE 更小的平均信息损失.图 5(b)表明,两种算法的运行时间均随 QI 数量的增加而上升,但 FADS 算法的上升速度明显慢于 CASTLE 算法.这主要是由于随着 QI 数量的增加,FADS 算法的运行时间以 $|S|$ 数量级增加,而 CASTLE 算法以 $|S|^2$ 数量级增加,当 $|S|$ 较大时,后者要明显大于前者.

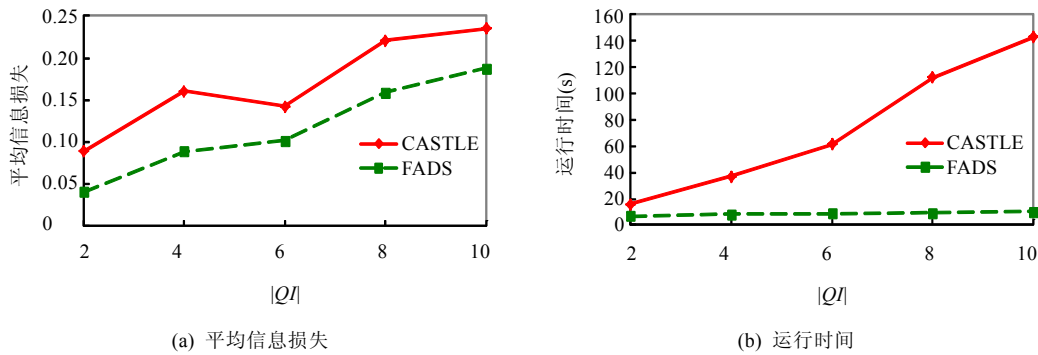


Fig.5 Average information loss and runtime with varying QI number (the 1st experimental group)

图 5 不同 QI 数量的平均信息损失和运行时间(第 1 组实验)

(2) FADS 算法和 CASTLE 算法、FAANST 算法的对比实验结果

首先测试数据量变化对算法性能的影响,算法参数按表 4 设置,实验结果如图 6 所示.

Table 4 Settings of the parameters of the algorithms running on the 2nd experimental group

表 4 第 2 组实验算法参数设置

Algorithm	Parameter settings
FADS	$K=100,  QI =6, \delta=10000, \tau=0.5, c_0=1.0$
CASTLE	$K=100,  QI =6, \delta=10000, \beta=50, \mu=100$
FAANST	$K=100,  QI =6, \delta=10000, \tau=0.5$

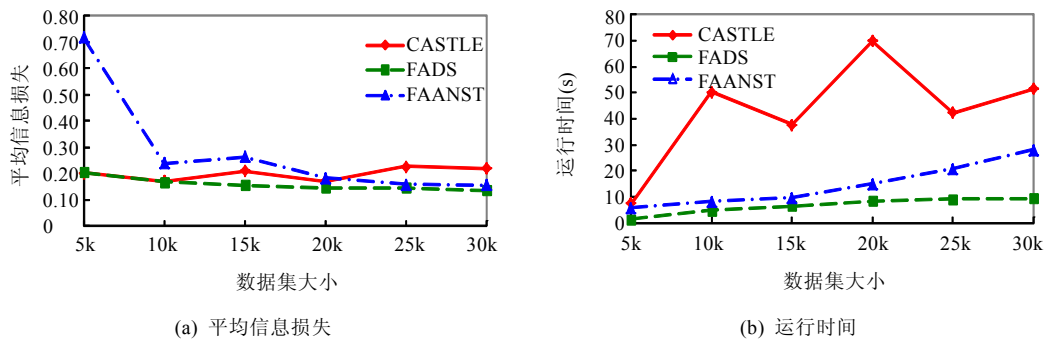


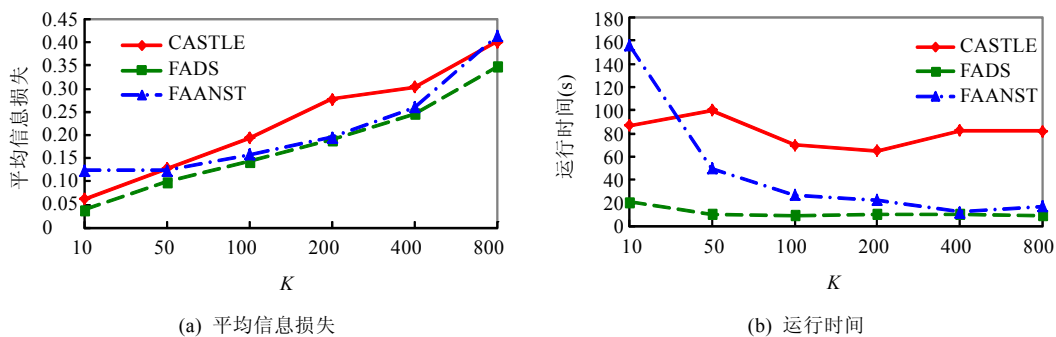
Fig.6 Average information loss and runtime with varying data size (the 2nd experimental group)

图 6 不同数据量的平均信息损失和运行时间(第 2 组实验)

从图 6(a)可以看出,当数据量较小时,FAANST 算法的平均信息损失略大于 CASTLE 算法,当数据量超过 20k 后开始小于 CASTLE 算法,且随着数据量的增加逐渐接近 FADS 算法.而 FADS 算法的平均信息损失不但小于 CASTLE 算法,而且小于 FAANST 算法,这主要是由于 FADS 算法中的函数 *greedyCondense()* 生成的  $K$  匿名簇的信息损失小于 FAANST 算法采用的  $K$ -Means 算法.此外,FAANST 算法的曲线波动较大,显示其采用的  $K$ -Means 算法对其稳定性影响较大.图 6(b)显示,FAANST 算法和 FADS 算法的运行速度均快于 CASTLE 算法,但由于 FAANST 算法的时间复杂度仍与  $|S|^2$  成正比,在数据量增大时,其运行时间与 FADS 算法的差距逐渐明显.

接着测试参数值变化对算法性能的影响.每次实验只改变 1 个参数的值,其他参数按表 4 取值. $\delta$ 值、 $K$  值和 QI 数量变化时的实验结果分别显示在图 7~图 9 中.

图 7(a)表明, $\delta$ 值对所有算法的影响都非常显著.FADS 算法由于限制了  $K$  匿名集合的大小,受  $\delta$ 值变化的影响小于其他两个算法,且其平均信息损失最小.FAANST 算法的平均信息损失接近 FADS 算法,但受  $K$ -Means 算法的影响,起伏变化较大.从图 7(b)可以看出, $\delta$ 值对 FAANST 算法和 FADS 算法的运行时间影响很小,但 FADS 算法的运行速度仍快于 FAANST 算法,这主要是由于 FAANST 算法中的  $K$ -Means 算法为了达到收敛需要的迭代次数,影响了其运行时间.

Fig.7 Average information loss and runtime with varying  $K$  (the 2nd experimental group)图 7 不同  $K$  值的平均信息损失和运行时间(第 2 组实验)

从图 8(a)可知,随着  $K$  值的增大,所有算法的平均信息损失都有所增加.FAANST 算法的平均信息损失与 FADS 算法接近,但仍略高于 FADS 算法,这主要是由于 FADS 算法生成的  $K$  匿名簇的信息损失较小.图 8(b)显示,所有算法的运行时间均随  $K$  值的增大而减少,但 FADS 仍是其中最快的.FAANST 算法的运行时间随  $K$  值的变化下降较明显,这主要是因为 FAANST 算法的时间复杂度直接与  $K$  值成反比,而其他两种算法受  $K$  值的影响较小.

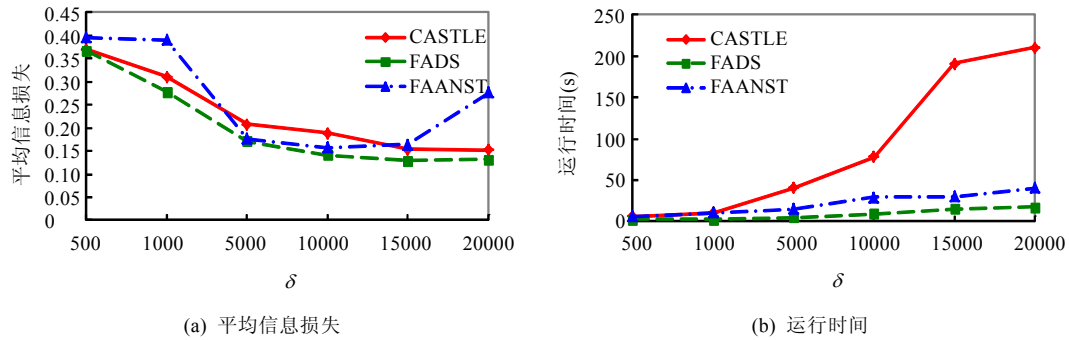
Fig.8 Average information loss and runtime with varying  $\delta$  (the 2nd experimental group)图 8 不同 $\delta$ 值的平均信息损失和运行时间(第 2 组实验)

图 9(a)显示,各算法的平均信息损失随 QI 数量的增加而上升,其中,FAANST 算法和 FADS 算法的平均信息损失均小于 CASTLE 算法,且两者几乎完全重合,表明在其他参数不变、只改变 QI 数量时,两个算法的聚类质量相近.图 9(b)显示,线性时间复杂度的 FADS 算法的运行时间优于其他两种算法.FAANST 算法的运行时间显著少于 CASTLE 算法,这主要是由于虽然两个算法的时间复杂度均与 $|S|^2$ 成正比,但由第 3.1 节的分析可知,CASTLE 算法的时间复杂度还包含一个 $|Q||S|\delta^2/K$ 项,在 $\delta$ 较大时,当 QI 数量增加时,其运行时间的增加量要大于 FAANST 算法.

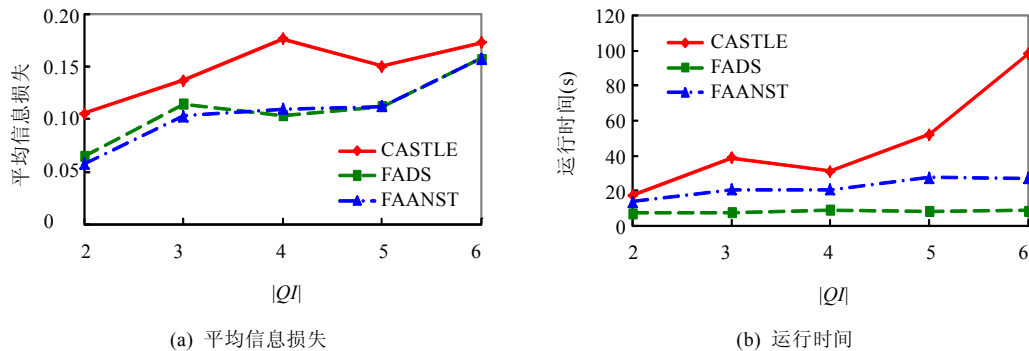


Fig.9 Average information loss and runtime with varying QI number (the 2nd experimental group)

图 9 不同 QI 数量的平均信息损失和运行时间(第 2 组实验)

将第 2 组实验结果与第 1 组实验结果对比可以发现,参与实验的算法在仅包含数值型属性数据上的平均信息损失均小于包含混合属性的数据,其需要的运行时间也更少.由第 2.2 节的定义可知,分类型属性信息损失的计算依赖于属性的 DGH,其层次划分具有间断性,由下层节点概化至上层节点可能产生较大的信息损失;而数值型属性的区间划分具有连续性,由连续区间覆盖造成的信息损失相对较小.此外,分类型属性在计算信息损失时需要遍历 DGH,而数值型属性仅需计算区间长度,因此后者的运算量较小.

我们还进一步研究了 FADS 算法的参数  $c_0$  和  $\tau$  对算法性能的影响.图 10 显示了当  $c_0$  取不同值时,信息损失和运行时间随数据量变化的情况.为使变化更加显著,实验时取参数  $\delta=1000, K=10$ ,其他参数按表 3 设置.从图 10 可以看出,当  $c_0$  的值较大时,可以得到较小的平均信息损失,但减小的幅度小于 0.01,而由此带来的运行时间的增加则比较明显.因此, $c_0=1.0$  是一个比较合适的取值.

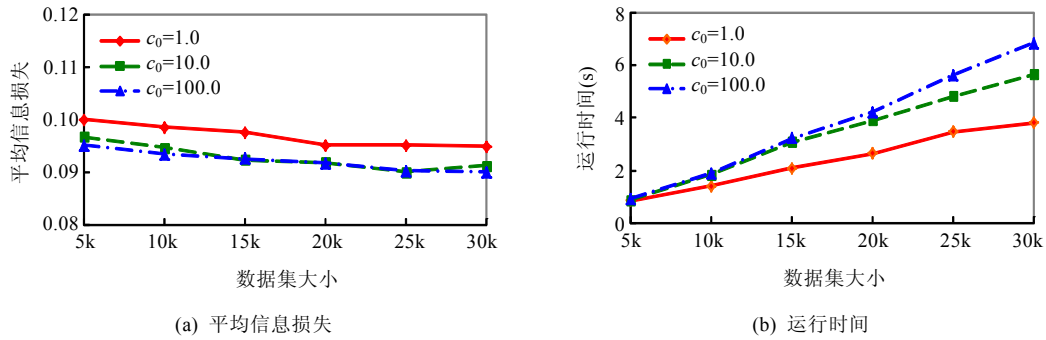
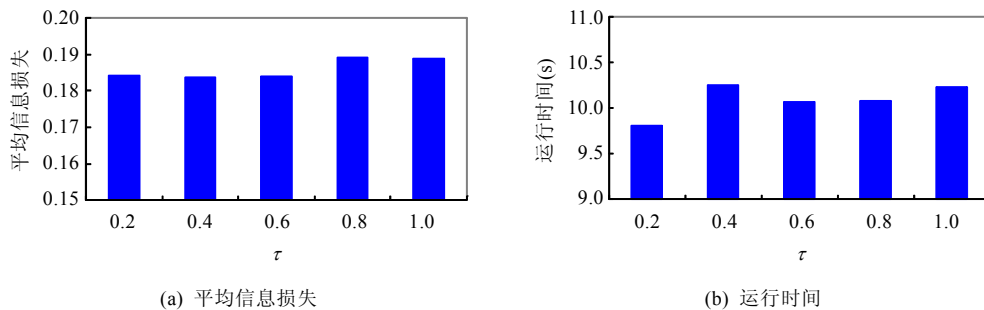
Fig.10 Average information loss and runtime with varying  $c_0$  and varying data sizes图 10 不同  $c_0$  值和数据量的平均信息损失和运行时间

图 11 显示出  $\tau$  取不同值对算法性能的影响.实验参数按表 3 设置.从图 11 可以看出,当  $\tau$  取不同值时,FADS 算法的平均信息损失均在区间  $[0.18, 0.19]$  内变化,运行时间均在  $9.7s \sim 10.3s$  之间变化.这说明  $\tau$  值的变化对算法性能影响很小,若将前面实验中的  $\tau$  值改成 0.5 以外的其他值,实验结果不会有显著变化.

Fig.11 Average information loss and runtime with varying  $\tau$ 图 11 不同  $\tau$  值的平均信息损失和运行时间

综合前述实验结果,可以得出以下结论:

- (1) 在算法性能指标与参数的关系方面,数据量与参数  $\delta$  的增大,均使算法的平均信息损失逐渐减小,而运行时间逐渐增加.CASTLE 算法和 FAANST 算法的时间复杂度与数据量和  $\delta$  的平方成正比,使其运行时间增加较明显;而 FADS 算法的线性时间复杂度使其运行时间增加较平缓.参数  $K$  和  $QI$  数量的增大,均使算法的平均信息损失逐渐增大,由于  $K$  值的大小直接影响  $K$  匿名簇集合  $Set_{wc}$  的大小,算法的平均信息损失随  $K$  值的变化均较为显著.算法的运行时间随参数  $K$  的增大而减小,随  $QI$  数量的增大而增加,但除了 CASTLE 算法以外,变化的幅度均较小.
- (2) 在算法性能对比方面,FADS 算法在平均信息损失和运行时间两个指标上均优于对比算法,并且在运行时间指标上,FADS 算法相对于 CASTLE 算法具有较为显著的优势,反映出 FADS 算法  $K$  匿名簇设计的重要作用:
  - 首先,采用单一  $K$  匿名簇集合  $Set_{wc}$  并设置其大小上限,一方面最大限度地保存已发布  $K$  匿名簇以供后续元组发布重用;另一方面使  $K$  匿名簇集合的搜索时间及存储空间不超过限定值,对于减小概化信息损失、保证算法的线性时间复杂度及固定空间复杂度具有关键作用.
  - 其次,舍弃复杂的分割与合并操作,改用基于  $k$  近邻思想的快速聚类算法一次性生成  $K$  匿名簇,一方面保证簇的大小不超过  $K$ ,从而不需要分割/合并操作;另一方面在全体元组集合  $Set_{tp}$  上聚类也

保证生成的簇具有较高质量,从而减小概化信息损失.

- 最后,在参数  $c_0$  和  $\tau$  上的实验结果表明,FADS 算法的平均信息损失受两个参数变化影响较小,因此,除了各匿名算法共有的参数  $K$  和  $QI$  数量以外,FADS 算法实际上只需要一个额外参数  $\delta$ .

## 5 结束语

针对数据流环境下的匿名保护问题,本文首先阐述了数据流匿名的基本概念和特点;然后对现有数据流匿名方法进行分析,并在此基础上提出一种新的基于聚类的数据流匿名方法,并证明其时间和空间的复杂度均为线性;最后,通过对比实验验证了提出的方法能够在快速实现数据流匿名的同时保证数据仍具有较高的可用性.今后的工作考虑引入  $l$  多样性等更多匿名准则,设计满足更高匿名要求的数据流匿名方法,并进一步提高算法的运行效率.

## References:

- [1] Wong RCW, Fu AWC, Wang K, Pei J. Anonymization-Based attacks in privacy-preserving data publishing. *ACM Trans. on Database Systems*, 2009,34(2):1–46. [doi: 10.1145%2f1538909.1538910]
- [2] LeFevre K, DeWitt DJ, Ramakrishnan R. Incognito: Efficient full-domain  $k$ -anonymity. In: *Proc. of the SIGMOD 2005*. ACM Press, 2005. 49–60. [doi: 10.1145/1066157.1066164]
- [3] Fung BCM, Yu PS. Top-Down specialization for information and privacy preservation. In: *Proc. of the ICDE 2005*. IEEE Computer Society, 2005. 205–216. [doi: 10.1109/ICDE.2005.143]
- [4] Bayardo RJ, Agrawal R. Data privacy through optimal  $k$ -anonymization. In: *Proc. of the ICDE 2005*. IEEE Computer Society, 2005. 217–228. [doi: 10.1109/ICDE.2005.42]
- [5] LeFevre K, DeWitt DJ, Ramakrishnan R. Mondrian multidimensional  $k$ -anonymity. In: *Proc. of the ICDE 2006*. IEEE Computer Society, 2006. 25–25. [doi: 10.1109/ICDE.2006.101]
- [6] Fung BCM, Wang K, Wang L, Hung PCK. Privacy-Preserving data publishing for cluster analysis. *Data & Knowledge Engineering*, 2009,68(6):552–575. [doi: 10.1016/j.datak.2008.12.001]
- [7] Wang ZH, Xu J, Wang W, Shi BL. Clustering-Based approach for data anonymization. *Ruan Jian Xue Bao/Journal of Software*, 2010,21(4):680–693 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/3508.htm> [doi: 10.3724/SP.J.1001.2010.03508]
- [8] Sweeney L.  $k$ -Anonymity: A model for protecting privacy. *Int'l Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2002,10(5):557–570. [doi: 10.1142/S0218488502001648]
- [9] Machanavajjhala A, Kifer D, Gehrke J, Venkitasubramaniam M.  $l$ -Diversity: Privacy beyond  $k$ -anonymity. *ACM Trans. on Knowledge Discovery from Data*, 2007,1(1):1–52. [doi: 10.1145/1217299.1217300]
- [10] Yang N, Tang CJ, Wang Y, Chen Y, Zheng JL. Clustering algorithm on data stream with skew distribution based on temporal density. *Ruan Jian Xue Bao/Journal of Software*, 2010,21(5):1031–1041 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/3470.htm> [doi: 10.3724/SP.J.1001.2010.03470]
- [11] Li FF, Sun JM, Papadimitriou S, Mihaila GA, Stanoi I. Hiding in the crowd: Privacy preservation on evolving streams through correlation tracking. In: *Proc. of the ICDE 2007*. IEEE Computer Society, 2007. 686–695. [doi: 10.1109/ICDE.2007.367914]
- [12] Cao JM, Carminati B, Ferrari E, Tan K. CASTLE: Continuously anonymizing data streams. *IEEE Trans. on Dependable and Secure Computing*, 2011,8(3):337–352. [doi: 10.1109/TDSC.2009.47]
- [13] Zhou B, Han Y, Pei J, Jiang B, Tao YF, Jia Y. Continuous privacy preserving publishing of data stream. In: *Proc. of the EDBT 2009*. New York: ACM Press, 2009. 648–659. [doi: 10.1145/1516360.1516435]
- [14] Li J Z, Ooi BC, Wang WP. Anonymizing streaming data for privacy protection. In: *Proc. of the ICDE 2008*. IEEE Computer Society, 2008. 1367–1369. [doi: 10.1109/ICDE.2008.4497558]
- [15] Wang WP, Li, JZ, Ai CY, Li YS. Privacy protection on sliding window of data streams. In: *Proc. of the 2007 Int'l Conf. on Collaborative Computing: Networking, Applications and Worksharing*. New York: IEEE Computer Society, 2007. 213–221. [doi: 10.1109/COLCOM.2007.4553832]



- [16] Zhang JW, Yang J, Zhang JP, Yuan YB. KIDS:  $k$ -Anonymization data stream base on sliding window. In: Proc. of 2010 the 2nd Int'l Conf. on Future Computer and Communication. IEEE Computer Society, 2010. V2-311–V2-316. [doi: 10.1109/ICFCC.2010.5497420]
- [17] Wang P, Lu JJ, Zhao L, Yang JW. B-CASTLE: An efficient publishing algorithm for  $k$ -anonymizing data streams. In: Proc. of 2010 the 2nd WRI Global Congress on Intelligent Systems. IEEE Computer Society, 2010. 132–136. [doi: 10.1109/GCIS.2010.196]
- [18] Zakerzadeh H, Osborn SL. FAANST: Fast anonymizing algorithm for numerical streaming data. In: Proc. of the 5th Int'l Workshop on Data Privacy Management and 3rd Int'l Conf. on Autonomous Spontaneous Security. Springer-Verlag, 2011. 36–50. [doi: 10.1007/978-3-642-19348-4\_4]
- [19] Meyerson A, Williams R. On the complexity of optimal  $k$ -anonymity. In: Proc. of the 23rd ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems. ACM Press, 2004. 223–228. [doi: 10.1145/1055558.1055591]
- [20] Atzori M. Weak  $k$ -anonymity: A low-distortion model for protecting privacy. In: Proc. of the Information Security Conf. 2006. Springer-Verlag, 2006. 60–71. [doi: 10.1007/11836810\_5]
- [21] Iyengar VS. Transforming data to satisfy privacy constraints. In: Proc. of the ACM KDD 2002. New York: ACM Press, 2002. 279–288. [doi: 10.1145/775047.775089]
- [22] Frank A, Asuncion A. UCI machine learning repository. Irvine: School of Information and Computer Science, University of California, 2010. <http://archive.ics.uci.edu/ml>

#### 附中文参考文献:

- [7] 王智慧,许俭,汪卫,施伯乐.一种基于聚类的数据匿名方法.软件学报,2010,21(4):680–693. <http://www.jos.org.cn/1000-9825/3508.htm> [doi: 10.3724/SP.J.1001.2010.03508]
- [10] 杨宁,唐常杰,王悦,陈瑜,郑皎凌.一种基于时态密度的倾斜分布数据流聚类算法.软件学报,2010,21(5):1031–1041. <http://www.jos.org.cn/1000-9825/3470.htm> [doi: 10.3724/SP.J.1001.2010.03470]



郭昆(1979—),男,福建福州人,博士,讲师,CCF 会员,主要研究领域为数据挖掘,灰色系统,决策支持系统,计算机软件理论.  
E-mail: [gukn@fzu.edu.cn](mailto:gukn@fzu.edu.cn)



张岐山(1962—),男,博士,教授,博士生导师,主要研究领域为灰色系统,商务智能,物流管理,系统仿真.  
E-mail: [zhangqs@fzu.edu.cn](mailto:zhangqs@fzu.edu.cn)