高维 Turnstile 型数据流聚类算法*)

周晓云[1] 张 净[1,2] 孙志挥[1]

(东南大学计算机科学与工程系 南京 210096)[1] (江苏大学电气信息工程学院 镇江 212001)[2]

摘 要 现有数据流聚类算法只能处理 Time Series 和 Cash Register 型数据流,并且应用于高维数据流时其精度不甚理想。提出针对高维 Turnstile 型数据流的子空间聚类算法 HT-Stream,算法对数据空间进行网格划分,在线动态维护网格单元信息,采用倾斜时间窗口存储统计信息,根据用户指定时间跨度离线输出聚类结果。基于真实数据集与仿真数据集的实验表明,算法具有良好的适用性和有效性。

关键词 数据流,子空间聚类,高维,倾斜时间窗口

An Efficient Clustering Algorithm for High Dimensional Turnstile Data Streams

ZHOU Xiao-Yun[1] ZHANG Jing[1,2] SUN Zhi-Hui[1]

(Department of Computer Science and Engineering, Southeast University, Nanjing 210096)[1]

(College of Electronic and Information Engineering, Jiangsu University, Zhenjiang 212001)[2]

Abstract Previous method only can deal with Time Series and Cash Register data stream. Moreover, the efficiency of clustering high dimensional data stream is not very satisfactory. In this paper a novel algorithm for clustering Turnstile data stream named HT-Stream is presented. HT-Stream partitions the space into grids, summarizes statistical information over data stream according to the tilted time window, and finds the clusters offline. HT-Stream can resolve high dimensional clustering problem and discover clusters with arbitrary shape. The experimental results on real datasets and synthetic datasets demonstrate promising availabilities of the approach.

Keywords Data stream, Subspace clustering, High dimension, Tilted time windows

1 引言

随着计算机技术的广泛应用,数据流(Data Streams)作为一类重要的数据来源,受到越来越多的关注,基于数据流模型的管理系统及其知识发现算法等已成为重要的应用前沿课题[1~3]。网络事件日志、电话呼叫记录、信用卡交易流、传感器网络等均可以看作基于数据流模型的数据集。它们具有数据量大、潜在无限、到达速率不确定等特点,若对这类数据存储后再多次扫描进行数据挖掘,其代价昂贵甚至是不可能的,且缺少实时性。因此迫切需要提出高效、可行的基于数据流模型的算法,使得在给定的有限的运行空间上,能够通过对数据流进行一次或较少次数的线性扫描,对其进行管理以及进一步的知识发现。

2 相关工作

一般而言,数据挖掘中的聚类分析既可以作为一个单独的工具用以发现数据库中数据分布的一些深入的信息,也可以作为其他数据挖掘分析算法的一个预处理步骤,在诸如市场分析、金融投资、医疗卫生等实际应用领域得到了广泛的应用。所谓数据聚类是根据数据的内在性质将数据分成一些簇,每一簇中的元素尽可能具有相同的特性,不同簇之间的特性差别尽可能大。传统的聚类算法往往需要多遍扫描数据集合,均不能直接应用到数据流上。

目前,基于数据流的算法研究已经成为重要的研究课题。S. Muthukrishnan 在文[2]中指出数据流主要包括 Time Series, Cash Register, Turnstile 三种类型,其一般性逐渐增大。现有的聚类算法只能处理 Time Series, Cash Register 型数据流,针对 Turnstile 型数据流的聚类算法国内外尚未见报道。

Time Series 型数据流比较简单,聚类可以分为子序列聚类(Subsequences Clusteing)和整体聚类(Whole Clustering)。E. Keogh 等在文[4]中指出时间序列子序列聚类是没有意义的,而整体聚类算法研究已相对比较成熟,如基于 k-means 的 Refinement[5]算法,基于小波的 I-Keans 算法[6],基于层次的 ODAC 算法[7]等。

近年来,在 Cash Register 型数据流上的聚类算法研究获得了重要的成果,已发表了一些有效算法, S. Guha 等提出 LOCALSEARCH 算法[8],该算法基于分治的思想使用一个不断迭代过程实现以有限空间对数据流进行 k-means 聚类。L. O' Callaghan 等在 LOCALSEARCH 基础上又提出了 STREAM 算法[9],并使用 SSQ 方法证明 STREAM 算法的聚类效果更好。C. Aggarwal 等提出一个聚类演化的数据流框架 CLUSTREAM[10]。在这个框架中,将数据流的聚类分成在线微聚类和离线宏聚类两个阶段,实现近期数据的聚类和用户指定时间段聚类。以上几种流数据聚类方法都使用 k-means 聚类思想,对非凸形状聚类效果不好,同时无法对高维数据进行聚类。孙焕良等提出的 CD-STREAM[11]算法,使用

*)本文得到国家自然科学基金(70371015)、教育部高等学校博士学科点科研基金(20040286009)、江苏省高校自然科学计划一般项目(05KJB520022)资助。**周晓云** 博士生,主要研究领域为数据挖掘与知识发现。**张 净** 博士生、讲师,主要研究领域为数据挖掘和知识发现。**孙志辉** 教授,博士生导师,主要研究领域为复杂系统集成、数据库、知识发现、数据挖掘。

基于空间划分的聚类方法对流数据进行聚类，对非凸形状聚类效果较好，但由于是在整个数据空间进行聚类，同样无法处理高维数据流聚类问题。C. Aggarwal 等在文[12]中提出 HPStream 算法，采用投影技术解决数据流的高维聚类问题，但要求用户输入参数平均聚类维数，在实际应用中往往比较困难，同时也没有解决对非凸形状聚类的问题。

本文的研究工作主要包括以下方面。针对 Turnstile 型数据流，提出高维数据流聚类算法 HT-Stream。该算法对数据空间进行网格划分，采用倾斜时间窗口存储数据流数据，利用在线网格信息统计与离线聚类相结合对流数据进行聚类分析，在着重解决数据流高维问题的同时实现对任意形状分布数据的聚类。基于真实数据集与仿真数据集的实验表明，算法是可行有效的。

3 问题描述及相关定义

假设 $A=\{A_1,A_2,\cdots,A_k\}$ 是一个有界、全部有序域的集合，$S=\{A_1\times A_2\times\cdots\times A_k\}$ 是一个 k 维数值空间，其中 A_1，$A_2,\cdots,A_k$ 表示 S 的 k 个维。k 维数据流 $X=\langle x_1,x_2,\cdots,x_m\rangle$ 表示 S 上的数据流 t 时刻的一个点集，其中 $x_i=\langle x_{i1},x_{i2},\cdots,x_{ik},e_i\rangle$ 表示一个数据点，x_{ij} 为数据点 x_i 的第 j 维的值，e_i 表示 x_i 的值，可以是正数也可以是负数。

定义 1(网格单元) 将流数据各维平均分割为 ξ 个区间，S 分割为 ξ^k 个不相交的超立方体，则每个超立方体称为一个网格单元，其中每个网格单元 u 的空间位置表示为 $\{v_1,v_2,\cdots,v_k\}$，$v_i=[l,h_i)$ 对应于第 i 个维上的一个右开的间隔段。

对于数据流 X 上的数据点 $x=\langle x_1,x_2,\cdots,x_k,e\rangle$ 及单元 $u=\{v_1,v_2,\cdots,v_k\}$，如果对所有的 v_i 都满足 $l_i\leqslant x_i<h_i$，则 x 属于 u，记为 $x\in u$。

定义 2(密度) 给定一时刻点 t 和时间跨度 h，$\{x_r,x_{r+1},\cdots,x_{r+s}\}$ 为 k 维数据流 X 在时间段 $(t-h,t]$ 内的点集，则网格单元 u 在 $(t-h,t]$ 时间段内的密度 $density(u)=\dfrac{count(u)}{N}$。其中 $N=\sum_{i=0}^{s}x_{r+i}.e$，$count(u)=\sum_{i=0}^{s}x_{r+i}.e(x_i\in u)$。

定义 3(密集网格单元) 给定一时刻点 t 和时间跨度 h，若网格单元 u 的密度 $density(u)\geqslant\tau$，则称网格单元 u 在时间段 $(t-h,t]$ 是密集的，其中 τ 为密度阈值。

定义 4(子空间) 数值空间 $Sub=A_{i1}\times A_{i2}\times\cdots\times A_{in}$，其中 $n<k$，并且当 $i<j$ 时，$t_i<t_j$ 成立，则称 Sub 为 S 的一个子空间。

与此类似，可以在子空间中定义网格单元、密度和密集网格单元等相应概念。

定义 5(聚类) 给定一时刻点 t 和时间跨度 h，k 维数据流 X 在时间段 $(t-h,t]$ 内的一个子空间聚类定义为在时间段 $(t-h,t]$ 内 $i(1\leqslant i\leqslant k)$ 维子空间中相连密集网格单元的最大集合。两个 i 维网格单元 u_1 和 u_2 相连的条件是它们之间有一个公共面，或者存在另一个 i 维网格单元 u_3，使得 u_1 与 u_3 相连，u_2 与 u_3 相连。

4 HT-Stream 算法及其构造

4.1 主要问题及解决思路

当数据维数很高时，数据的分布规律经常呈现出反常的特征，如噪声水平提高，数据密度稀疏，常规意义下的距离定义不再有效等，这种高维反常现象在数据挖掘中被称为“维数灾难(Curse of dimensionality)”。子空间聚类是解决高维数据聚类问题的有效方法之一，在静态数据集聚类中已得到广泛的研究，并已有许多成熟的算法[13]。HT-Stream 算法采用子空间聚类思想，按照自底向上的搜索策略，在子空间中进行数据流聚类。自底向上搜索策略的依据是若一个子空间网格单元是密集的，那么它的所有低维投影网格单元都是密集的；反之，若一个子空间网格单元不是密集的，那么它的所有高维扩展网格单元都不是密集的。

基于网格和密度的技术可以实现对任意形状分布的聚类，并且可以过滤噪声，但是存在以下两个问题：(1)对数据空间进行网格划分，网格单元的数目与数据维数呈指数级增长；(2)网格划分的粒度越细，聚类的精度越高，但同时产生的网格单元也越多。传统的基于网格和密度的聚类算法由于需要保留所有网格单元的密度信息，无法将其保存在有限内存中，若保存至硬盘，必将影响算法的响应速度，因此无法应用于数据流。HT-Stream 采用近似的方法对低维子空间网格进行统计信息的记录，再利用自底向上的搜索策略发现高维子空间密集网格单元，由此可显著减少需要保存信息的规模，并可在内存中进行处理。

4.2 网格单元信息的保存

HT-Stream 算法采用 Bloom Filter[14] 对低维子空间网格单元密度近似估计。Bloom Filter 的最大特点是，仅使用一小块远小于数据集数据范围的内存空间表示数据集，并且各个数据仍然能被区分开来。假设所申请的内存大小为 m 比特位，该方法创建 g 个相互独立的哈希函数，能将数据集均匀映射到 $[1...m]$ 中去。对任何元素，利用哈希函数进行计算，得到 g 个 $[1...m]$ 之间的数，并将内存空间中这 g 个对应比特位都置为 1。这样，就可以通过检查一个元素经过 g 次哈希操作后，是否所有对应的比特位都被置为 1 来判断该元素是否存在。这种判断方法可能产生假正(False Positive)—虽然某元素并不存在，但是它所对应的 g 个比特位都已经被其他元素所设置了，从而导致被误认为存在。然而假正发生的概率随着内存的增加可以非常小。SBF[15] 对其进行了改进，在每一个位置上都用计数器代替比特位，从而不仅能够判断元素是否存在，而且能够估算元素的值，同时 SBF 具有可以进行加减操作的性质。

HT-Stream 算法在具体实现时将低维子空间网格单元进行编码，映射成 $[1...M]$ 的整数，在内存中保存 m 个元组，每个元组结构为 $(id,count)$，利用 g 个哈希函数将 $[1...M]$ 映射到 $[1...M]$。这样，在较高正确概率的情况下得到数据流的低维网格单元的密度估计，根据估计可以得到低维子空间密集网格单元，再根据自底向上搜索策略，获得更高维子空间网格单元的密度估计。具体将多少维以下的子空间网格单元进行编码将根据内存大小和精度需要由用户提供维数阈值参数 l 设定。设 k 维数据流 DS 每维分割成 ξ 个间隔，维数阈值为 l，则 $M=\sum_{i=1}^{l}C_k^i\xi^i$。

HT-Stream 算法在线保存一个数据结构 $GriSet$，包含三部分：低维网格单元估计 BF(Bloom Filter)，高维候选密集网格单元集 HCDSET 和密集网格单元集 $DSET$。其中 $DSET=\bigcup_{r=1}^{k}D_r$，$D_r$ 表示 $r(1\leqslant r\leqslant k)$ 维子空间密集网格单元集。BF 保存的元组结构为 $(id,count)$，$HCDSET$、$DSET$ 保存的元组结构为 $(item\_id,count,level)$，其中 $item\_id$ 为网格单元的编码，$count$ 为网格单元的数据总量，$level$ 为网格单元的维数。算法实现时采用 TreeHash[16] 对元组进行存储，搜索和更新操作。

数据结构 *GridSet* 具有可以进行加减操作的性质，因此 HT-Stream 可以有效处理 Turnstile 型数据流。

4.3 HT-Stream 的构造

4.3.1 倾斜时间窗口

HT-Stream 采用倾斜时间窗口存储数据信息，它的基本思想是以小粒度存储距当前较近的数据，以大粒度存储距当前较远的数据，这种方式实现了精度与空间复杂度的折中(trade-off)。实际应用中可选择自然倾斜时间窗口(Natural Tilted-Time Window)或对数倾斜时间窗口(Logarithmic Tilted-Time Window)等。HT-Stream 采用自然倾斜时间窗口(如图 1 所示)，每一个倾斜时间窗口保存一个 *GridSet* 数据结构，这样为了保存一年的数据流信息，HT-Stream 只需存储 4+24+31+12=71 个 *GridSet* 数据结构。有关倾斜时间窗口更详细的介绍见文[11,17]。

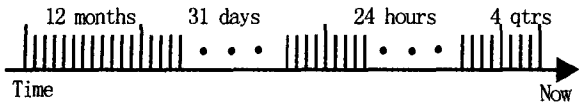

图 1 自然倾斜时间窗口

4.3.2 在线信息统计

HT-Stream 将各个时间窗口内的流数据存储到一系列的 *GridSet* 数据结构中，其中一个时间粒度单位内的流数据对应一个 *GridSet*。对于一个时间窗口内到达的每一个数据，*GridSet* 进行一次更新操作，具体算法描述如下：

```
算法 GridSet_Update
输入 k 维数据流 DS；维数阈值 l；哈希函数 h1,h2,…,hg；密度阈值 τ；划分间隔 ξ；
步骤
    Let GridSet be empty;
    N=0;//N 为数据总量
    For each datum x arrived in DS {
      N=N+x.e;
    For all f-subunit(1≤f≤l)s of u(x∈u) {
      id=convert(s,ξ,l);
      For (i=1;i≤g;i++)
      VB[hi(id)].count=VB[hi(id)].count+x.e;
    //VB[i]表示获得 BF 第 i 个位置上的元组
      If (s not in Df) {
        If (all VB[hi(id)].count≥τ×N) {
          insert s into Df;
    DB[s].count=min(VB[hi(id)].count);
    //DB[s]表示获得 s 在 DSET 中的元组
        }
    } Else if (all VB[hi(id)].count≥τ×N)
        DB[s].count=DB[s].count+x.e;
        Else
     delete s and all superunit(s) from DSET;
    //superset(s)表示 s 所有高维扩展网格单元
    }
    j=l+1;
    While (Dj≠φ) and (j≤k) do {
     For all j-subunit s of u(x∈u){
      If (s in HCDSET) {
        CDB[s].count=CDB[s].count+x.e;
       If (CDB[s].count=0)
       delete s from HCDSET;
    //CDB[s]表示获得 s 在 HCDSET 中的元组
          } Else if
          (all (j-1)-subunit of s in HCDSET){
          insert s into HCDSET;
              CDB[s].count=x.e;}
          If (s not in Dj) {
            If (CDB[s].count≥τ×N) {
              insert s into Dj;
               DB[s].count=CDB[s].count;}
          } Else If (CDB[s].count≥τ×N){
              DB[s].count=DB[s]+x.e;
            } Else
    delete s and all superunit(s) from DSET;
        }
    j=j+1;
     }
  }
```

定理 1 数据结构 *GridSet* 出现假正(False Positive)的概率 $P\approx(1-e^{-g(\sum_{i=1}^{l}C_k^i\xi^i)/m})^g$，其中 l 为维数阈值，ξ 为划分间隔，m 为申请的内存大小，g 为哈希函数个数。

证明：在 *GridSet* 中，对于维数大于维数阈值 l 的子空间网格单元，其候选集是根据低维网格单元产生的，所以假正误差也由低维网格单元决定。因此，只要考察低维(维数不大于 l)网格单元的误差。由文[15]可知，SBF 出现假正概率 $P\approx(1-e^{-gM/m})^g$。在 *GridSet* 中 $M=\sum_{i=1}^{l}C_k^i\xi^i$，所以出现假正的概率 $P\approx(1-e^{-g(\sum_{i=1}^{l}C_k^i\xi^i)/m})^g$，定理得证。

HT-Stream 能够提供出现假正的概率保证，进而有效保证聚类结果的精度。由定理 1 可知在实际运用中，出现假正的概率是很低的，并且和内存大小 m、哈希函数个数 g 成反比，与维数阈值 l、每维间隔数 ξ 成正比。

4.3.3 离线聚类

当用户需要知道近期一段时间内的数据流聚类情况，可以采用指定时间跨度聚类。设 T_c 为当前时刻，用户指定的 T_c 以前的时间跨度为 h，算法读取从 t 往前 h 时间段内的 *GridSet* 序列进行合并后聚类。当 h 在一个 *GridSet* 的中间位置时，根据时间跨度 h 离这个 *GridSet* 开始与终止时刻点的远近选择是否将该 *GridSet* 进行合并。

定理 2 若 u 为时间段 $[T_c-h,T_c]$ 内的子空间密集网格单元，则 u 至少在一个 $GridSet_{p+i}$ $(1\leqslant i\leqslant q)$ 上是子空间密集单元，其中 $[T_c-h,T_c]$ 内的网格单元统计信息序列为 $GridSet_{p+1}$，$GridSet_{p+2}$，…，$GridSet_{p+q}$。

证明：用反证法，设 u 在任何一个 $GridSet_{p+i}$ $(1\leqslant i\leqslant q)$ 上都不是密集的，即 $density(u)^{p+1}<\tau$，$count(u)^{p+i}<\tau\times N^{p+i}$，则 $\sum_{j=1}^{q}count(u)^{p+i}<\tau\times\sum_{j=1}^{q}N^{p+i}$，即 $density(u)^{[T_c-h,T_c]}<\tau$，$u$ 不是 $[T_c-h,T_c]$ 内的子空间密集网格单元。与题设矛盾，定理得证。

由定理 2，合并算法将 $[T_c-h,T_c]$ 时间段内 *GridSet* 的各个相应网格单元进行相加，再考察合并后 *GridSet* 中的 *DSET*，将不满足密集条件的网格单元删除。*GridSet* 合并算法具体描述如下：

```
算法 GridSet_Merge
输入 [Tc-h,Tc]时间段内的网格单元统计信息序列 GridSetp+1，GridSetp+2，GridSetp+q 及相应数据总量 Np+1，Np+2，…，Np+q；维数阈值 l；哈希函数 h1,h2,…,hg；密度阈值 τ；划分间隔 ξ。
输出 [Tc-h,Tc]时间段内网格单元统计信息 ΔGridSet。
步骤
    Let ΔGridSet be empty;
    For (i=0;i<m;i++)
    For (j=1;j≤q;j++)
    VB[i]Δ.count=VB[i]Δ.count+VB[i]p+j.count;
    For (j=1;j≤q;j++)
      For every s in HCDSETp+j
        If s not in ΔHCDSET{
        Insert s into ΔHCDSET;
        CDB[s]Δ.count=CDB[s]p+j.count;
        } Else
CDB[s]Δ.count=CDB[s]Δ.count+CDB[s]p+j.count;
      For (j=1;j≤q;j++)
        For every s in DSETp+j
          If s not in ΔDSET{
          Insert s into ΔDSET;
           DB[s]Δ.count=DB[s]p+j.count;
          }Else
  DB[s]Δ.count=DB[s]Δ.count+DB[s]p+j.count;
    f=1;
    While (f≤l) do {
      For all f-unit s in ΔDf{
        id=convert(s,ξ,l);
      If
    (exit VB[hi(id)]Δ.count<τ×∑j=1q Np+j))
      delete s and superunit(s) form ΔDSET;
       }
       f=f+1;
```

```
}
While (f≤k) do{
  For all f-unit s in ΔDf{
  If (CDB[s]Δ.count<τ×(∑_{j=1}^{q} N_{p+j}))
  delete s and superunit(s) form ΔDSET;
   }
   f=f+1;
}
```

连接 $\Delta GridSet$ 中相连密集网格单元集，以 DNF 范式输出结果，具体算法见文[13]。

5 算法性能分析与实验结果

5.1 复杂度分析

本文构造的高维数据流聚类算法 HT-Stream 具有良好的空间与时间效率。

在算法运行过程中，在线信息统计只需维护 1 个 *GridSet* 数据结构。*GridSet* 空间复杂度包括三个部分：*BF* 所需内存空间 $O(m)$，高维(维数大于 l)候选密集网格单元所需内存 $O(|HSCDSET|)$和密集网格单元所需存储空间 $O(|DSET|)$。在传统子空间聚类算法中，低维候选密集网格单元需要大量的存储空间，随着维数的增加，高维候选密集网格数量显著减少。HT-Stream 用 $O(m)$空间对低维候选密集网格单元进行估计，并且用户可以根据内存大小和精度需要设定维数阈值参数 l，因此 $O(m+|HCDSET|)$可以控制在有限的内存范围之内。所以 HT-Stream 可以在有限内存空间中实现。下节的实验结果进一步说明了本文中算法所需的内存较少。

定理 3 HT-Stream 算法具有相对于数据流总量 R 线性规模的时间复杂度。

证明：在最坏情况下动态更新一次 *GridSet* 时间复杂度为 $O(2^k)$，因此算法对于总数据量 R 更新的时间复杂度为 $O(2^kR)$，具有线性复杂度，定理得证。

在实际情况下，高维密集网格单元的数量很少，并且真正存在的密集网格单元的最高维数远小于 k。因此 HT-Stream 算法的计算量远小于最坏情况的时间估计。

5.2 实验结果

本节对所提出的 HT-Stream 算法进行性能测试。实验平台配置如下：Intel 1.8G/512MB，Windows2000(Server 版)，所用代码均用 Visual C++(6.0)实现。实验所使用的数据共有 2 种，第一种是网络入侵检测数据集 KDD-CUP-99，该数据集中的数据对象分为五大类，包括正常的连接、各种入侵和攻击等。第二种是仿真数据集，用户可以通过输入参数来控制产生数据集的结构与大小，参数包括数据集的大小、维数、聚类个数和各维上的取值范围等。文中将用如下记号表示仿真数据集：'B'、'C'、'D'、'S'分别表示数据集大小、所含聚类个数、数据空间维数、所含聚类的子空间的维数。例如 B100kC10D50S10 表示大小为 100k，空间维数为 50，在不同的 10 维子空间中存在 10 个聚类的数据集。

为了测试算法的精度，我们对比 HT-Stream 和 HPStream，采用真实数据集 KDD-CUP-99 和仿真数据集 B500kC4D30S6，KDD-CUP-99 选取其 34 维连续属性进行网格划分。HPStream 算法是处理高维数据集的，具有一定的可比性。实验中将聚类精度定义为该数据集中数据点聚类正确的比例。图 2、图 3 分别显示了在真实数据集和仿真数据集下 HT-Stream 与 HPStream 的精度比较。由于 HT-Stream 算法采用了网格技术，对任意形状的分布聚类效果较好，因此 HT-Stream 算法相比 HPStream 具有更高的精度。实验中 $l=5,\tau=0.005,\xi=15$。

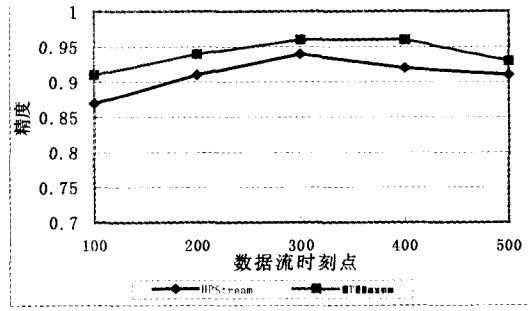

图 2 精度比较(流速 400 元组/s，KDD-CUP-99)

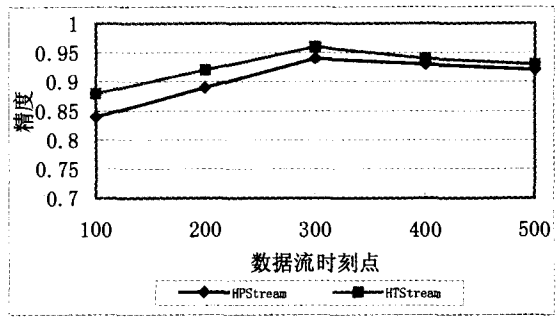

图 3 精度比较(流速 500 元组/s，仿真数据集)

以下实验中，在产生的仿真数据集聚类区域中加入少量带特殊标记的点，表示为负值数据点模拟 Turnstile 型数据流，用以验证 HT-Stream 算法对 Turnstile 型数据流的适用性。

为测试算法的内存使用情况，分别生成数据空间维数为 10、15、20、25 和 30 的仿真数据集 B1000kC4S6，算法运行稳定时，使用的内存如图 4 所示。可以看到，算法所需要的内存保持在一个较小的范围内。其内存使用对数据维数具有良好的伸缩性。支持度阈值越大，需要动态维护的密集网格单元越少，所用内存也越少。实验中 $l=5,\xi=15$。

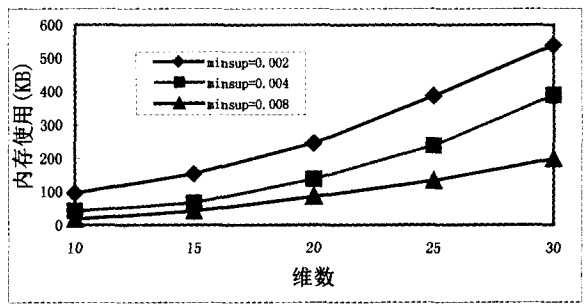

图 4 不同维数下的内存使用(流速 1000 元组/s)

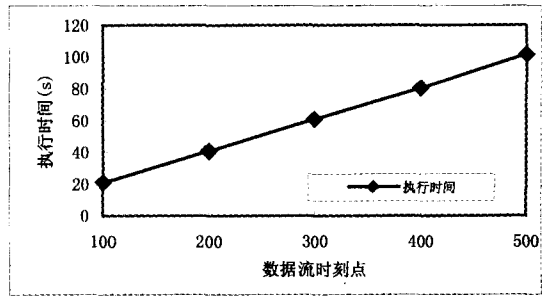

图 5 对数据流时间跨度的伸缩性(流速 1000 元组/s)

(下转第 37 页)

法解决冲突问题的效能。最后通过仿真试验证明该算法是有效的、可行的，可以适应分布式的网络环境。下一步的工作是解决如何在真实的应用环境下使用该算法，譬如在多点安全检测技术里，算法如何与 NAT 相结合等等。

参 考 文 献

1 Lahey K. RFC2923: TCP Problems with Path MTU Discovery [EB/OL]. dotRocket, Inc, 2003

2 林绍太，张会汀，郑力明. IP 分片重组算法(RFC815)的实现及其改进[J]. 计算机工程与设计，2005，26(4)：911～913

3 Luckie M, Cho K, Owens B. Inferring and Debugging Path MTU Discovery Failures [C]. Internet Measurement Conference 2005, Berkeley, CA, 2005

4 Medina A, Allman M, Floyd S. Measuring the Evolution of Transport Protocols in the internet [J]. ACM Computer Communications Review, 2005, 35 (2): 37～52

5 Pearce J P, Maheswaran R T, Tambe M. Local Algorithms for Distributed Constraint Optimization in Dynamic, Anytime Environments [C]. Autonomous Agents and Multiagent Systems 2005, Utrecht, the Netherlands, 2005

6 刘昌华，左爱群. Check Point FireWall - 1 防火墙的特性及应用[J]. 武汉工业学院学报，2004，23(2)：10～13

7 Decwrl J M, Deering S. RFC1191 : Path MTU Discovery [EB/OL]. Stanford University, 1990

8 McCann J, Deering S, Mogul J. RFC1981 : Path MTU Discovery for IP version 6 [EB/OL]. Digital Equipment Corporation, 1996

9 郑卫斌，段中兴，高磊，等. 基于 IP 流本地性的状态检测性能优化方法[J]. 西安交通大学学报，2004，38(4)：413～416

(上接第 17 页)

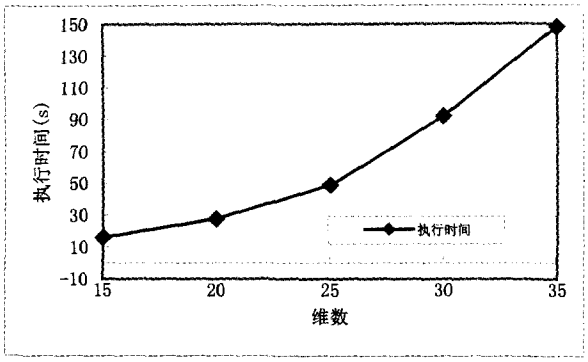

图 6 对数据流维数的伸缩性

为测试算法对数据流时间跨度的伸缩性，生成 B500kC4D30S6 的数据集，如图 5 所示，算法执行时间与数据集大小成线性变化。为测试算法对数据集维数的伸缩性，生成 B500kC4S6 维数为 15、20、25、30 和 35 维仿真数据集。如图 6 所示，算法对数据维数具有较好的伸缩性。实验中 $l=5$，$\xi=15$。

结论 本文研究了 Turnstile 型数据流上的聚类问题，提出高维数据流聚类算法 HT-Stream。该算法采用子空间聚类思想，将数据空间进行网格划分，在一定内存、时间限制下动态估计网格单元统计信息并以倾斜时间窗口进行存储，在用户提出查询时，离线输出聚类结果。HT-Stream 能够提供出现假正的概率保证进而提高聚类的精度，在着重解决数据流高维问题的同时，实现对任意形状分布数据的聚类。实验结果表明，本文提出的算法是可行有效的，进一步提高了算法的实现效率和精度及对高速数据的适应性，并将其扩展到分布式环境是下一步的研究内容。

参 考 文 献

1 Babcock B, Babu S, Datar M, et al. Models and Issues in Data Stream Systems. In: Proceedings of the 21st ACM Symposium on Principles of Database Systems, 2002. 1～16

2 Muthukrishnan S. Data streams algorithms and applications. In: Proc of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 2003. 413～413

3 金澈清，钱卫宁，周傲英. 流数据分析与管理综述. 软件学报，2004，15(8)：1172～1181

4 Keogh E, Lin J, Truppel W. Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research. In: Proceedings of the IEE International Conference on Data Mining. IEEE Computer Society Press, 2003. 115～122

5 Bradley P S, Fayyad U M. Refining Initial Points for K-Means Clustering. In: Proceedings of 15th International Conference on Machine Learning. Morgan Kaufmann, 1998. 91～98

6 Vlachos M, Lin J, Keogh E, et al. A Wavelet-Based Anytime Algorithm for K-Means Clustering of Time Series. Workshop on Clustering High Dimensionality Data and Its Applications, at the 3 SIAM International Conference On Data Mining. San Francisco, CA, 2003

7 Rodrigues P, Gama J, Pedroso J P. Hierarchical Time-Series Clustering for Data Streams

8 Guha S, Mishra N, Motwani R, et al. Clustering Data Streams. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science. Washington, DC: IEEE Computer Society, 2000. 359～366

9 O'Callaghan L, Mishra N, Meyerson A, et al. Motwani. Streaming-Data Algorithms for High-Quality Clustering. In: Proceedings of the 18th International Conference on Data Engineering. Washington, DC: IEEE Computer Society, 2002. 685～704

10 Aggarwal C, Han J, Wang J, et al. A Framework for Clustering Evolving Data Streams. In: Proceedings of the 29th International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc, 2003. 81～92

11 孙焕良，赵法信，鲍玉斌，等. CD-Stream——一种基于空间划分的流数据密度聚类算法. 计算机研究与发展，2004，41(Suppl)：289～294

12 Aggarwal C, Han J, Wang J, et al. A Framework for Projected Clustering of High Dimensional Data Streams. In: Proceedings of the 30th International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc, 2004. 852～863

13 Agrawal R, Gehrke J, Gunopulos D, et al. Automatic Subspace Clustering of High Dimensional Data for Data Mining Application. In: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1994. 94～105

14 Bloom B. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, 1970, 13(7): 422～426

15 Cohen S, Matias Y. Spectral Bloom Filters. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. San Diego: ACM Press, 2003. 241～252

16 Jin R, Agrawal G. An Algorithm for In-Core Frequent Itemset Mining on Streaming Data. In: Proceedings of the 5th IEEE International Conference on Data Mining. IEEE Computer Society, 2005. 210～217

17 Giannella C, Han J, Pei J, et al. Mining Frequent Patterns in Data Streams at Multiple Time. Next Generation Data Mining, 2003. 97～106