一种有效的增量聚类算法*)

许毕峰 冯少荣 薛永生 刘笑锋 翁 伟
(厦门大学计算机科学系 厦门361005)

摘 要 聚类是数据挖掘领域中最活跃的研究分支之一,聚类技术在其他的科学领域也有广泛的应用。迄今为止已经提出了大量的聚类算法,其中基于密度的DBSCAN算法因其很多优点而备受关注,为了减少DBSCAN的区域查询次数,降低I/O开销而提出的改进算法有FDBSCAN、LSNCCP等。随着应用的发展,增量聚类显得越来越重要,而现有的增量聚类算法存在很大的局限性。基于LSNCCP,提出了一种有效的增量聚类算法,同时它也可以用于对LSNCCP进行性能优化。

关键词 数据挖掘,聚类,密度,最大核心点集,增量聚类算法

An Effective Incremetal Clustering Algorithm

XU Bi-Feng FENG Shao-Rong XUE Yong-Sheng LIU Xiao-Feng WENG Wei
(Department of Computer Sciences, Xiamen University, Xiamen 361005)

Abstract Clustering is one of the most flourish direction of data mining. It has been applied abroad at other scientific fields. Many clustering algorithms have been proposed so far, and the DBSCAN algorithm which was density-based was famous for it's advantages. In order to decrease the amount of regional queries and operations of I/O, some people suggested some advanced algorithms such as FDBSCAN, LSNCCP. With the development of application, incremental clustering algorithm became more important, while the incremental clustering algorithms have been suggested have a lot of limitation. Based on LSNCCP, we propose a new effective incremental clustering algorithm called INCCP, which can be used to improve the efficiency of LSNCCP too.

Keywords Data mining, Clustering, Density, LSNCCP, Incremental clustering algorithm

1 概述

聚类是数据挖掘领域中最活跃的研究分支之一,聚类技术在统计数据分析、模式识别、图像处理等其他许多科学领域有广泛的应用。所谓聚类就是将数据对象分组成为多个类或簇(cluster)。在过去的十几年中,人们提出了很多聚类算法[1~9],其中DBSCAN[1]算法因为其可以发现任意形状的聚类结果,而且可以排除噪声的干扰等优点而备受关注。为了减少DBSCAN的区域查询次数,降低I/O开销而提出的改进算法有FDBSCAN[2]、LSNCCP[3]等。在越来越多的应用中,必须对收集来的大量的数据进行聚类。在有些应用中,由于数据规模太大,不能把整个数据集储存在主存储器里。现行的聚类算法为了遍历这样的整个数据集合,不得不进行频繁的I/O操作。另外,由于现在用来聚类的数据源丰富多样,有些数据源象数据仓库,它更新的周期长,而且每次更新的数据量很大。这些都使增量聚类算法越来越重要,而现已提出的增量聚类算法都存在很大的局限性。基于LSNCCP,提出了一种有效的增量聚类算法,同时它也可以用于对LSNCCP进行性能优化。

本文第2节介绍基于密度聚类算法的相关的工作,第3节给出LSNCCP算法思想,第4节给出增量算法INCCP,第5节从理论上分析算法的时间复杂度,并给出实验的测试结果,最后是总结。

2 相关工作

基于高密度连接区域密度聚类算法DBSCAN[1]利用类的密度连通特性,可以在带有"噪声"的空间数据库中发现任意形状的类。DBSCAN通过检查所给数据集合当中每个点的Eps邻域来寻找聚类。首先它从第一个点开始,如果它是关于Eps和MinPts的核心点就创建一个新的类,然后DBSCAN反复从这些核心点寻找直接密度可达的点,这个过程可能会涉及一些密度可达类的合并,直到没有新的点可以归入此类。如果第一个点不是核心点,就先把它暂时标志成为噪声点,然后跳到下一个点。对于下一个点,如果它是核心点而且没有被分入某一类,那就像对待第一个点一样创建一个新的类,并把那些从它可以密度可达的点全部加入这一类中。这个过程直到没有新的点可以添加到任何类的时候结束。

如果能够降低区域查询的次数,我们就可以改进DBSCAN的时间复杂度。对于密集的类来说,在一个核心点的邻域中有相当多的点可以不用作为类扩展用的核心点,因为一个点的Eps邻域的点通常是被其他的点的Eps邻域所覆盖。为了减少DBSCAN的时间复杂度,应该选择一个点的Eps邻域内的部分点来做进一步扩展的核心点,FDBSCAN[2]算法正是基于这样的想法提出的。

*)本课题得到福建省自然基金资助(A0310008)、福建省高新技术研究开放计划重点项目资助(2003H043)。许毕峰 硕士研究生,主要研究方向为数据仓库、数据挖掘等。冯少荣 副教授,硕士生导师,现主要研究方向为XML与半结构化数据库系统,数据挖掘与KDD,数据仓库与OLAP技术,基于Web的数据库技术,多数据库信息集成技术。薛永生 教授,主要研究方向为数据库理论与应用、分布式数据库、数据仓库、数据挖掘、网络技术、计算机应用技术等。刘笑锋 硕士研究生,主要研究方向为数据仓库、数据挖掘等。翁 伟 硕士研究生,主要研究方向为分布式数据库、数据仓库、数据挖掘等。

文[4]给出了一种基于 DBSCAN 的增量聚类算法，但是这个算法对数据的更新过于敏感，对于任何一个数据的插入和删除操作都会可能引起一个类的合并或分解，而且它每次都只能孤立地处理一个更新，存在太多的冗余操作。文[5]给出另外一个基于密度的增量算法 IGDCA，它存在两个缺陷：在划分网格的时候是基于对要聚类数据空间进行划分，但是如果这个范围所选过大，则要聚类的网格数量非常可观，如果所选范围太小，很有可能出现更新的数据超出所选范围；在进行增量更新时，如果是删除操作，那 IGDCA 只是调用它的非增量算法进行处理，如果是插入操作，在有大数据量进行更新时，跟非增量算法比较起来，所获得的收益也很有限。

3 LSNCCP 算法思想

基于跟 FDBSCAN 类似的想法，文[3]提出另外一种减少 DBSACN 区域查询的聚类算法 LSNCCP。与 FDBSCAN 不同的是，它需要做区域查询的核心点的个数更好。

在 LSNCCP 算法中定义了核心点之间的三类不同的关系：

定义1 如果两个核心点 P,Q 之间的距离 $Dist(P,Q)>2*Eps$，则称 P,Q 两个核心点之间是相离的。

定义2 如果两个核心点 P,Q 之间的距离 $Eps<Dist(P,Q)<=2*Eps$，则称 P,Q 是相交的。

定义3 如果两个核心点 P,Q 之间的距离 $Dist(P,Q)<=Eps$，则称 P,Q 是相含的。

首先，LSNCCP 算法找出一个给定集合中最大不相含的核心点的集合(Largest Set of Not-Covered Core Points)，这需要遍历每个点，看看它跟已经找到的集合中的每个点的距离是否大于 Eps，如果是就把它加入到集合中去。然后在最大核心点集中每个核心点上做区域查询，对非核心点进行分类，同时删除核心点集合当中的非核心点。最后把所有的点映射到每个类当中并处理丢失点，没有分类标志的点就是噪声点。LSNCCP 没有丢失点，算法时间复杂度后文跟增量算法一起分析。

4 增量算法 INCCP

假定我们已经得到了一个由 LSNCCP 算法进行聚类的集合 D，数据更新发生时候，我们的聚类受到哪些影响，只考虑删除和插入两种情况。

当从数据集合当中删除一些点以后，如果删除是已经被聚类的点，则被选在最大核心点集合当中的一些核心点例如 P 很可能就不再是核心点了，它必须从最大核心点集合当中删除，而原来点 P 的 Eps 邻域内那些被聚类的点也不再被聚类。但由于 P 的 Eps 邻域内其他点有可能是核心点，我们还必须在最大核心点集合中删除一些核心点以后再对最大核心点集合进行扩充。如果删除本身就是噪声点，那么对聚类结果没有影响。

考虑插入数据的情况，如果插入的数据在核心点集合中某个核心点的 Eps 邻域内，则它直接就可以聚入核心点所在的类，如果插入的数据不能聚入已经有的类，则它的 Eps 邻域内只有噪声点，这样就有可能产生新的核心点，我们也必须扩充最大核心点集合。

4.1 算法描述

下面的算法描述当中，要用到下面一些定义和记号：

定义4 已聚类数据集合 D 中要插入的新数据记为 $InsertD$。

定义5 已聚类的数据集合 D 中待删除的数据集合记为 $DeleteD$。

定义6 已聚类的数据集合 D 经过增量算法的输出记为 $OutD$。

定义7 数据集合 D 的最大不相含核心点集记为 $LSNCCP$。

为了描述我们的算法，首先证明下面两个定理：

定理1 在已聚类的数据集合 D 中插入一个新的数据点 P 时，则新产生的核心点只可能出现在 P 点的 Eps 邻域内。

证明：如果有一点 Q 因为 P 的插入而从非核心点变成核心点，则 P 点一定在 Q 点的 Eps 邻域内，那么 Q 也在 P 的 Eps 邻域内。

定理2 在已聚类的数据集合 D 中删除一个旧的数据点 P 时，则由核心点变为非核心点的点只可能出现在 P 点的 Eps 邻域内。

证明：与定理1的证明相似(略)。

4.2 INCCP 算法

我们先给出处理插入和删除的算法：

算法1 插入操作增量更新最大不相含核心点集算法

```
InsertReference ( D,Eps,MinPts,LSNCCP,InsertD,cid)
  FOR i=1 to |InsertD|{
      P= InsertD[i]; OutD=D+InsertD;
      IF(Dist(LSNCCP,P)>Eps ) THEN {
          Result=OutD. RegionQuery(P,Eps);
          IF(|Result|>=MinPts) THEN {
              LSNCCP. add (P); R. cid = P; {R |
              RResult}
              P. cid = LSNCCP. NextID(cid); P. -
              number=(|Result|; }
          ELSE IF (OutD. SearchCorePoint (Result,
              Minpts,Eps)!=NULL) THEN
              P. cid = OutD. SearchCorePoint (Result,
              Minpts,Eps);
          }
          ELSE{
              P. cid=R{Q | Q∈LSNCCPΛDist(P,Q)
              <=Eps };
              Q. number + = 1; {Q | Q ∈
              LSNCCPΛDist(P,Q)<=Eps } }
      }
  }
}
```

算法2 删除操作增量更新最大不相含核心点集算法

```
DeleteReference ( D,Eps,MinPts,LSNCCP,DeleteD,cid){
  FOR i=1 to |DeleteD|{
      P= DeleteD [i]; OutD =D-DeleteD;
      IF (P. cid! =NULL) THEN {
          Result = OutD. RegionQuery ( P, Eps );
          StrartD=Result LSNCCP;
          IF (StrartD! =o) THEN {
              FOR j=1 to |StrartD|{
              Q=StrartD [ j]; Q. number - =1
              IF (Q. number<MinPts) THEN {
                  LSNCCP. delete (Q); Result'=
                  OutD. RegionQuery(Q,Eps);
                  R. cid = Null; {R | RResult'}
                  InsertD. add (Result'); }
              }
          }
      }
      D. delete (p);
  }
  InsertReference ( D,Eps,MinPts,LSNCCP,InsertD)
}
```

在算法1中，如果 $D=LSNCCP=\Theta$，那么它可以实现 LSNCCP 的算法的同样的功能。算法对数据集合 $InsertD$ 进行遍历，对于 $InsertD$ 中每一点 P 来说，如果它跟最大核心点

集合中的每个点的距离都大于 *Eps* 则在 *P* 周围做区域查询，如果 *P* 本身是核心点，就把它加入最大核心点集合当中，否则查找它的 *Eps* 邻域内看是否有核心点存在，如果有，则把 *P* 点映射到这个核心点。如果第二个这样的核心点存在，在后来的聚类算法当中，这两个核心点将会被聚为一类。如果它与几个 *LSNCCP* 集合当中的点的距离都小于 *Eps*，则把 *P* 影射到其中一个核心点，后面的聚类算法会把这些点都归于一类。

在算法2中，利用了算法1中生成聚类时候的信息，对于每一个待删除的点 *P*，算法2首先判断它是不是噪声点，如果不是噪声点，而且 *P* 是映射到最大不相含核心点集合上的，就在 *P* 上做区域查询，并找出在 *P* 点的 *Eps* 邻域与最大核心点集合的交集 *StartD*，由于 *P* 是映射到最大不相含核心点集合上的，因此 *StartD* 一定非空。对于这个集合当中的每一点 *Q*，如果它因为删除 *P* 而从核心点变为非核心点，就把 *Q* 从最大核心点集合当中删除，并把 *Q* 周围 *Eps* 邻域内的所有点置无映射标志，并添加到 *InsertD* 集合当中，无论如何在处理完这些之后，我们都必须把 *P* 从数据集合 *D* 中删除，最后我们调用插入算法把那些因删除点而失去映射标志点重新映射到最大核心点集合。

算法3 INCCP 聚类算法

```
CLUSTERING( Eps,MinPts,LSNCCP,cid){
    FOR i=1 to |LSNCCP|{
        P= LSNCCP [i];
        IF(P.cid>cid ) THEN {
            P.cid=NextID (cid); cid=NextID (cid);
        }
        FOR j=i+1 to |LSNCCP| {
            Q=LSNCCP[j];
            IF (Dist(P,Q)<=2 * Eps) THEN {
                Q.cid = P.cid; }
        }
        Arrange(LSNCCP);
    }
}
```

算法1和算法2只是给出了集合 *D* 中的新增点和删除以后剩下的点到最大不相含核心点集合上的映射，我们还必须把 Lsnccp 聚类，这个过程在算法3中完成。算法3利用了算法1和2中最大核心点集合当中每个点上 *cid* 信息来进行。在聚类时遵循这样的原则：如果点 *P*，*Q* 可以被聚为一类，则选取它们类标志 *cid* 比较小的作为它们的类标志。

每次在一个核心点上搜索完与它同一类的点以后，算法就把所有点的标志改为它们标志中最小的标志，并且对 LSNCCP 集合按照 *cid* 大小进行从小到大排序，又基于我们的 LSNCCP 集合自身本来就是按照从小到大排序的事实，我们有如下的定理：

定理3 在算法3当中，为了找出一个点的同类的点，我们只需要搜索排在这个点后面的 LSNCCP 集合当中的点即可。

证明：由于在算法3当中，在找出一个点 *P* 的同类点以后，我们都会重新排序 LSNCCP 集合，所以紧跟在点 *P* 之后的点 *Q* 要么是 *P* 的同类点，*Q* 与 *P* 已经有了同样的标志，要么是一个新的类的开始点。如果 *Q* 是 *P* 的同类点，则在 *Q* 上找同类点的时候只需要向后搜索就可以，因为 *Q* 前面跟它同类的点已经做过与它同样的标志；如果 *Q* 不是 *P* 的同类点，则 *Q* 不可能跟 *P* 前面的点同类，否则，它就会在排序算法的作用下排到 *P* 的前面。总之，在点 *Q* 找同类点只需要搜索排在它后面的点。

定理3保证了聚类算法的有效性。

5 算法分析

在本部分，先从理论上分析 LSNCCP 和 INCCP 的时间复杂度，然后给出 DBSCAN、LSNCCP 和 INCCP 的实验对比。

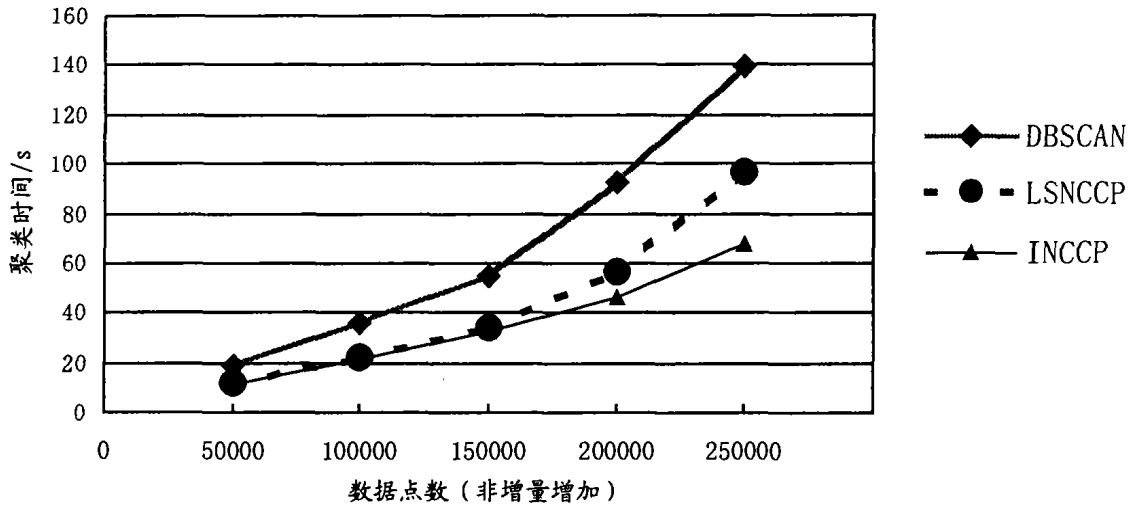

图1 DBSCAN、LSNCCP、INCCP 对 SQL SERVER 2000的测试结果

5.1 算法时间复杂度分析

先分析 LSNCCP 的时间复杂度，LSNCCP 的时间复杂度与最大核心点集合中点的个数有直接的关系。对于最大核心点集合当中点的个数，我们有如下的定理：

定理4 在算法 LSNCCP 当中，最大核心点集合中点的个数小于一个常数 *K*，这个常数等于所要聚类的集合 *D* 所占空间的体积和以 *Eps* 为半径的空间球体的体积之比，而与所要聚类的点的个数 *n* 无关。

证明：略。

在把集合 *D* 中的点映射到最大核心点集合的时候，所需要做的区域查询次数近似于最大核心点集合中的点的个数，因此，这个步骤的时间复杂度为 $O(K*n)$，第二个步骤，把核心点集合当中的进行聚类时间复杂度为 $O(K*\log_k)$，所以 *LSNCCP* 时间复杂度为 $O(n+k*\log_k)$。这个值跟 IGDCA 的非增量算法 GDCA 的时间复杂度相当。

在处理增量更新的时候，如果不考虑 I/O 操作，INCCP 与 LSNCCP 算法时间复杂度相当。跟 LSNCCP 算法一样，INCCP 不存在丢失点。

5.2 实验分析

实验环境，P4 1.6GHz CPU，物理内存256MB，可用内存

100MB，硬盘空间40GB，操作系统 Windows 2000 SERVER，数据库软件为 SQL SERVER 2000，程序用 JAVA 语言编写。图1、图2和图3就是对 DBSCAN、LSNCCP、和 INCCP 的性能测试的结果。

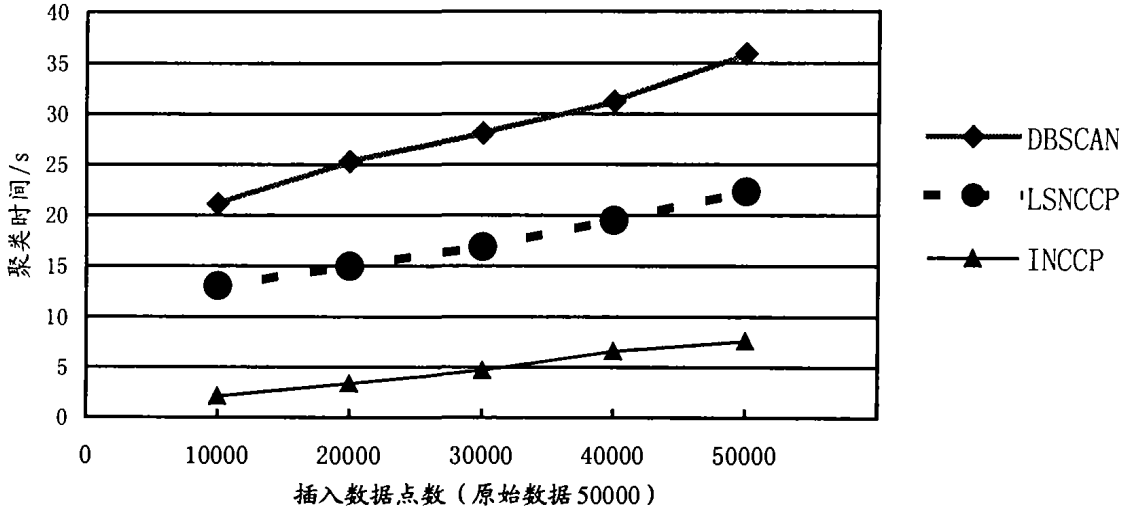

图2 DBSCAN、LSNCCP、LNCCP 针对插入数据的测试结果

图1显示的是三个算法在5个不同数量的数据点集合上的运行时间。当数据量小于150000的时候，INCCP 算法的性能略优于 LSNCCP 算法，而这两个算法的性能远优于 DBSCAN 算法；当数据量超过150000的时候，DBSCAN 和 LSNCCP 性能迅速变得很差，这是由于 PC 主存的限制造成的，当数据量超过150000达到200000的时候，DBSCAN 和 LSNCCP 为了处理这种大数据量情况频繁执行的 I/O 操作，而 INCCP 算法相对来说执行了较少的 I/O 操作，性能较优。

图2给出三个算法在50000个已聚类数据上插入不同数量的记录个数以后重新聚类的测试结果。结果显示，在增量插入的时候，INCCP 算法数倍优于 DBSCAN 和 LSNCCP 算法，这种优势并不因为数据插入的增多而减弱。

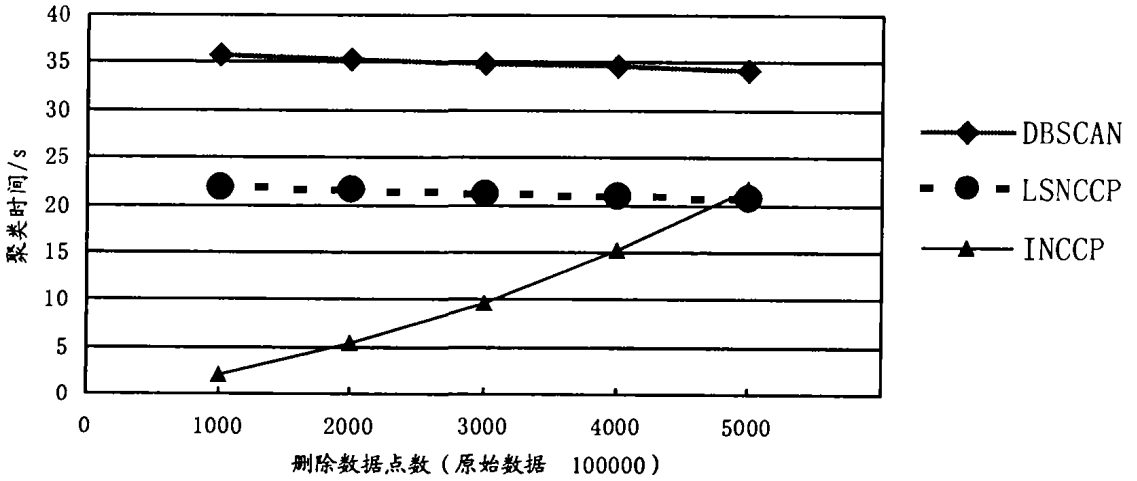

图3 DBSCAN、LSNCCP、INCCP 针对删除数据的测试结果

图3给出三个算法在100000个已聚类数据上删除不同数量的记录个数以后重新聚类的测试结果。由图中可以看出，当删除的数据量只占数据集合的很少部分的时候，INCCP 的性能优势比较明显，当删除的数据量越来越多，它的性能越来越差，当删除的数据量达到数据总量的5%的时候，它已经要用比 LSNCCP 算法重新聚类更长的时间。

总结 聚类是数据挖掘领域中最活跃的研究分支之一，聚类技术在其他的科学领域也有广泛的应用。基于 LSNCCP 算法，提出一种增量的聚类算法 INCCP，它在一定程序上改善了 LSNCCP 的性能，实现了的增量聚类，在大数据量的时候，它的性能也很稳定。今后我们的工作主要集中在改善算法在处理大规模删除操作时的性能，以及在高维空间中研究本文算法的性能。

参 考 文 献

1 Ester M, Kriegel H-P, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Simoudis E, Han J, Fayyad, U M , eds. Proc. of the 2nd Intl. Conf. on Knowledge Discovery and DataMining. Portland, Oregon: AAAI Press, 1996. 226～231

2 周水庚，周傲英，曹晶，胡运发. 一种基于密度的快速聚类算法. 计算机研究与发展，2003，37(11)：1287～1292

3 薛永生，翁伟，文娟，王劲波. 一种基于最大不相含核心点集的聚类算法(待发表)

4 Ester M, Kriegel H-P, Sander J, et al. Incremental clustering for mining in a data warehousing environment. In: Gupta A, Shmueli O, Widom J, eds. Proc. of the 24th Intl. Conf. on Very Large Data Bases. New York: Morgan Kaufmann Publishers Inc., 1998. 323～333

5 CHEN Ning, CHEN An, ZHOU Long-xiang. An Incremental Grid Density-Based Clustering Algorithm. Journal of software, 2002, 13(01): 1～7