

摘要

聚类分析作为数据挖掘的一个重要任务，具有广泛的应用领域，这些不同的应用都对聚类分析算法提出了新的要求。

本文提出了基于网格的并行聚类分析算法 PGMCLU，该算法的创新点主要包括：定义了网格紧凑度、网格密度连通、网格特征值、簇密度以及簇相似度的概念；提出了基于网格单元划分的数据分区方法，基于网格密度连通概念的局部聚类方法，以及基于簇相似度量度的局部聚类合并方法；实现了对网格密度阈值参数 $minPts$ 的自适应设置。该算法可以较好地处理高维和海量数据集，并具有识别不同形状和密度簇的能力。

数据流是指潜在无限的、持续而快速到达的具有时间顺序的数据对象的集合。数据流的实时性和潜在无限性，决定了数据流聚类分析算法与传统的基于静态数据的聚类分析算法相比，具有一些新的特性。

本文提出了基于网格的数据流聚类分析算法 GC-Stream，该算法的创新点主要包括：提出了描述网格单元概要信息的特征向量结构；对 SP-Tree 做了改进，提出了基于 List 结构的 LSP-Tree 空间索引结构；提出了对网格单元信息的指数衰减策略，以及对噪声网格单元和过时网格单元的剪枝策略。该算法较好地满足了数据流聚类分析的实时性要求，并对内存空间具有动态的适应性。

详细而全面的实验证明了 PGMCLU 和 GC-Stream 算法的正确性和有效性，因此，这些研究成果具有重要的理论价值和实际意义。

关键词：网格，并行，聚类分析，多密度簇，数据流聚类，LSP-Tree

Abstract

Clustering analysis, as an important task of data mining, has wide application fields. These different applications raise some novel requirements for clustering analysis algorithm.

This thesis proposes a novel grid-based parallel clustering algorithm for multi-density datasets, called PGMCLU. The innovative works of it are as follows. Define the concepts, including grid compactness, grid density-connected, grid feature, cluster density and cluster similarity. Propose the method for data partition based on grid partition, the method for local clustering based on grid density-connected concept, and the method for merging local clusters based on cluster similarity measure. Realize the adaptive set for parameter *minPts*. PGMCLU algorithm can better handle high-dimensional and massive datasets, and can be capable of identifying clusters with distinguished shape and density.

Data stream is a sequence composed of a series of infinite, successive, high-speed, and time-ordered data objects. Data stream has the characteristics of real-time and infinity, which determines that clustering algorithm for data stream compared with traditional clustering algorithm for static dataset has some distinguished properties.

This thesis proposes the grid-based clustering algorithm for data stream, shorten for GC-Stream. The innovative works of it are as follows. Propose the concept of grid feature vector for describing the grid summary information. Improve the SP-Tree structure, and propose the novel spatial index structure LSP-Tree based on List data structure. Propose the exponential damped strategy for grid information, and the pruning strategy for noisy grid and outdated grid. GC-Stream algorithm can better satisfy the real-time requirement of data stream clustering, and can be adaptive for memory size.

.Detailed and complete experiments have proved the correctness and effectiveness of PGMCLU and GC-Stream algorithm, therefore, these novel algorithms will have significant theoretic value and practical role.

Keywords: grid, parallelism, clustering analysis, multi-density cluster, clustering

analysis for data stream, spatial partition tree based on List data structure

原创性声明

本人郑重声明：本人所呈交的学位论文，是在导师的指导下独立进行研究所取得的成果。学位论文中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究成果做出重要贡献的个人和集体，均已在文中以明确方式标明。



本声明的法律责任由本人承担。

论文作者签名： 陈毅 日期： 2010.5.18

关于学位论文使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用学位论文的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本学位论文。本人离校后发表、使用学位论文或与该论文直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

保密论文在解密后应遵守此规定。

论文作者签名：  导师签名：  日期： 2010.5.21

第一章 绪论

1.1 研究背景及意义

数据挖掘 (data mining) 是指发现数据中蕴含的潜在有用的知识的过程, 知识包括规则、模式及结构等。数据挖掘涉及到多个学科, 包括数据库和数据仓库、统计学、信息检索、人工智能、神经网络、模糊集、粗糙集、信号处理、高性能计算等。数据挖掘理论的应用领域广泛, 包括图像处理、生物信息学、气象信息分析、社会网络分析、图挖掘、入侵检测、数据流挖掘、时间序列预测等。基本的数据挖掘技术包括频繁模式挖掘、分类和预测以及聚类分析。

聚类分析 (clustering analysis) 是一项基本的数据挖掘任务。聚类分析是将数据对象集合根据对象之间的相似度量划分为簇 (cluster) 的过程。聚类分析的基本方法包括: 基于划分的方法 (partition-based methods)、基于层次的方法 (hierarchy-based methods)、基于密度的方法 (density-based methods)、基于网格的方法 (grid-based methods) 和基于模型的方法 (model-based methods)。基于划分的方法包括 k-means [1]、PAM [2]、CLARA [2] 和 CLARANS [3] 等。基于层次的方法包括 BIRCH [4]、ROCK [5]、CURE [6]、Chameleon [7] 等。基于密度的方法包括 DBSCAN [8]、OPTICS [9]、DENCLUE [10] 等。基于网格的方法包括 CLIQUE [11]、STING [12]、WaveCluster [13]、DCLUST [14]、PROCLUS [15] 等。基于模型的方法包括 SOM [16] 和 COBWEB [17] 等。除了这些基本的聚类分析方法之外, 目前, 聚类分析的热点研究领域包括基于约束的聚类、数据流聚类、子空间聚类、增量聚类、基于遗传算法的聚类、基于蚁群算法的聚类等。

基于网格的聚类分析算法是聚类分析中一种非常重要的方法, 在聚类高维和海量数据集时具有明显的优势。基于网格的聚类算法将数据空间量化为有穷数目的网格单元, 然后将数据对象投影到这些网格单元中, 并保存网格单元的摘要信息 (summarization information), 所有的聚类操作都在这些网格单元上面进行。不同的基于网格的聚类分析算法在如下方面存在差异, 包括网格划分策略 (常见的包括静态划分和动态划分两种)、网格摘要信息结构、网格单元集合索引结构、噪声数据的处理和簇的识别机制等。目前, 最新的基于网格的聚类分析算法包括

GRIDSCAN[18]、GMDBSCAN[19]、EDACluster[20]、GriDBSCAN[21]、GNN[22]、SCI[23]、MMNG[24]、GDILC[25]、IGDCA[26]算法等，这些算法中的部分算法将在 2.3 节中详细分析。基本的基于网格的聚类分析算法的聚类过程的时间复杂度主要依赖于数据空间中包含的网格单元的数目，具有近似线性的时间复杂度。此外，其还能够较好地处理高维和海量数据集，在增量聚类和子空间聚类等方面也体现出优势，因此，它在聚类分析中得到了广泛的研究和应用。

集群系统[27][28][29]（PCs cluster system）在近年来得到广泛的应用。集群系统是将一组高性能的计算机通过特定的网络结构联接起来，在操作系统和并行环境的支撑下，实现对系统资源的统一管理和协调调度。它通过消息传递的机制为程序设计人员提供了一个整体的并行编程环境。基于 MPI（message passing interface）的并行编程技术提高了集群系统环境下并行程序开发的效率。由于集群系统具有高性能（high performance）、可扩展性（scalability）、高可用性（availability）、透明性（transparency）、可编程性（programmability）等典型特征，因此，集群系统在大规模数据处理、图像处理、工程计算等领域得到越来越广泛的应用。

数据的海量性和高维性都给聚类分析算法带来了挑战，而集群系统的这些特征给研究集群系统下的并行的聚类分析算法带来了机遇。此外，实际的数据集中经常包含密度不同的簇，传统的基于密度的聚类分析算法（如 DBSCAN）往往不能得到准确的聚类结果，而针对多密度数据集的聚类分析算法（如 SNN[30][31]）又存在算法时间复杂度高以及聚类结果精度差（主要体现在对噪声数据的处理方面）等问题，因此，研究适合于多密度数据集的聚类分析算法也是一个热门的研究方向。

数据流是指连续且具有时间顺序的数据构成的集合。数据流是伴随着人们对实时数据的处理而产生的，如电信通话数据、银行交易数据、证券交易数据、网络流量监测数据、网络操作日志数据、气象监测数据、传感器数据等[32][33][34][35]。数据流具有连续性、按时间顺序的、潜在无限的、快速变化的等特性，这些特性给数据流聚类分析算法带来了挑战。与传统的基于静态数据的聚类分析算法相比，数据流聚类分析算法具有三个明显的特性：（1）它强调时间性，数据流聚类分析算法必须能够发现数据流的演变规律或任意时间段内的数据流聚类结果。（2）单遍扫描。由于受到内存空间的限制和数据流聚类分析算法实

时性的要求, 数据流聚类分析算法必须实现对数据对象的单遍扫描。(3) 强调对摘要数据结构 (summary data structure) 的设计, 数据流聚类分析算法必须将数据对象的信息以摘要数据结构的形式保存起来, 以方便聚类算法对摘要结构的分析。目前, 典型的数据流聚类分析算法包括 STREAM[36]、CluStream[37]、HPStream[38]、AIN-Stream[39]、Cell trees[40]等。针对数据流聚类分析算法的这些新的特性, 数据流聚类分析算法已经变成数据挖掘中的一个研究热点。

1.2 研究内容

本文针对当前聚类分析算法存在的问题, 结合最新的研究成果, 主要开展了对基于网格的并行的聚类分析算法和基于网格的数据流聚类分析算法的研究, 具体的研究内容包括:

(1) “维灾难” (dimension curse) 问题已经成为聚类分析算法处理高维数据集时的一个主要问题。同时, 许多聚类分析算法 (如 DBSCAN、SNN 等) 在处理海量数据集时也面临高的时间复杂度 (如 $o(N^2)$, 其中 N 是数据对象的数目) 的困扰。此外, 针对多密度数据集的聚类分析算法存在时间复杂度高及聚类结果精确度不高 (主要表现在对噪声数据的处理方面) 等问题。本文针对这些问题, 提出了基于网格的并行的聚类分析算法 (grid-based parallel clustering analysis algorithm for multi-density datasets, 简称 PGMCLU)。PGMCLU 算法基于数据并行 (data parallelism) 的思想, 这是一种典型的分而治之的方法, 通过各个节点对数据的独立聚类 and 最终聚类结果的合并实现了对整个数据集的聚类。PGMCLU 算法提出了网格紧凑度、网格密度直接可达、网格密度可达、网格密度连通等概念, 其中网格紧凑度 (grid compactness) 度量较好地反映了网格中数据点之间的紧密程度; 提出了新的网格划分方法, 以及基于参考维的数据分区策略; 实现了根据数据的分布特征自动确定 $minPts$ 参数的值; 为了更好地适应分而治之的策略, 提出了利用网格特征向量来描述网格的摘要信息结构, 并建立了 SP-Tree, 提高邻居网格的查找效率; 提出了基于网格密度连通概念的聚类方法, 以及识别边界网格中边界数据点的方法; 提出了簇密度和簇相似性的概念, 并且基于它们提出了簇合并算法。最后, 对该算法进行了性能分析和实验验证。

(2) 针对如何设计有效的能够对连续的数据流进行快速处理, 并能发现数

据流的演变规律的要求,本文提出了基于网格的数据流聚类分析算法(grid-based clustering analysis algorithm for data stream, 简称 GC-Stream)。GC-Stream 算法采用了衰减窗口模型(damped window model)来发现数据流的演变规律。GC-Stream 算法提出了新的聚类模型 LSP-Tree; 提出了聚类模型维护策略以及聚类模型的剪枝策略,较好地实现了模型对数据对象的快速处理,以及适应了数据流聚类分析算法对内存空间的限制要求。最后,通过实验验证和评价了算法的正确性和性能。

1.3 论文组织结构

本文围绕着基于网格的聚类分析算法的研究与实现,从并行化基于网格的聚类分析算法和设计基于网格的数据流聚类分析算法两个方面,开展了研究工作。论文内容按照以下结构组织:

第一章:绪论。主要介绍了课题研究的背景、意义、内容及论文组织结构。引出了论文的主要研究内容为基于网格的并行的聚类分析算法以及基于网格的数据流聚类分析算法。

第二章:基于网格的聚类分析算法概述。首先,介绍了聚类分析的概念及过程,对聚类分析过程的每个步骤中涉及到的关键技术进行了详细的阐释。其次,分析了不同的应用对聚类分析算法提出的特定要求,并给出了聚类分析算法面临的挑战。最后,阐释了基于网格的聚类分析算法的基本原理及特点,并详细分析了四种典型的基于网格的聚类分析算法的原理及特性,包括 CLIQUE、GRIDBSCAN、GMDBSCAN 和 GNN 算法。

第三章:并行的基于网格的聚类分析算法:PGMCLU。首先,描述了 PGMCLU 算法的总体框架,对算法中涉及到的几个重要的过程进行了阐释。其次,介绍了算法中涉及到的基本概念。然后,详细描述了算法的实现过程,包括数据分区、构建 SP-Tree、局部聚类和局部聚类合并这四个关键的步骤,对步骤中涉及到的关键技术进行了具体阐释。接下来,从时间复杂度和加速比方面分析了算法的性能。最后,从聚类准确性、相对加速比以及效率三个方面对算法进行了实验验证和性能评价。

第四章:基于网格的数据流聚类分析算法 GC-Stream。首先,介绍了数据流

的概念，以及数据流聚类分析的特性、窗口模型及典型算法。其次，描述了算法的总体框架。然后，从聚类模型，模型维护、更新和剪枝策略方面对算法进行了详细的描述。最后，分析算法性能并给出实验验证和性能评价。

第五章：总结与展望。总结本文的研究工作，并展望未来的研究方向。

第二章 基于网格的聚类分析算法概述

2.1 聚类分析概念及过程

聚类分析 (clustering analysis) 是数据挖掘 (data mining) 技术的重要一种。相对于分类算法而言, 聚类分析算法事先不知道类的标号, 因此, 其也被称为无监督的学习 (unsupervised learning)。聚类分析指的是将对象集合根据对象之间的相似度量划分为不同的簇 (cluster) 的过程。对象之间的相似度量通常通过计算它们之间的距离来度量, 距离的计算方式因聚类分析处理的数据类型的不同而异。聚类分析源于统计学、数据挖掘、生物学以及机器学习等多个领域。目前, 聚类分析已被广泛地应用于信息检索、图像处理、模式识别、Web 信息处理、市场调研、地理学、医学、生物信息学、空间数据分析等多个方面。一个有实际价值的聚类分析算法通常包括 5 个关键的处理步骤: 数据预处理, 相似度量定义, 聚类, 聚类结果输出, 聚类结果解释。

2.1.1 数据预处理

数据预处理主要包括数据清理, 数据变换和数据规约功能。数据清理包括补充缺失值、光滑数据、剔除噪声数据等。数据变换将数据转换成适合于聚类分析的形式, 数据变换通常包括光滑、聚集、数据泛化、规范化和属性构造。数据规约是在保持原数据集的完整性的同时得到数据集的规约表示, 聚类分析中经常用到的规约方法是属性子集选择 (或特征子集选择), 尤其是聚类分析算法在处理高维数据集时。属性子集选择方法通常需要从给定的属性集中选取与聚类分析最相关的属性子集, 这是由于随着维度的增加, 数据的分布日趋稀疏, 而只有少数的维会影响最终簇的形成, 但是其它不相关的维的数据可能会以噪声数据的形式影响真实的簇。数据预处理不必是聚类分析过程的必需步骤, 但是数据预处理有助于提高聚类分析的结果质量。

2.1.2 定义相似度量

相似度定义过程主要是定义数据对象之间的相似度量指标,常见的度量数据对象之间相似度的方法是计算数据对象之间的距离,距离越短,数据对象越相似。目前,出现了一些新的度量对象之间相似度的指标,具体介绍如下:

(1) 随着图在复杂结构建模中的广泛应用,图挖掘已经变为数据挖掘中一个重要和活跃的课题,其包括频繁子图挖掘、频繁结构模式、图分类、图聚类等。在基于图的聚类中,结点是对象,结点之间的边表示对象之间的联系。簇被定义为连通分支 (**connected component**),即簇中的对象互相连通,而不同簇中的对象之间不连通。

(2) 在基于网格的聚类算法中 (**grid-based clustering**),我们将数据空间量化为有穷数目的网格,所有的聚类分析操作都在网格集合上进行。网格信息中通常保存了网格的密度值,即网格中包含的数据点的数目。网格 C_j 和网格 C_i 是相似的,记作 $similar(C_i, C_j)$,则它们满足式 2-1 所示的条件。

$$\frac{\min(C_i.density, C_j.density)}{\max(C_i.density, C_j.density)} > \varphi, \text{ and } C_j \in Neighbours(C_i) \quad (2-1)$$

(3) 在 SNN (**shared nearest neighbors**) 算法中,针对基于密度的聚类分析算法对不同密度簇的识别能力差的问题,提出了 SNN 相似度量。该度量首先计算每个数据对象的 k 个最近邻居,然后将对象之间的相似度定义为共享近邻数目。对于数据点 x_i 和 x_j , 它们之间相似度 $similarity(x_i, x_j)$ 的计算公式如式 2-2 所示。如果 x_i 和 x_j 相互在对方的 k 近邻中,则 $similarity(x_i, x_j)$ 之间的相似度被定义为它们共享近邻的数目,否则,相似度被定义为 0。SNN 相似度较好考虑了对象周围区域数据的分布情况,提高了聚类分析算法对不同密度簇的识别能力。共享最近邻聚类算法为了计算每个数据对象的 k 个最近邻居,其必须计算任意两个数据对象的距离,因此,在一般情况下,该聚类算法的时间复杂度是 $O(n^2)$ 。在特殊情况下(例如在处理低维数据时),如果使用索引技术(如基于区域划分的 R 树)可以将查找 k 最近邻的时间复杂度降低到 $O(n \log^n)$ 。

IF($x_i \in k_nearest_neighbor(x_j)$, and $x_j \in k_nearest_neighbor(x_i)$)
 THEN $similarity(x_i, x_j) = k_nearest_neighbor(x_j) \cap k_nearest_neighbor(x_i)$ (2-2)
 ELSE $similarity(x_i, x_j) = 0$

通常情况下，数据对象之间的相似度可以通过相似度矩阵 (similarity matrix) 给出，如式 2-3 所示，该结构是一个 $n \times n$ 矩阵， d_{ij} 表示对象 i 和对象 j 之间的相似度， $d_{ij} \geq 0$ ，对象 i 和对象 j 越相似， d_{ij} 的取值越大。

$$\begin{pmatrix} 0 & \dots & d_{1n} \\ d_{21} & 0 \dots & d_{2n} \\ \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} \dots & 0 \end{pmatrix}_{n \times n} \quad (2-3)$$

2.1.3 聚类

在数据预处理和定义相似度度量之后，就可以进入聚类处理阶段。典型的聚类分析算法可以划分为三种常见类型：

(1) 以目标函数最优化为原则，将数据对象进行分组。如基于划分的方法和基于模型的方法，大部分都以优化目标函数为准则，将数据对象划分到不同的簇中。

(2) 根据簇的特性将对象分组。如在基于密度的簇中，簇被定义为最大的密度相连 (density-connected) 对象的集合。在基于网格的聚类分析算法中，簇被定义为最大的密度连通网格形成的集合。在 SNN 算法中，基于对象的 k 最近邻计算，如果对象 x_i 和 x_j 相互在对方的 k 最近邻中，并且共享近邻数目大于阈值 λ ，则将它们划分在同一个簇中。

(3) 依据簇之间的相似性将对象分组。如在凝聚层次聚类算法中，初始时将每个数据对象都视为一个子簇，然后根据簇之间的相似度合并这些子簇，直到用户指定的条件满足 (例如簇的数目达到了用户规定的簇数目)。

2.1.4 聚类结果输出

聚类结果输出是将代表最终簇的对象以某种方式输出。常见的输出方式包括：输出簇中包含的数据点对象；输出能够反映簇特征的代表性数据点；以簇特征的

形式输出簇结果（如式 2-4 所示，利用 X 和 Y 的取值范围来描述簇 C_x ）；输出簇中包含的网格集合（如式 2-5 所示，簇 C_x 由网格 $G_i (i=1,2,\dots,n)$ 构成）；输出簇的中心点（如在处理城市规划的选址问题时，通常将该问题抽象为一个基于约束的聚类问题来解决，此种聚类问题最终的输出结果是簇的中心点，这代表了最佳的选择地点）。

$$C_x = \{x_i \leq X \leq x_j, y_m \leq Y \leq y_n\} \quad (2-4)$$

$$C_x = \{G_1, G_2, \dots, G_i, \dots, G_n\} \quad (2-5)$$

2.1.5 聚类结果解释

在获得聚类分析形成的簇结果后，对聚类结果的解释也是聚类分析的一个重要步骤。通过对结果的解释，可以发现实际问题中隐藏的潜在有用的模式，例如对超市购物人群的特征进行聚类分析，可以将顾客分为若干个目标顾客群，通过对各个群体的特征进行分析，可以为他们制定适当的营销策略。针对高维数据集的处理，部分聚类分析算法采用了属性子集选择和维度规约方法，这些方法一方面降低了数据集的维数，为聚类分析算法得到高质量的聚类结果提供了保证，另一方面也给聚类结果的解释带来了挑战。针对海量数据集的处理，某些聚类分析算法利用统计学中的抽样（sampling）技术选取了数据集中的部分样本数据，这样极大地减少了要处理的数据集的规模，此种方法在提高算法效率的同时也为结果的解释带来了困难。当数据集的维数较高时（ $d > 3$ ），聚类结果的可视化显示也是一个问题。目前，常见的可视化显示技术是空间变换技术，该技术利用降维的方法将高维空间中的数据变换到低维空间中显示，但要达到无损变换也是一个难点。

2.2 聚类分析算法的要求及挑战

随着聚类分析算法理论的广泛研究，其应用领域也越来越广泛，各种应用都对聚类分析算法提出了特定的要求。本节将从聚类分析算法处理的数据所拥有的数据特性、簇特性以及聚类本身的限制条件三个方面来讨论聚类分析算法所面临的一些要求。从数据特性的角度分析，聚类分析算法的要求包括处理不同类型的

数据、剔除噪声、支持对高维数据集的聚类。从簇特性的角度来看，聚类分析算法的要求主要是能够处理任意形状的簇。从聚类分析算法本身的限制条件来看，聚类分析算法的要求包括输入参数少、可伸缩性、增量聚类、支持基于约束的聚类。

(1) 能够处理不同类型的数据。聚类分析算法必须具有处理不同类型的能力。许多聚类分析算法只能处理数值类型的数据，一个好的聚类分析算法应该能够处理多种类型的数据，如分类变量、二元变量、序数变量以及比例标度的变量等。

(2) 剔除噪声。许多数据集中常常包含噪声数据，如果不能合理地处理这些噪声数据，将会严重影响聚类算法的效率及聚类结果质量。如基于划分的 k -means 算法[1]对噪声或孤立点数据就比较敏感，它们将减缓簇均值的收敛速度，增加了算法的时间复杂度。而对于基于密度的 DBSCAN[8]算法，其需要两个输入参数 ε 和 $minPts$ (ε 表示邻域半径， $minPts$ 表示 ε 半径内包含的最小数据点数目)，在聚类的过程中，为了确定核心对象 (core object)，其需要计算每个对象的 ε 邻域内包含的数据点的数目 $total_pts$ ，如果 $total_pts$ 小于 $minPts$ ，则将该对象视为噪声数据点进行处理，形式化的描述如式 2-6 所示。在基于网格的聚类分析算法 CLIQUE[11]中，需要计算网格中包含的数据点的数目 $counts$ ，如果 $counts$ 的值小于输入参数 $minPts$ ($minPts$ 表示网格中包含的数据点的最少数目)，则将该网格定义为稀疏网格，这样也很好地剔除了噪声的影响，形式化的描述如式 2-7 所示，其中 C_i 表示 i^{th} 网格， $I_j (j=1,2,\dots,d)$ 表示网格 C_i 在 j^{th} 维的索引， x_k 表示 k^{th} 个数据点。

$$total_pts(x_j) = \left| \left\{ x_i \mid x_i \in \varepsilon_radius(x_j) \right\} \right| \quad (2-6)$$

if $total_pts(x_j) < minPts$, then x_j is noise (or outlier)

$$\text{suppose } C_i = (I_1, I_2, \dots, I_d),$$

$$counts(C_i) = \left| \left\{ x_k \mid x_{k1} = I_1, x_{k2} = I_2, \dots, x_{kd} = I_d \right\} \right| \quad (2-7)$$

$$\text{if } counts(C_i) < minPts, \text{ then } C_i \text{ is sparse grid.}$$

(3) 支持对高维数据集的聚类。在高维数据集中，通过传统的欧几里得距离来度量对象之间的相似度变得不再可行，这是因为，随着数据集维度的迅速增长，数据点的分布越来越稀疏，对象之间的欧几里得距离趋于一致。面对这种困难，可以采取两种方法来解决这些问题。一种方法是采用属性子集选择或维规约

技术,降低数据集的维度。另一种方法是定义新的相似度量指标,如 SNN 算法中的共享 k 最近邻等。

(4) 能够处理任意形状的簇。簇的形状可以是球形的、矩形的、椭圆形的、甚至是任意形状的。一个好的聚类分析算法应该能够处理任意形状的簇。基于划分的 k -means 算法和基于层次的 BIRCH 算法都只能处理球形的簇。真正能够处理任意形状簇的算法是基于密度的 DBSCAN 算法,该算法不再考虑使某个目标函数达到最优值,而是将簇定义为由密度连通数据点构成的最大集合,簇的形成是基于数据点之间的密度可达性来定义的,由于较好地考虑了数据点之间的关系,因此, DBSCAN 可以发现数据集中包含的最自然的簇。其它的能够处理任意形状的簇的算法包括 Chameleon 和 CURE。

(5) 输入参数少。聚类分析算法通常要求用户输入聚类参数,如 k -means 算法需要用户输入期望的簇的个数 k , DBSCAN 算法需要输入 ε 和 $minPts$ (ε 表示邻域半径, $minPts$ 表示 ε 半径内包含的最少数据点数目), CLIQUE 算法需要输入网格的间隔长度 L 以及网格中包含的最少数据点的个数 $minPts$, 这些参数的确定对用户来说是困难的。此外,聚类分析算法易受参数影响,参数的差异将影响聚类效率及聚类结果质量。例如对于 DBSCAN 算法而言,当数据集中包含的簇的密度不同时,使用全局的阈值参数 ε 和 $minPts$, 在低密度区域,对象的 ε 邻域内包含的数据点将可能小于 $minPts$, 此时, DBSCAN 算法将会将这些数据点误判为噪声,从而不能得到理想的聚类结果。因此,为了将参数对聚类算法的影响程度降到最低,理想的方法应该是根据数据的分布特征自适应地设置关键参数。

(6) 可伸缩性。聚类分析经常要处理大型的数据集,而许多聚类分析算法仅仅适合处理中小规模的数据集。如对 DBSCAN 算法而言,为了确定对象 x_i 的 ε 邻域内包含的数据点数目,必须首先计算出对象 x_i 到任意数据对象 $x_j (j=1,2,\dots,n)$ 之间的距离,因此, DBSCAN 算法的时间复杂度和空间复杂度都为 $o(n^2)$, 特殊地,当建立了基于区域查询的 R 树索引时,其时间复杂度将降低到 $o(n \log n)$ 。而对基于网格的聚类算法而言,其将数据空间量化为有穷数目的网格,所有的聚类分析操作都在网格上进行,将数据对象指派到指定的网格并计算网格的密度所需的时间复杂度是 $o(m)$ (m 是数据集中数据点的数目),如果为邻居网格的查找建立了空间索引结构,如 SP-Tree 树,则算法总的复杂度是

$o(m \log^m)$ 。在聚类分析算法处理大型数据集时, 算法时间复杂度为 $o(\log^n)$ 是合适的, 而 $o(n^2)$ 的时间复杂度就显的不合实际。解决聚类分析算法可伸缩性的重要方法包括数据抽样和划分等, 但是这些方法可能对最终聚类结果的精确性产生一定的影响。在保证聚类结果质量的前提下, 可以考虑设计并行的聚类分析算法, 这将是提高聚类分析算法可伸缩性的一个重要途径。

(7) 增量聚类 (incremental clustering)。增量聚类是指当有新的数据点到达时, 聚类分析方法能够即时更新已有的聚类结构, 而无需重新运行聚类分析算法。对于数据流聚类分析算法而言, 增量聚类这一特性显的异常重要, 数据流聚类分析算法必须对潜在无限的数据做出即时的处理。基于网格的聚类分析算法很好地支持了增量聚类的功能, 当有新的数据点 x_i 到来时, 只需确定 x_i 所属的网格单元并更新网格单元的密度 *counts* (*counts*+1) 即可, 这样实现了对新的数据点的快速处理。针对经常有新的数据生成的数据集而言 (在实际应用中, 大部分数据集都是这种情况, 如电信数据、网络流量数据、银行交易数据等), 支持增量聚类功能的聚类分析算法是必需的, 因此, 研究增量聚类算法也是聚类分析的重要研究课题。

(8) 基于约束的聚类。在实际应用中, 可能需要处理基于约束的聚类分析。根据约束条件约束的对象不同, 可以将约束条件大体上分为三类, 包括特征级 (或属性级) 约束, 数据对象级约束和簇级约束。特征级约束是对数据集中的属性施加的约束条件, 如顾客信息构成的数据集中, 规定顾客的年龄范围或收入范围等。数据对象级约束是指对数据对象本身或数据对象之间的关系施加的约束, 数据对象之间的约束常见的有两种, 包括 *must-link* 关系和 *cannot-link* 关系, 这些约束关系可能来自于用户对最终簇的期望, 也可能是通过一些背景知识转换而来的。簇级约束是对最终簇的特性施加的约束条件, 如规定每个簇中包含的数据对象的数目等。针对约束条件的类型不同, 对约束条件的处理方式也不相同, 例如在基于障碍物的聚类分析中, 当采用基于网格的方法处理此类问题时, 针对数据对象级的约束, 基于网格的方法经常是将障碍物这种实际的数据对象首先抽象为抽象的空间对象 (如点、线、面等), 然后将障碍物形成的网格视为稀疏的网格, 从而达到处理障碍物的目的; 当采用基于密度的方法时, 而对于数据对象之间的关系施加的约束条件, 可以在根据密度可达概念形成簇的过程中考虑到约束条件,

从而达到处理数据对象之间约束条件的目的。如何在考虑约束条件的情况下得到高质量的聚类结果也是聚类分析算法面临的挑战。

上面具体分析了实际的应用领域对聚类分析算法提出的一些特定要求,而这些要求也就为聚类分析算法的设计提出了新的挑战。聚类分析算法面临的挑战包括可伸缩性(包括数据规模的可伸缩性和数据维度的可伸缩性),对多种数据类型的聚类,确定聚类输入参数,对多密度数据集的聚类,发现任意形状的簇,增量聚类,基于约束的聚类等。目前,聚类分析的研究领域日益广泛,如数据流聚类分析,文本聚类分析,多媒体数据的聚类分析,生物数据的聚类分析,时序数据的聚类分析,对图的聚类分析等,这些研究领域给聚类分析技术提出了更多的挑战,如相似度的度量、距离的计算、在线聚类、数据的单次扫描、数据对象索引结构、数据对象的存储、结果的解释等。

2.3 基于网格的聚类分析算法

基于网格的聚类分析算法(grid-based clustering analysis algorithm)的基本思想是对数据集的每一个维进行划分,这样便可将数据空间量化为有穷数目的互不重叠的网格,所有的聚类分析操作都在这些网格上进行。基于网格的聚类算法的优点是聚类分析算法的时间复杂度独立于数据对象的数目,只与网格的数目有关,极大地提高了聚类效率。另外,由于使用摘要数据结构来描述网格单元信息,因此,其也适合增量聚类。基于网格的聚类分析算法的缺点是粗略地将稀疏网格($C_i.counts < \tau$)中的数据点处理为噪声,降低了聚类结果的精确度。此外,基于网格的聚类分析算法的聚类结果过分地依赖于输入参数 τ , τ 值的选择对用户来说是困难的, τ 值过小,则可能导致将不同密度的簇合并在一起, τ 值过大,则可能将低密度区域的数据点识别为噪声,因此,可行的方法是根据数据的分布特征自动地确定 τ 。在 τ 值的计算方面,可以通过最大密度网格的密度来计算 τ ,也可以在聚类的过程中采用密度阈值递减技术选择多个密度阈值 τ (这种方法将提高基于网格的聚类分析算法处理多密度数据集的能力)。通常情况下,基于网格的聚类分析算法包含3个基本的处理步骤。(1)划分网格单元。即对数据空间的每一维进行划分,形成网格结构。维的划分策略常见的包括将每一个维划分为等宽的间隔,这样如果每个维被划分为 m 个间隔,则整个数据空间将被划为为 m^d

个互不重叠的网格单元；另外一种划分网格单元的方法是采用自适应的网格划分策略，根据对象在每一维的密度分布情况不同将维划分为不同大小的网格（如果每一维被划分的网格数目是 $I_i (i=1,2,\dots,d)$ ，则整个数据空间被划分为 $\prod_{i=1}^d I_i$ 个网格单元）。（2）识别核心网格。将数据对象投影到指定的网格，并计算网格的密度（即单位体积中包含的数据点的数目，如果网格的体积相同，则密度被定义为网格中包含的数据点的数目）。如果网格 C_i 的网格密度大于密度阈值 τ ，则将 C_i 标记为核心网格；否则，将其标记为稀疏网格并将其删除。（3）识别簇。簇是由邻接的核心网格构成的。识别簇的过程是一个迭代的过程，每次从未聚类的核心网格集合中选择一个网格作为初始网格，从该网格出发采用深度优先或广度优先策略寻找邻居网格中的密度相似网格，并将其添加到当前簇中，直到没有新的核心网格被加入到当前簇为止。下面将介绍几种典型的基于网格的聚类分析算法：

1. CLIQUE (CLustering In QUEst) 算法[11]。CLIQUE 算法为发现高维空间的子空间中的簇提供了一种途径。对于实际的应用来说，相对于整个空间中形成的簇而言，子空间中可能包含着潜在有用的簇。对于 d 维的数据空间，其子空间的数目是 $2^d - 1$ ，对于这样庞大的一个基数，如果对数据集中的数据在每一个子空间中的投影都做聚类分析以便发现包含在该子空间中的簇，这样的方法显然是不可行的。CLIQUE 算法利用了单调性质，对于统一的网格密度阈值 $minPts$ ，如果网格 C_i 在 k 维空间中是密集网格，则网格 C_i 在 k 维空间的所有子空间中对应的网格单元也都是密集的。反之，如果网格 C_j 在 $k-1$ 维子空间不是密集网格，则在 k 维空间中由该网格中的数据点形成的网格也不可能是密集网格，这条性质称为 CLIQUE 算法的反单调性质。CLIQUE 算法包括 3 个主要的实现步骤。（1）找出对应于每个属性的一维空间中的稠密网格，形成一维空间中的稠密网格单元集合 DG_1 。（2）令 $k=2$ ，由 $k-1$ 维空间中的稠密网格单元集合 DG_1 形成 k 维空间中的候选稠密网格集合 $candidate_DG_k$ ，根据 CLIQUE 算法的反单调性质，筛选 $candidate_DG_k$ 中的候选稠密网格单元，得到 k 维空间中的稠密网格单元集合 DG_k 。令 $k=k+1$ ，迭代执行步骤（2），直到候选稠密 k 维网格单元集合为空。（3）通过寻找邻接的稠密网格形成的最大区域来发现簇。为了提高聚类结果的可解释性，CLIQUE 算法利用 DNF 表达式来描述簇。对于 k 维子空间 S ，生成

最小簇描述 (minimal cluster descriptions) 过程的输入是 S 中连通的 k 维网格单元的集合, 每一个这样的集合对应一个簇 C , 输出是能够覆盖该簇的多个区域形成的集合 R , 并根据 R 得到最小覆盖, 簇 C 和覆盖区域 R 之间的关系如式 2-8 所示, 其中 G_j 表示包含在簇 C 中的 j^{th} 网格。CLIQUE 算法的优点是可伸缩性, 以及可以发现任意子空间中的簇, 并对最终形成的簇提供了描述表达式。缺点是采用全局的网格密度阈值 τ 来识别不同子空间中的核心网格将导致聚类结果的不精确。

$$\begin{aligned} &\text{suppose } R = \{R_1, R_2, \dots, R_n\} \\ &\text{if } (R \text{ is a cover of cluster } C \text{ in dataspace } S) \\ &\text{then } R_i (i = 1, 2, \dots, n) \text{ contained in } C, \text{ and } G_j \in C \text{ contained at lease one } R_i \end{aligned} \quad (2-8)$$

2. GRIDBSCAN (Grid Density-Based Spatial Clustering of Applications with Noise) 算法[18]。GRIDBSCAN 算法是对经典的基于密度的聚类分析算法 DBSCAN 的改进。DBSCAN 算法的优点是可以发现任意形状的簇, 其缺陷主要包括对输入参数 ε 和 $minPts$ 比较敏感, 不能较好地识别不同密度的簇, 以及高的时间复杂度 $o(n^2)$ 。GRIDBSCAN 算法采用了基于网格的方法来改进 DBSCAN 算法, 实现了对多密度数据集的处理以及 ε 和 $minPts$ 参数的自动设置。

GRIDBSCAN 算法主要分为三个步骤, (1) 选择合适的网格划分策略将数据空间量化为网格单元集合, 并保证每个单元的密度尽可能相同。(2) 首先根据数据集中数据对象的数目 ND 来确定网格单元密度的上界 ub 和下界 lb , 然后根据 ub 和 lb 的值来识别网格集合中密集网格集合 DC 和稀疏网格集合 SC , DC 和 SC 的描述如式 2-9 所示。接下来, 针对 DC 和 SC 中的网格单元 C_i , 分别计算它们的 ε_i , $minPts_i$, ND_i (i^{th} 网格中包含的数据点的数目) 和 $indexD_i$ (i^{th} 网格的索引)。对于 $(\forall C_i)(C_i \in SC)$, 根据式 2-10 和 2-11 调整它们的 ε_i 和 $minPts_i$ 的值。然后, 类似于 CLIQUE 算法通过网格单元密度的相似性来合并网格单元一样, GRIDBSCAN 算法合并网格单元通过网格单元之间 ε 值的相似性, 合并的具体方法是通过直接 ε 可达 (对于给定的阈值 θ , 如果网格 C_j 从网格 C_i 直接 ε 可达, 则网格 C_j 是网格 C_i 的邻居网格并且它们之间满足式 2-12 所示的条件) 和 ε 可达的概念。接下来更新合并后的网格的 ε_i , $minPts_i$, ND_i 和 $indexD_i$ 。(3) GRIDBSCAN 算法根据步骤 (2) 中识别的不同的 ε_i 和 $minPts_i$ 的值, 利用 DBSCAN 算法对数据集中的数据点进行聚类。

$$\begin{aligned}
lb &= \min(0.01 \times ND, 10), ub = \min(0.05 \times ND, 30) \\
SC &= \{C_x | C_x \in C, \text{ and } lb < C_x.counts < ub\} \quad (2-9) \\
DC &= \{C_x | C_x \in C, \text{ and } ub \leq C_x.counts\} \\
C_i.\varepsilon_i &= \min(C_j.\varepsilon_j), C_j \in DC, \text{ and } C_j \in NEG(C_i) \quad (2-10)
\end{aligned}$$

$$C_i.minPts_i = \max(C_j.minPts_j), C_j \in DC, \text{ and } C_j \in NEG(C_i) \quad (2-11)$$

$$1 - \frac{\min(C_i.\varepsilon_i, C_j.\varepsilon_j)}{\max(C_i.\varepsilon_i, C_j.\varepsilon_j)} < \theta \quad (2-12)$$

GRIDBSCAN 算法的时间主要耗费在计算 DC 和 SC 网络的 ε_i 和 $minPts_i$ 上, 该计算过程由于要计算网格内数据点之间的平均距离, 因此该过程的时间复杂度是 $o(NC \times ND_i^2)$, 其中 ND_i 表示 i^{th} 网格中包含的数据点的数目。在最坏情况下, 调整稀疏网络的属性 ε_i 和 $minPts_i$ 的时间复杂度是 $o(|SC| \times (|SC| + |DC|))$ 。在最好情况下 (为网格建立空间索引结构), 根据 ε 可达的概念合并网格的时间复杂度是 $o(NC \times \log^{NC})$, 而 DBSCAN 算法利用密度可达的概念合并数据点的时间复杂度是 $o(ND \times \log^{ND})$ ($NC \ll ND$)。在步骤 (2), GRIDBSCAN 算法总的复杂度是 $o(NCM \times ND \times \log^{ND})$, 显然, GRIDBSCAN 算法的时间复杂度高于 DBSCAN 算法。实验结果表明, GRIDBSCAN 在处理多密度数据集时能够将数据对象分配到正确的簇, 在聚类准确性方面明显优越于 DBSCAN。但是, GRIDBSCAN 具有高的时间复杂度, 因此, 其特别适合于处理中小规模的空间数据库。

3. GMDBSCAN (Multi-Density DBSCAN Cluster Based on Grid) 算法[19]。GMDBSCAN 算法基于网格结构和空间索引技术, 对 DBSCAN 算法做了改进, 主要解决了 DBSCAN 算法存在的两个问题, 不能准确地处理多密度数据集和高的时间复杂度。GMDBSCAN 算法的主要实现步骤包括 4 步。(1) 将数据空间划分为网格, 然后将数据点投影到指定的网格并建立空间索引 SP-Tree。令数据点的数目为 n , 假定每个网格包含的平均数据点数目为 k , 则每个维将被划分为 $\lceil \sqrt[n/k] \rceil$ 段, 整个数据空间将被划分为 $(\lceil \sqrt[n/k] \rceil)^d$ 个网格。划分网格之后, 对非空网格建立空间索引 SP-Tree 树。(2) 创建位图 (bitmap)。DBSCAN 算法为了确定对象的 ε 邻域内包含的数据点数, 需要计算对象与其它任意对象之间的距离。

在 GMDBSCAN 算法中, 为了节省存储空间, 创建了位图结构 BMP (元素记为 $b_{ij}(i=1,2,\dots,n \text{ and } j=1,2,\dots,n)$), 该结构中共包含 n^2 位, 如果对象 j 在对象 i 的 ε 邻域内, 则将 b_{ij} 和 b_{ji} 记为 1, 否则将 b_{ij} 和 b_{ji} 记为 0, 这样, 当确定一个对象 (记为 k) 是否为核心对象时, 只需计算 BMP 中 k^{th} 行中 1 的数目。事实上, 在计算位图结构 BMP 中的元素 b_{ij} 时, 只需计算对象 i 与自己所在网格中的数据点以及邻居网格中的数据点之间的距离即可, 这样极大地减少了计算对象之间距离的复杂度。(3) 针对每个网格中包含的数据点, 确定其局部的 Eps 和 $MinPts$ 。 Eps 和 $MinPts$ 的计算公式如式 2-13 和 2-14 所示, 从它们可以看出, 所有的网格采取统一的 Eps , 而 $MinPts$ 因网格密度 GD 的不同而有所差异。(4) 聚类。对每个网格中的数据点, 基于参数 Eps 和特定于该网格的 $MinPts$ 使用 DBSCAN 算法对它们进行聚类, 形成许多分布的局部簇。最后, 对这些子簇进行合并, 如果两个子簇共享数据点并且它们的密度相似, 则将它们合并为一个较大的簇。

$$Eps = 2 \sqrt[4]{n/k} \quad (2-13)$$

$$MinPts = \lfloor factor * GD \rfloor \quad (2-14)$$

GMDBSCAN 算法最坏情况下的时间复杂度是 $O(n \times 3^d + (n/k) \log(n/k))$ 。GMDBSCAN 算法能够较好地处理多密度数据集, 同时也通过网格和空间索引技术降低了 DBSCAN 算法的时间复杂度。

4. GNN 算法 (Grid-based share nearest neighbor clustering algorithm) [22]。SNN 算法是处理多密度数据集的典型算法, 它通过计算数据对象之间共享 k 最近邻的数目来度量数据对象之间的相似度, 这种相似度度量方式较好地考虑了数据对象周围区域数据点的分布情况, 因此能较好地识别不同密度的簇。在 SNN 算法中, 如果对象 x_i 和 x_j 共享的 k 最近邻的数目 (记为 $shared_k_nearest_neighbors(x_i, x_j)$) 大于给定的阈值 φ 并且相互在对方的 k 最近邻集合中, 则将对象 x_i 和 x_j 分配到同一个簇中。SNN 算法的缺陷主要表现为对噪声数据敏感及高的时间复杂度 $o(n^2)$ 。

GNN 算法对 SNN 算法进行了改进。改进主要体现在两个方面: (1) GNN 算法利用了基于网格的技术将数据空间划分为网格集合并将数据对象映射到网格中。在确定网格密度阈值参数 $minPts$ 时, 采用了平均网格密度的思想, 根据网

格的统计信息，自动计算 $minPts$ 参数。 $minPts$ 参数的计算公式如式 2-15 所示，其中 N 表示数据点的数目， $Grid_Num$ 表示非空网格单元 ($G_i.counts > 0$) 的数目， $Max_Grid_Density$ 表示最大网格密度。利用 $minPts$ 参数便可以识别核心网格、提取边界数据点以及剔除噪声或离群点。(2) 计算核心网格的中心点，并用中心点来代替网格单元。根据输入参数 k (最近邻的数目) 和 $Shared_Points$ (共享数据点数)，利用 SNN 算法对这些中心点进行聚类，如果网格 G_i 的中心点 $G_i.center$ 和网格 G_j 的中心点 $G_j.center$ 属于相同的簇，则将它们所对应的网格 G_i 和 G_j 中的数据点划分到相同簇中，从而实现对原始数据集的聚类目的。

$$minPts = \left\lfloor \frac{N/Grid_Num + Max_Grid_Density}{2} \right\rfloor \quad (2-15)$$

GNN 算法的时间复杂度是 $o(N + m^d + NCG^2)$ ，其中 N 是数据点的数目， m 是每一维划分网格的数目， d 是数据集的维数目， NCG 是核心网格的数目。而 SNN 算法的时间复杂度是 $o(N^2)$ 。由于 $NCG \ll N$ ，因此 GNN 算法的时间复杂度低于 SNN 算法。

2.4 本章小结

本章主要介绍了聚类分析的概念、过程、要求以及方法，而方法的介绍以基于网格的聚类分析方法为重点。首先介绍了聚类分析的概念，以及聚类分析过程的 5 个处理步骤，详细讨论了每个处理步骤中的关键技术。其次，描述了不同的应用对聚类分析算法提出的特定要求，并辅以实例说明；在分析聚类分析算法要求的基础上，引出了聚类分析算法面临的挑战。最后，介绍了基于网格的聚类分析算法的基本思想和原理，并详细分析和讨论了四种典型的算法，包括 CLIQUE、GRIDBSCAN、GMDBSCAN 和 GNN 算法，这些算法都从不同的角度阐释了基于网格的聚类算法在聚类分析中的优势。

第三章 基于网格的并行聚类算法： PGMCLU

第二章中从实现机制和特性方面对目前最新的基于网格的聚类算法进行了分析，本章我们设计和实现了一个并行的基于网格的多密度聚类算法（Parallel Grid-based Clustering Algorithm for Multi-density Datasets，简称 PGMCLU）。首先阐述了算法的总体框架，其次描述了算法涉及到的相关概念，然后具体地描述了算法的实现过程，最后对算法性能进行了详细的理论分析及实验验证。

3.1 算法总体框架

在本节，我们将对 PGMCLU 算法的总体框架进行描述。Algorithm 3-1 给出了该算法的总体框架结构。PGMCLU 算法的输入包括：待聚类的数据集 D ，并行计算节点数目 n ，参考维的索引 $refDim$ 。输出结果是代表最终簇的那些网格集合 CC ，对于这些 CC ，包含在这些 CC 中的数据点构成了最终簇。PGMCLU 算法从总体上可以分为 5 个重要的处理阶段。

Algorithm 3-1: PGMCLU($D, n, refDim$)
Input: the dataset D , the number of nodes n , the index of $refDim$.
Output: cluster cells CC .
Procedure:

- (1) **FOR** each node i **DO BEGIN**
- (2) $P \leftarrow Data_Partition(I_{ref}, L, n)$; // I_{ref} is the length of dimension $refDim$;
- (3) // L is the length of interval for grid.
- (4) $SP_Tree \leftarrow Construct_SP_Tree(D_i)$;
- (5) $Local_clustering(SP_Tree, flag, clusterID)$; // $flag=2, clusterID=0$;
- (6) **IF** (exists boundary grids in SP_Tree) **THEN**
- (7) $Handle_Boundary_Grid(SP_Tree, flag)$; // $flag=1$;
- (8) **END**
- (9) $Merge_Local_Clusters(chuInfo)$;
- (10) $Cluster_output(SP_Tree, CC)$;
- (11) **END**

(1) 数据分区阶段。对应步骤 (2)。 $Data_Partition$ 过程依据参考维的维长度 I_{ref} 、网格划分间隔长度 L （在 3.3.1 节定义 3-5 中给出了计算公式），以及并行计算的节点数目 n ，得到 i^{th} 节点在 $refDim$ 维的网格划分集合 P 。在得到网格

划分集合 P 之后, 各个节点仅仅处理满足条件 $I_{j,refDim} \in P_i$ (该公式见 3.3.1 节公式 3-8) 的数据点。

(2) 构建 SP-Tree 阶段。对应步骤 (4)。*Construct_SP_Tree* 过程针对数据集 D_i 构建 SP-Tree。SP-Tree 的叶子节点中包含了对应网格的特征向量信息 GF , GF 中包括了网格的统计摘要信息。在局部聚类的过程中, 聚类算法依据 GF 中保存的信息便可以实现对网格的聚类过程。

(3) 局部聚类阶段。对应步骤 (5) ~ (7)。*Local_Clustering* 过程是一个迭代过程, 它每次从未聚类的核心网格中 ($clusterID = 0; flag = 2$) 选择一个密度最大的网格 max_Grid 作为初始网格, 采用深度优先策略 (depth-first strategy) 不断寻找网格密度连通 $density_connected(C_i, C_j)$ (定义见 3.2 节定义 3.9) 的最大区域, 直到没有新的网格加入到簇 C_x 中。*Local_Clustering* 过程处理完所有核心网格后, *Handle_Boundary_Grid* 过程将对边界网格 ($flag = 1$) 进行处理, 它通过计算边界网格 BG_i 到其邻居核心网格 $NCG(BG_i)$ (定义见 3.3.3 节式 3-13) 的距离来确定边界网格所属的簇。

(4) 合并局部簇阶段。对应步骤 (8)。*Merge_Local_Clusters* 过程在节点之间交换簇信息 $cluInfo$ (见 3.3.4 节式 3-20), 并依据簇相似度 $similarity(c_x, c_y)$ (见 3.3.4 节式 3-16) 的值和簇连通性特征 (见 3.3.4 节式 3-18) 来更新簇的组编号 $group(c_x)$ (见 3.3.4 节式 3-19)。

(5) 输出聚类结果阶段。对应步骤 (9)。*Cluster_output* 过程将输出节点 i 中形成每个簇的网格, 而网格中包含的数据点构成了簇。

3.2 重要概念

在描述算法的具体实现过程之前, 首先给出算法中涉及到的重要概念, 其包括: d 维空间、网格、核心网格、稀疏网格、邻居网格、网格紧凑度、网格质心、网格距离、直接密度可达、密度可达、密度连通、簇、空间划分树。

定义 3.1: (d 维空间) 假定 $A = (D_1, D_2, \dots, D_d)$ 是一个包含 d 个元素的有界定义域的集合, 那么 $S = D_1 \times D_2 \times \dots \times D_d$ 被称作 d 维空间 (d -dimensional space), 其中 D_1, D_2, \dots, D_d 分别表示空间 S 的维 (属性或字段)。在 d 维空间中包含 Num 个数据点

的数据集可以表示为 $X = (x_1, x_2, \dots, x_{Num})$, 其中 $x_i = (x_{i1}, x_{i2}, \dots, x_{id})(i = 1, 2, \dots, Num)$, $x_{ij}(x_{ij} \in D_j)$ 表示 i^{th} 数据对象的 j^{th} 度量。

定义 3.2: (网格、核心网格和稀疏网格) 划分数据空间 S 的每个维为 m 个等间隔, 则整个数据空间 S 被划分为 m^d 个不相交的 d 维网格单元 (grid 或 cell), 每个网格可以被描述为 (u_1, u_2, \dots, u_d) , 其中 $u_i = [l_i, h_i)(i = 1, 2, \dots, d)$ 是一个左闭右开的区间。每个网格也可以被表示为, 其中 $c_i = (I_{i1}, I_{i2}, \dots, I_{id})$, $I_{ik}(k = 1, 2, \dots, d)$ 表示 i^{th} 网格在 k^{th} 维的索引。若包含在一个网格内的数据点总数超过输入参数 $minPts$, 则定义该网格为核心网格 (core grid), 否则定义为稀疏网格 (sparse grid)。

定义 3.3: (邻居网格, neighbor grid) 如果网格 $c_j = (I_{j1}, I_{j2}, \dots, I_{jd})$ 是网格 $c_i = (I_{i1}, I_{i2}, \dots, I_{id})$ 的邻居, 记作 $Neighbor(c_i, c_j)$, 那么 c_i 和 c_j 或者存在一个公共面, 即存在 $d-1$ 维 (假定 $d-1$ 维分别是 $1, 2, \dots, d-1$) 满足 $I_{jk} = I_{ik}(k = 1, 2, \dots, d-1)$ 及 $|I_{jd} - I_{id}| = 1$ 。或者 c_i 和 c_j 至少有一个公共点, 即满足 $(\forall k)(|I_{jk} - I_{ik}| \leq 1)$ $k = 1, 2, \dots, d$ 。通常, 网格 c_i 的邻居网格集合可以表示为 $Neighbors(c_i)$ 。

定义 3.4: (网格紧凑度, grid compactness) 网格紧凑度被定义为如下公式:

$$compactness = \sqrt{\frac{\sum_{i=1}^m \sum_{j=1}^m \|x_i - x_j\|^2}{m(m-1)}} \quad (3-1)$$

其中 m 表示包含在给定网格内部的数据点的总数。网格的紧凑度表示网格内部任意两个数据点之间的平均距离 (average coupled distance), 其体现了网格内部数据点之间的紧密程度。

定义 3.5: (网格质心, grid centroid) 网格的质心被定义为如下公式:

$$centroid_{kj} = \frac{\sum_{i=1}^m x_{ij}}{m} \quad (1 \leq j \leq d) \quad (3-2)$$

其中, $centroid_{kj}$ 表示 k^{th} 网格的质心在 j^{th} 维的坐标, m 表示包含在 k^{th} 网格内的数据点的总数。

定义 3.6: (网格之间的距离, distance between grids) 网格之间的距离可以用如下公式描述:

$$d(c_i, c_j) = \sqrt{(\text{centroid}_{i1} - \text{centroid}_{j1})^2 + \dots + (\text{centroid}_{id} - \text{centroid}_{jd})^2} \quad (3-3)$$

任意两个网格之间的距离实际上是通过这两个网格的质心之间的距离来度量的。

centroid_{ik} ($k=1, 2, \dots, d$) 表示 i^{th} 网格的质心在 k^{th} 维的坐标。

定义 3.7: (直接密度可达, directly density-reachable) 给定两个核心网格 c_i 和 c_j ,

如果它们满足以下条件:

$$(1) \quad d(c_i, c_j) \leq \delta$$

$$(2) \quad 1 - \frac{\min(\text{compactness}_i, \text{compactness}_j)}{\max(\text{compactness}_i, \text{compactness}_j)} < \varphi \quad (3-4)$$

$$(3) \quad c_j \in \text{Neighbors}(c_i)$$

则称网格 c_j 从网格 c_i 出发是直接密度可达的, 记作

$\text{directly_density_reachable}(c_i, c_j)$ 。其中 δ 和 φ 都是用户预先设置的参数。

定义 3.8: (密度可达, density-reachable) 对于给定的参数 δ 和 φ , 如果存在一个

网格序列 $c_1, c_2, \dots, c_k, c_{k+1}, \dots, c_n$, $c_1 = c_i$, $c_n = c_j$, 并且

$\text{directly_density_reachable}(c_k, c_{k+1})$ ($1 \leq k \leq n-1$), 则称网格 c_j 是从网格 c_i 关于

参数 δ 和 φ 密度可达的, 记作 $\text{density_reachable}(c_i, c_j)$ 。

定义 3.9: (密度连通和簇, density-connected and cluster) 对于网格 c_i 和 c_j , 如果

存在网格 c_k , 对于给定的参数 δ 和 φ 满足条件 $\text{density_reachable}(c_k, c_i)$ 和

$\text{density_reachable}(c_k, c_j)$, 则称网格 c_j 从网格 c_i 关于参数 δ 和 φ 是密度连通的。

通常地, 簇被定义为基于密度连通的最大密度连通网格的集合。

定义 3.10: (空间划分树, Spatial Partition Tree, 简称 SP-Tree[41]) 在基于网格

的聚类算法中, SP-Tree 是一种非常重要的空间索引结构, 其将非空网格单元集合按照它们的

空间位置结构组织成一种树形结构, 具有结构简单、搜索效率高及可扩展性强等特点。在 d 维空间 S 中, 数据集 X 在网格划分集合 P 上的分布所形成的 SP-Tree 的结构可以描述如下:

(1) SP-Tree 有且仅有一个根节点, 共包含 $d+1$ 层。

(2) 数据集 X 中的每一维在 SP-Tree 中都有其对应层。SP-Tree 的 $(d+1)^{\text{th}}$ 层节点对应所有非空的网格单元。

(3) 在 SP-Tree 结构中, 除 $(d+1)^{\text{th}}$ 层 (通常称作叶子节点层, leaf node layer) 外, i^{th} ($i=1,2,\dots,d$) 层 (通常称作非叶子节点层, non-leaf node layer) 由许多内部节点构成, 在 i^{th} 层其结构可以被描述为 $(c_{ji}, pnextLay)$, 其中 c_{ji} 表示网格 c_j 在 i^{th} 维的索引, $pnextLay$ 是一个指针, 在 d^{th} 层, 其指向叶子节点, 而在其它层, 其指向 $(i+1)^{\text{th}}$ 层节点。

(4) 从根节点到叶子节点的每条路径对应于一个非空网格单元。

3.3 算法详细描述

在 3.1 节对算法的总体框架的描述中, 我们将算法的实现过程主要分为四个阶段, 分别是数据分区 (Data Partition)、构建 SP-Tree (Construct SP-Tree)、局部聚类 (Local Clustering) 和合并局部簇 (Merging Local Clusters)。在本节, 我们将围绕这四个阶段, 对算法实现过程展开详细的描述。

3.3.1 数据分区

数据分区是该算法并行计算的关键, 其主要任务是将待聚类数据按照特定的分区策略进行划分, 以便实现算法对数据的并行处理。对于聚类分析算法而言, 数据分区的质量好坏将直接影响并行算法运行时节点之间的通信成本, 以及最终聚类结果的聚类质量, 因此, 选择适当的数据分区策略对于基于数据并行的并行聚类算法是非常重要的。

在该算法中, 我们首先将数据空间量化为有穷数目的网格, 网格划分的关键是确定各维的划分长度。PGMCLU 算法采用统一的间隔长度将数据空间的各维划分为网格单元, 这样保证了整个数据空间中所有网格的体积相同。间隔长度的确定利用公式 (3-5) 来进行计算, 其中 Num 表示数据点的总数, $coefficient$ ($coefficient \in Z^+$) 称为调整系数 (adjustment coefficient), 其值可由用户根据具体的应用需求进行适当调整, d 表示维的总数, $I_i = [high_i - low_i]$ ($i=1,2,\dots,d$) 表示数据空间中 i^{th} 维的长度。

$$L = \left[\sqrt[d]{\frac{I_1 \times I_2 \times \dots \times I_d}{Num / coefficient}} \right] \quad (3-5)$$

其次，我们从维集合中选择一个维作为数据分区的参考维（reference dimension），记作 $refDim$ 。选定 $refDim$ 维后，各个节点将根据 $refDim$ 维的网格数目 $NG[refDim]$ 和并行计算的节点数目 $numprocs$ 来确定 $refDim$ 维的一个网格划分集合 P ， $|P|$ 的值可以通过公式 (3-6) 计算得到， P 中的每一个元素 P_i 是一个索引集合，记作 $P_i = \{I_1, I_2, \dots\}$ ， P_i 中的每一个元素表示 i^{th} 个节点要处理的网格在 $refDim$ 维的索引， $|P_i|$ 的计算公式如式 3-7 所示。

$$|P| = \min(numprocs, NG[refDim]) \quad (3-6)$$

```

IF numprocs ≤ NG[refDim] THEN
  IF NG[refDim] / numprocs ≠ 0 THEN
    { |Pi| = NG[refDim] / numprocs (1 ≤ i < numprocs)
      |Pi| = NG[refDim] % numprocs (i = numprocs) }
    ELSE |Pi| = NG[refDim] / numprocs
  IF numprocs > NG[refDim], THEN |Pi| = 1 (i = 1, 2, ..., NG[refDim])

```

最后，在数据分区的过程中， i^{th} 个节点仅仅处理数据集中数据点所属的网格单元在 $refDim$ 维的索引在 P_i 集合中的数据点。形式化地，假定数据点用 x_j 表示， x_j 所处的网格单元用 $c_k = (I_{j1}, I_{j2}, \dots, I_{j,refDim}, \dots, I_{jd})$ 来表示，则 i^{th} 个节点处理的数据点满足公式 (3-8) 所给出的条件。Algorithm 3-2 详细描述了数据分区过程的实现机制。

$$I_{j,refDim} \in P_i \quad (3-8)$$

Algorithm 3-2: Data Partition. Calculate the left index *celll*, and the right index *cellr* in reference dimension *refDim* for each node.

Input:
sampleFile : the file name of sample file;
configInfoFile : the file name of configuration information;

Procedure:

- (1) readConfigInfo(*configInfoFile*); // get some initial input arguments for clustering
- (2) MPI_Comm_rank(MPI_COMM_WORLD, &*myid*);
- (3) MPI_Comm_size(MPI_COMM_WORLD, &*numprocs*);
- (4) Get the number of grids for each dimension, and put them into the array *NG*;
- (5) IF (*numprocs*>1) THEN // i.e. the program runs in more than one node
- (6) IF (*myid* < (*numprocs*-1)) THEN
- (7) *NG[refDim]* /= *numprocs*;
- (8) Get *celll*= *myid** *NG[refDim]*, *cellr*=(*myid*+1)* *NG[refDim]*-1 in dimension *refDim*;
- (9) ELSE
- (10) *celll*=*myid**(*NG[refDim]*/*numprocs*);
- (11) *NG[refDim]*=(*numprocs*-1)* (*NG[refDim]*/ *numprocs*);
- (12) Get *cellr*=*celll*+*NG[refDim]*-1 in dimension *refDim*;
- (13) ENDF
- (14) ELSE // the program runs only in one node
- (15) Get *celll*=0, and *cellr*= *NG[refDim]*-1 in dimension *refDim*;
- (16) ENDF
- (17) Get data points that satisfy the index in *refDim* between *celll* and *cellr* for each node,
- (18) and construct SP-Tree;

3.3.2 构建 SP-Tree

在基于网格的聚类算法中，SP-Tree[41]是一种非常高效的索引结构，它描述了数据空间中网格之间的空间结构。在传统的 SP-Tree 索引结构中，叶子节点的结构是非常简单的，对于网格 c_k ，其所包含的信息如公式 (3-9) 所描述。 c_k 只包含了网格中包含的数据点 $\{x_i, x_j, \dots, x_m\}$ 和数据点的总数 *counts*。简单的网格描述结构虽然简化了聚类结果的生成过程，但是影响了最终聚类结果的质量。

$$c_k = \left\{ \{x_i, x_j, \dots, x_m\}, counts \right\}, \left| \{x_i, x_j, \dots, x_m\} \right| = counts \quad (3-9)$$

$$\text{in which } x_i = (x_{i1}, x_{i2}, \dots, x_{id}), 1 \leq i \leq m^d$$

在该算法中，我们提出利用网格特征值 (grid feature, 简称 GF) 来描述网格信息。GF 汇总了网格的摘要信息。GF 定义如式 (3-10) 所示，其中 *counts* 是网格中数据点的数目，*compactness* 是网格的紧凑度，*centroid* 是网格的质心，*flag* 标记网格的类型 (0: 稀疏网格, 1: 边界网格, 2: 核心网格)，*clusterID* 是网格的簇编号，*points* 是网格中所包含的数据点。GF 实际是对给定网格的信息

的统计汇总, 这样算法在聚类时所需要的一切度量值都可以通过网格特征值计算。

$$GF = \langle counts, compactness, centroid, flag, clusterID, points \rangle \quad (3-10)$$

对于聚类算法而言, 通常要求用户设置一个或多个参数, 这也就是聚类算法的参数选择特性。由于参数的选择可能是困难的, 因此, 通常的要求是“参数越少越好”。具体地讲, 在基于网格的聚类算法中, 为了有效识别核心网格以及离群点(outliers), 传统的算法通常需要用户输入网格密度阈值参数 $minPts$ 。然而, 要求用户根据领域知识设置聚类初始输入参数将带来两方面问题, 一方面聚类结果对于输入参数十分敏感, 参数设置的细微差异将显著改变聚类结果, 另一方面, 选择合适的参数值可能是困难的, 尤其是对海量数据集和高维数据集来说更是如此。因此, 如何根据数据的分布特征自动地确定聚类算法的参数已经成为用户对聚类算法特性的重要要求。在该算法中, 我们利用公式 (3-11) 实现了 $minPts$ 参数的自适应设置, 其中 Num 是数据集中数据点的总数, $gridNum$ 是非空网格的总数, max_Grid 是非空网格中特征值 $counts$ 最大的网格的 $counts$ 值。从公式 (3-11) 可以看出, $minPts$ 参数的计算利用了统计学的方法, 根据非空网格中包含的平均数据点数来刻画 $minPts$ 。

$$minPts = (\lceil Num / gridNum \rceil + max_Grid) / 2 \quad (3-11)$$

3.3.3 局部聚类

在算法执行的第二阶段, 算法通过对数据点的处理在内存中建立和维护了 SP-Tree 结构, 该结构保存了网格的特征值信息以及网格的空间位置结构。基于 SP-Tree 结构, 算法将对网格进行局部聚类, 并输出聚类结果。局部聚类算法(LC)的伪代码如 Algorithm 3-3 所示。

LC 算法的输入是 SP-Tree, 输出则是能够代表簇的网格集合 $Cells$, 这些 $Cells$ 中包含的数据点构成了最终簇的聚类结果。该算法主要分为三个处理阶段:

(1) 选取最大密度网格。对应步骤 (3)。 $Select_Initial_Grid$ 过程从当前未聚类的核心网格中选择网格密度最大的网格, 并将其作为当前聚类的初始网格 IG , 即 IG 满足公式 (3-12) 所描述的条件。

$$IG \in \{C_i | C_i.flag = 2, C_i.clusterID = 0\}, \text{ and } IG.counts = \max \{C_i.counts\} \quad (3-12)$$

(2) 聚类核心网格。对应步骤(5)。将 IG 作为当前聚类初始网格, 即 $C_i = \{IG\}$, 采用深度优先策略, 从邻居网格集合中寻找直接密度可达网格, 并将它们加入到 C_i 集合中, 直到没有新的未处理的核心网格加入到 C_i 。这一过程是一个迭代的过程, 通过不断发现直接密度可达的网格来扩展当前簇。当簇 C_i 形成之后, 算法检查网格集合中是否还存在未处理的核心网格, 如果存在, 算法将重复处理阶段(1)和(2), 直至所有的核心网格都被处理。

(3) 处理边界网格。对稀疏网格的处理也是基于网格的聚类分析算法的一个重要内容。边界网格是一种特殊的稀疏网格, 特殊之处在于其邻居网格中存在核心网格。由于边界网格中可能存在簇的边界数据点, 因此, 对边界网格的处理将有效提高聚类的精确性。在该算法中, 对于每个边界网格 BG_i , 计算其与邻居核心网格集合中各个网格之间的距离, 并从中选择最短距离 min_dis , 即 min_dis 的取值满足公式(3-13)。如果 min_dis 满足公式(3-14)中给定的条件, 则将边界网格中的数据点划分到与边界网格之间的距离最近的邻居核心网格所属的簇中。其中, d 是数据集中维数目, L 是网格的间隔长度。

$$min_dis = \min \left\{ dist(BG_i, NCG_j) \mid NCG_j \in Neighbors(BG_i), \text{and } NCG_j.flag = 2 \right\} \quad (3-13)$$

$$min_dis \leq \sqrt{d}L \quad (3-14)$$

Algorithm 3-3: LOCALCLUSTERING (SP-TREE)

Input: *SP-Tree*

Output: clustered *SP-Tree*

Procedure:

```

(1)  FOR each node  $i$  DO BEGIN
(2)    WHILE exist non-clustered core grids DO BEGIN
(3)       $IG \leftarrow Select\_Initial\_Grid(SP-Tree, flag, clusterID)$ ; //  $flag=2, clusterID=0$ ;
(4)      DO
(5)         $Extend\_Cluster(SP-Tree, IG)$ ;
(6)      WHILE (no new grid is added to current cluster )
(7)      END
(8)    END
(9)     $Handle\_Boundary\_Grid(SP-Tree, flag)$ ; //  $flag=1$ ;
(10)  END

```

3.3.4 局部聚类合并

在局部聚类阶段, 基于 SP-Tree 结构, 各个节点通过三个处理阶段得到了局

部簇。在局部聚类合并阶段，算法将合并每个数据划分的局部聚类结果，以便得到全局的聚类结果。首先，我们定义簇密度（cluster density）和簇相似度（cluster similarity）的概念。簇密度 $density(c_x)$ 由式（3-15）定义，其利用统计的方法计算簇中所包含网格的平均密度（即平均数据点数）。簇相似度 $similarity(c_x, c_y)$ 由式（3-16）定义，可以用来评估簇 c_x 和 c_y 之间的相似度。

$$density(c_x) = \frac{\text{number of data points in cluster}}{\text{number of grids in cluster}} \quad (3-15)$$

$$similarity(c_x, c_y) = \frac{\min(density(c_x), density(c_y))}{\max(density(c_x), density(c_y))} \quad (3-16)$$

簇相似度的定义为合并簇提供了重要的参考准则。为了方便说明簇的合并过程，这里给出簇的编号结构 $Cno(C_x)$ ，如式（3-17）所示， $clusterID(C_x)$ 表示簇 c_x 的簇编号， $group(c_x)$ 表示簇 c_x 的组编号。初始时， $group(c_x)$ （或 $group(c_y)$ ）被初始化为 $clusterID(C_x)$ （或 $clusterID(C_y)$ ），随着簇的不断合并，它们的值得到更新。如果簇 c_x 和 c_y 的相似度 $similarity(c_x, c_y)$ 大于指定的阈值 λ ，并且它们之间是连通的（定义如式（3-18）所示），那么将合并簇 c_x 和 c_y 并根据式 3-19 更新簇的组编号 $group(c_x)$ 和 $group(c_y)$ 。通过在邻近节点（adjacent node）之间交换簇信息 $cluInfo$ ， $cluInfo$ 的结构如式 3-20 所示，我们将得到全局的聚类结果。其中， $density(C_x)$ 表示簇 c_x 的簇密度， $clusterID(C_x)$ 表示簇 c_x 的簇编号， $BPG(C_x)$ 表示簇 c_x 的边界划分网格信息。局部聚类合并算法的详细描述如 Algorithm 3-4 所示。

$$Cno(C_x) = \langle clusterID(C_x), group(c_x) \rangle \quad (3-17)$$

$$NCG(C_i) = \{C_j \mid C_j \in Neighbors(C_i), \text{ and } C_j.flag = 2\} \quad (3-18)$$

$$(\exists C_i)(C_i \in C_x) \text{ and } (\exists C_j)(C_j \in NCG(C_i) \wedge C_j \in C_y)$$

$$group(c_x) = \min(group(c_x), group(c_y)), group(c_y) = \min(group(c_x), group(c_y)) \quad (3-19)$$

$$cluInfo(C_x) = \langle density(C_x), clusterID(C_x), group(C_x), BPG(C_x) \rangle \quad (3-20)$$

Algorithm 3-4: mergeLocalClu**Input:** λ : the threshold of cluster density for merging two clusters;**Procedure:**

```

(1) MPI_Comm_rank (MPI_COMM_WORLD, &myid);
(2) MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
(3) IF (numprocs>1) THEN
(4)   IF (myid==0) THEN
(5)     MPI_Send array PcluInfo to node myid+1; //PcluInfo saves cluster information;
(6)   ELSE
(7)     MPI_Recv the array PcluInfo from node myid-1, and put them into array recvBuf;
(8)     Merge (PcluInfo, recvBuf,  $\lambda$ );
(9)     update the group no of non-merged clusters in current node;
(10)    IF (myid<(numprocs-1)) THEN
(11)      MPI_Send the array PcluInfo that has been modified to node myid+1;
(12)    ENDIF
(13)  ENDIF
(14)ENDIF

```

3.4 PGMCLU 算法性能分析

在本节，将对 PGMCLU 算法的性能展开分析，主要包括时间复杂度（time complexity）和加速比（speedup）。

假定 N 表示样本数据集 S 中数据点的数目， d 是维的数目， p 是并行计算节点数目， k 是参考维的索引， $H_i (i=1,2,\dots,d)$ 是 i^{th} 维的网格数目。在数据划分阶段，每个节点计算 $refDim$ 的网格划分集合 P 所需的计算时间复杂度是 $o(\alpha)$ ，其中 α 是一个常数；计算 $H_i (i=1,2,\dots,d)$ 的时间复杂度是 $o(d)$ 。在构建 SP-Tree 阶段，最坏情况下（即网格集合中不存在非空网格），构建 SP-Tree 的时间复杂度是 $o(N_i \times (\log_2^{H_1} + \log_2^{H_2} + \dots + \log_2^{H_k/p} + \dots + \log_2^{H_d}))$ ，其中 N_i 表示包含在 $i^{\text{th}} (i=1,2,\dots,p)$ 个节点中的数据点的数目，由于 d 和 $H_i (i=1,2,\dots,d)$ 是常数，因此构建 SP-Tree 这个过程的时间复杂度可以近似地表示为 $o(N_i)$ 。理想情况下，如果每个节点包含的数据点的数目近似相同，则构建 SP-Tree 的时间复杂度是 $o(N/p)$ 。确定网格特征向量 GF 中 $flag$ 特征值的时间复杂度是 $o((\prod_{i=1}^d H_i)/p)$ 。确定网格特征向量 GF 中 $compactness$ 的值是构建 SP-Tree 过程中最耗时的一个步骤，时间复杂度是 $o(NC_i \times ND_i^2)$ ，其中 NC_i 是 i^{th} 节点中包含的核心网格的数目（ NC_i 的上界是 $(\prod_{i=1}^d H_i)/p$ ）， ND_i 表示 i^{th} 核心网格中数据点的数目， ND_i^2 的

出现是由于要计算核心网格中数据点之间的平均距离，因此，较小的 ND_i 有利于降低算法的时间复杂度。当局部聚类时，在最坏情况下（网格集中的所有网格都是核心网格，即 $C_i.flag = 2$ ），计算密度连通网格最大区域的时间复杂度是 $o(((\prod_{i=1}^d H_i) / p) \times 2d \times (\log_2^{H_1} + \dots + \log_2^{H_1/p} + \dots + \log_2^{H_d}))$ ，其中 $2d$ 表示网格 C_i 的邻居网格的数目（其根据定义 3.3 中“存在一个公共面”这种情况而计算得到）， $\log_2^{H_1} + \dots + \log_2^{H_1/p} + \dots + \log_2^{H_d}$ 表示查找网格 C_i 的邻居网格 C_j ($C_j \in Neighbors(C_i)$) 所需的时间复杂度，由于 d 和 $H_i (i=1,2,\dots,d)$ 都是常数，因此，该时间复杂度可以近似地表示为 $o((\prod_{i=1}^d H_i) / p)$ 。在局部聚类合并过程中，节点之间交换簇信息 $cluInfo$ 所需的时间复杂度是 $o(p-1)$ 。

综合以上讨论，该算法总的时间复杂度是 $o(d + N_i + ((\prod_{i=1}^d H_i) / p + NC_i \times ND_i^2 + p))$ 。理想情况下，若数据分区过程中各个节点所分配的数据点的数目相同，则时间复杂度可以近似表示为 $o(d + N/P + ((\prod_{i=1}^d H_i) / p + NC_i \times ND_i^2 + p))$ 。令 $p=1$ ，可以计算出算法在单个节点运行时的时间复杂度为 $o(d + N + ((\prod_{i=1}^d H_i) + NC_i \times ND_i^2))$ 。从以上分析可以得出结论：从理论上分析，该算法具有近似线性的加速比。

3.5 PGMCLU 算法实验结果与性能评价

所有实验均在甘肃省计算中心曙光 4000L 集群系统上进行，操作系统为 Linux。该集群系统包含 48 个 2.2GHz CPU 和 24 个 2.2GHz 双核 CPU，14 个 146G Ultra30 SCSI 硬盘。该算法使用 C+MPI 语言实现，MPI 库的版本是 mpich2-1.0.7。

在下面的实验中，我们将选取不同类型（真实数据集和模拟数据集）的代表性数据集，从聚类准确性、相对加速比和效率三个方面来对算法性能进行验证和评价。

3.5.1 聚类准确性分析

聚类准确性分析 (analysis of accuracy) 是指通过输入不同类型的代表性数据

集来测试算法是否可以准确地处理不同类型的数据集。实验中选择了两个代表性数据集。

第一个数据集（简称为 DS1）来源于 CURE 算法，DS1 包含 100,000 个数据点，其中包含 1 个大的圆形簇和 2 个小的圆形簇，以及 2 个椭圆形状的簇，此外，该数据集中还分布着大量随机的噪声数据点。该数据集主要用来测试算法对噪声点的敏感性、对大型数据集以及多密度数据集的处理能力。图 3-1 给出了 PGMCLU 算法在数据集 DS1 上的聚类结果。

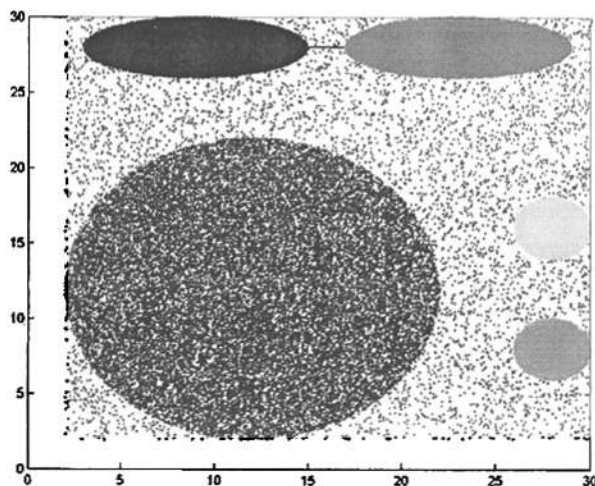


图 3-1 PGMCLU 算法在数据集 DS1 上的聚类结果

Fig. 3-1 Clusters discovered by PGMCLU on DS1

第二个数据集（简称 DS2）来源于 Chameleon 算法（Chameleon 是一种层次聚类算法，利用动态建模来确定簇对之间的相似度。与聚类分析中一些著名的算法（如 BIRCH 和 DBSCAN）相比，其在发现高质量的任意形状的簇方面具有优势）。该数据集包含 8,000 个数据点，这些数据点共形成了 6 个簇。每个簇都具有任意的形状。选择该数据集主要是用来测试算法对任意形状数据集的处理能力。图 3-2 给出了 PGMCLU 算法在数据集 DS2 上的聚类结果。

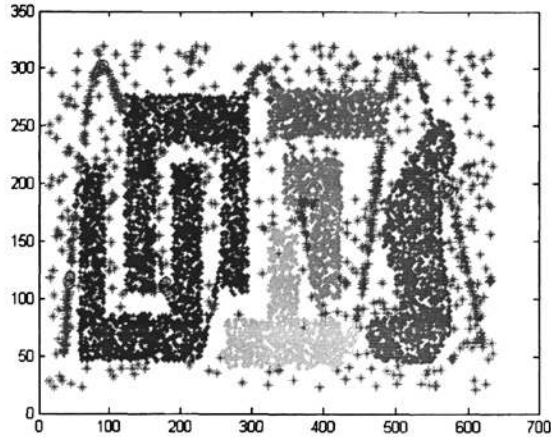


图 3-2 PGMCLU 算法在数据集 DS2 上的聚类结果

Fig. 3-2 Clusters discovered by PGMCLU on DS2

从图 3-1 和图 3-2 可以看出, PGMCLU 算法在处理多密度数据集、包含离群点的数据集、任意形状数据集和大型数据集方面, 具有较好的处理能力。

SNN (share nearest neighbours, 共享最近邻居) 算法是一种典型的针对多密度数据集的聚类算法, 该算法定义了新的相似度度量, 该度量通过计算数据点之间共享 k 最近邻居的数目来定义数据点之间的相似度, 这种相似度度量方式较好地反映了对对象的周围区域数据分布特征。表 3-1 给出了 PGMCLU 算法和 SNN 算法在聚类准确性方面的比较结果。准确性是通过算法对离群点的识别成功率来度量的。从表 3-1 可以看出, 当数据集中噪声数据的分布比较密集时, SNN 算法准确性较差, 而 PGMCLU 算法则表现出较高的准确性。

表 3-1 PGMCLU 和 SNN 算法的准确率比较结果

Table 3-1 The result of accuracy comparison for PGMCLU and SNN algorithm

Algorithm	Datasets	Accuracy (%)
PGMCLU	DS1	92
	DS2	89
SNN	DS1	71
	DS2	78

3.5.2 相对加速比分析

为了度量相对加速比 (relative speedup), 我们保持数据集的大小不变, 而改变节点的数目。当节点数目为 m 时, 相对加速比的定义如式 (3-21) 所示。

$$relative\ speedup(m) = \frac{runtime\ on\ one\ node\ for\ parallel\ clustering\ algorithm}{runtime\ on\ m\ nodes\ for\ parallel\ clustering\ algorithm} \quad (3-21)$$

针对数据集 DS1 和 DS2, 我们分别测试 PGMCLU 算法在可变节点数目上的运行时间, 并计算相对加速比。图 3-3 给出了 PGMCLU 算法在数据集 DS1 和 DS2 上的相对加速比计算结果。从图 3-3 中可以看出, 实验结果表明算法表现出了近似线性的加速比, 与理论上分析的加速比特性相一致。在数据集 DS1 上, 随着并行计算节点数目的增加, 相对加速比略低于理想的线性加速比, 这主要是受到数据分区的不均等特性和通信成本的影响。然而, 在数据集 DS2 上, 当并行计算的节点数目增加到 8 时, 相对加速比曲线呈现出明显的下降趋势。这是由于当一个并行聚类算法对大型数据集进行聚类时, 算法的计算时间远远大于通信时间, 这导致了相对较高的相对加速比; 然而, 当并行聚类算法处理规模较小的数据集时, 随着节点数目的增加, 相对于较小的计算时间而言, 算法的通信时间将在算法的整个运行时间中占据较大的比例, 而这将极大地影响算法的相对加速比。因此, 在实际的应用中, PGMCLU 算法在处理大型的数据集时将显示出较优的性能。

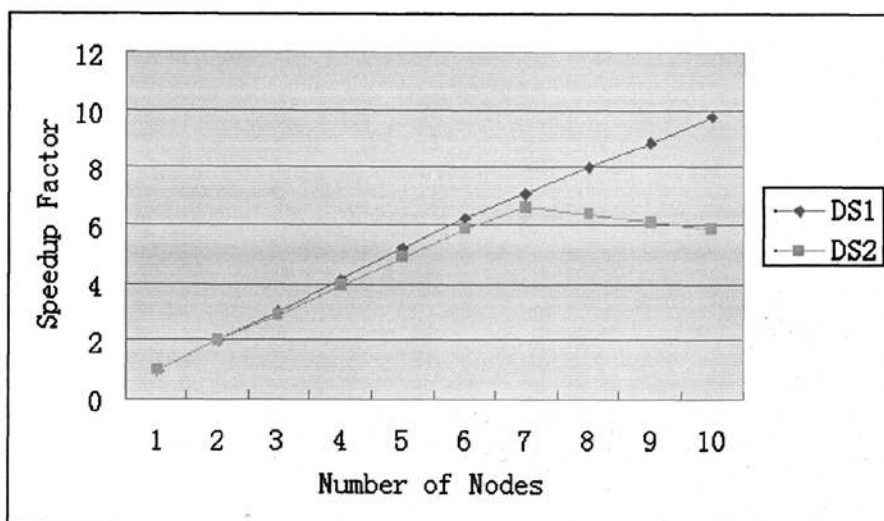


图 3-3 PGMCLU 算法在数据集 DS1 和 DS2 上的相对加速比计算结果

Fig. 3-3 The results of relative speedup for PGMCLU on DS1 and DS2

3.5.3 效率分析

对于一个并行聚类算法而言, 好的相对加速比并不能完全地反映优的节点利用率, 通过分析算法的效率, 我们可以得到并行算法运行时需要的最佳节点数目。当节点数目为 m 时, 效率的定义如式 3-22 所示。

$$efficiency(m) = relative\ speedup(m)/m \quad (3-22)$$

为了测试 PGMCLU 算法的效率, 我们利用文献[42]中提供的数据集生成算法 (由 Dave Dubin 提供的该算法的一个权威及公开的实现 “clusgen.c” 可以从网址 <http://alexia.lis.uiuc.edu/~dubin> 上下载), 生成 3 个 5 维的模拟数据集, 它们包含的数据点数分别是 40,000, 160,000, 640,000。该数据集生成算法的优点是我们可以精确描述待生成数据集的特征 (如维数目、数据集规模及噪声点数目等) 来生成尽可能多的满足指定条件的数据集, 这为数据集的生成提供了便利。图 3-4 给出了 PGMCLU 算法在 3 个模拟数据集上的效率比较结果。从该图可以看出, 对于相同的节点数目, 算法在大型的数据集上呈现出较高的效率, 这是由其较高的加速比决定的。

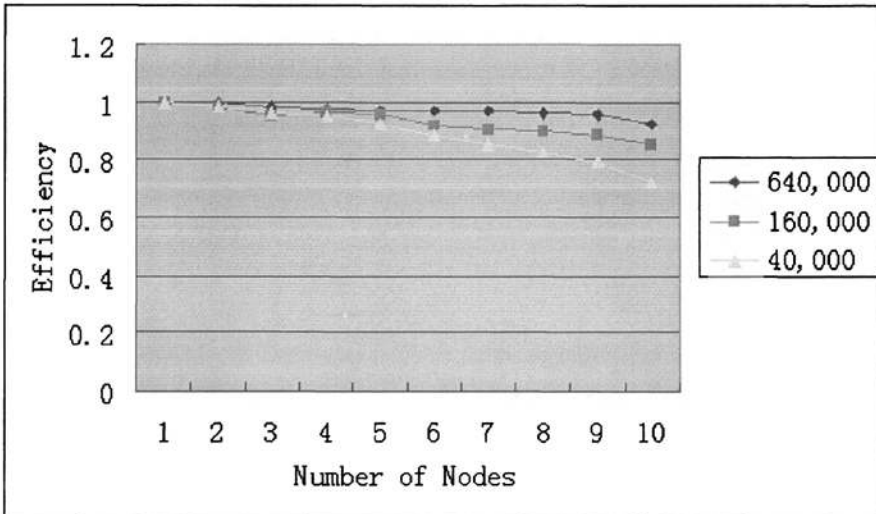


图 3-4 PGMCLU 算法在 3 个模拟数据集上的效率比较

Fig. 3-4 The efficiency of PGMCLU on three synthetic datasets

3.6 本章小结

该章介绍了聚类分析的概念、过程、特性及挑战, 提出了一种新的并行的基于网格的聚类算法 PGMCLU。该算法利用网格紧凑度来精确反映网格中数据的紧密程度, 为多密度数据集的密度度量提供了精确的计算指标; 利用网格特征值来描述网格的摘要信息, 为聚类提供了较好的参考信息; 利用 SP-Tree 结构存储网格的空间位置结构, 提高了邻居网格的检索效率。此外, PGMCLU 算法还提

出了基于 *refDim* 维的数据分区策略；基于密度连通网格的局部聚类方法；以及局部聚类合并策略。然后，我们从理论上分析了 PGMCLU 算法的时间复杂度以及加速比。最后，选取不同类型的代表性数据集对聚类算法的性能（包括聚类准确性、相对加速比以及效率）进行了验证和评价。

第四章 基于网格的数据流聚类分析算法： GC-Stream

本章将研究基于网格的数据流聚类分析算法(Grid-based Clustering algorithm for data stream, 简称 GC-Stream)。首先, 介绍数据流的概念, 以及数据流聚类分析的特性、窗口模型及典型的算法。其次, 描述 GC-Stream 算法的总体框架。接下来, 描述 GC-Stream 算法的聚类模型、网格信息衰减策略、聚类模型维护策略及聚类模型剪枝策略。最后, 分析 GC-Stream 算法的性能并给出实验验证和性能评价结果。

4.1 数据流聚类分析概述

本节将介绍数据流的概念, 详细阐释数据流聚类分析算法的特性和窗口模型, 并重点分析几种典型的数据流聚类分析算法的聚类模型和信息衰减策略。

4.1.1 数据流的概念

从广义的角度讲, 数据流(Data Stream)是指以流的形式产生的数据。从数据流挖掘的角度讲, 数据流一般指潜在无限的、持续而快速到达的具有时间顺序的数据序列。形式化地, 假定 t 表示数据到达的时间戳, x_t 表示在 t 时刻到达的数据点, 则当前时刻 T_c 时的数据流 D_{T_c} 可表示为如式 4-1 所示的形式。从数据流的定义可以看出, 数据流具有海量性、实时性、高速性等特性。随着数据流在不同的应用中的广泛生成, 越来越多的研究人员开始研究数据流挖掘技术, 包括数据流中的频繁模式挖掘、动态数据流的分类以及聚类演变数据流等。

$$D_{T_c} = \{x_i, \dots, x_t, \dots, x_j\} (i < t < j) \quad (4-1)$$

4.1.2 数据流聚类分析算法的特性

传统的聚类分析算法通常面对的是静态数据, 采用脱机的方式对静态数据进

行处理。而数据流聚类分析算法通常处理的是动态数据流，数据流的潜在无限特性以及快速到达特性决定了数据流聚类分析算法与传统聚类分析算法相比具有以下几个重要的特性[43-46]:

(1) 单遍扫描。一方面数据流是无限增长的，受内存空间的限制，内存中不可能保存所有的数据流对象；另一方面数据流聚类分析算法实时性高，不可能对海量的数据流进行多遍扫描，因此，数据流聚类分析算法要求对数据流对象进行单遍扫描。

(2) 聚类模型使用概要数据结构保存数据流对象信息，并具有增量更新特性。数据流聚类分析算法需要在内存中维护聚类模型，存储数据流对象的概要数据结构(summary data structure)，当有新的数据流对象到来时，对聚类模型进行增量更新。

(3) 数据流聚类分析算法必须实现对数据流对象的快速处理。数据流对象持续而快速地到达，数据流聚类分析算法必须对它们做出及时的处理。

(4) 数据流聚类分析算法必须使用有限的内存。这是由数据流的潜在无限性来决定的。数据流聚类分析算法必须考虑到剪枝和淘汰策略(pruning and elimination strategy)，以便优化聚类模型结构。

(5) 数据流聚类分析算法的聚类结果的近似性。由于数据流聚类分析算法不能保存数据流对象并且不能够对数据流对象实现多遍扫描，只能维护数据流的概要数据结构，因此，数据流聚类分析算法的聚类结果精度会受到影响，只能得到近似的结果。但是，在实际应用中，应该设计尽可能真实反映数据流对象特征的概要数据结构，以便提高聚类结果的精确度。

(6) 数据流聚类分析算法必须在线维护聚类模型，并能在用户要求的任意时刻及时反馈聚类结果。

(7) 数据流可能是随着时间动态演变的，因此数据流聚类算法应该实现对模型的动态更新，以此来反映数据流的演化(evolution)(或进化)规律。

4.1.3 数据流聚类分析算法的窗口模型

对于数据流聚类分析算法而言，面对持续而快速到达的数据流对象，其必须解决几个问题：如何确定对海量的数据流对象中的哪些数据流对象进行聚类分析；

如何实现对数据流对象的快速增量聚类; 如何实现对模型的动态更新来适应数据流的变化。这些问题都涉及到数据流聚类模型的问题, 为了很好地解决上述问题中的数据流对象选择和模型动态更新的问题, 数据流聚类分析算法通常采用窗口模型 (window model)。窗口是指针对特定的数据流, 数据流聚类分析算法所关注的对象的范围。窗口模型包括界标窗口模型 (landmark window model, 也称为里程碑窗口模型), 滑动窗口模型 (sliding window model), 衰减窗口模型 (damped window model) [47]。不同的窗口模型都提供了相应的窗口机制。

(1) 界标窗口模型。界标窗口模型处理的数据流对象是从数据流开始时刻 T_s 到当前时刻 T_c 的所有数据流对象, 即数据流的所有历史数据。形式化地, 可以描述为式 4-2 所示, 其中 x_{T_i} 表示 T_i 时刻到达的数据流对象。界标窗口模型中的所有数据流对象 x_{T_i} 的信息权值相同, 即 x_{T_i} 的信息权值不会因时间的推移而发生变化。基于界标窗口模型的聚类分析算法可以发现数据流的全局模式, 但是不能很好地反映数据流的进化趋势。

$$D_{T_c} = \{x_{T_s}, \dots, x_{T_i}, \dots, x_{T_c}\} (T_s < T_i < T_c) \quad (4-2)$$

(2) 滑动窗口模型。滑动窗口模型只处理当前窗口大小 (window size) 范围内的数据流对象。窗口大小的计算方式通常由两种, 包括基于时间的和基于数据流对象计数的, 因此, 滑动窗口模型也依据窗口大小的计算方式不同被分为两种类型, 包括基于时间 (time-based) 的滑动窗口模型和基于数据流对象计数 (size-based) 的滑动窗口模型。对于基于时间的滑动窗口模型, 假定当前时间戳为 T_c , 数据流的起始时间戳为 T_s , 窗口大小为 WTS , 则针对当前时间戳 T_c , 基于时间的滑动窗口模型要处理的数据流对象的集合 D_{T_c} 可以描述为式 4-3 所示形式。对于基于数据流对象计数的滑动窗口模型, 假定窗口大小为 WN , 则在时间戳为 T_c 时, 基于数据流对象计数的滑动窗口模型所要处理的数据流对象的集合 D_{T_c} 可以表示为式 4-4 所示形式。在滑动窗口模型中, D_{T_c} 所包含的数据流对象是最近一段时间内最新的数据流对象, 因此, D_{T_c} 集合中的数据流对象可以较好地反映数据流的当前趋势。滑动窗口模型中窗口大小范围内包含的数据流对象具有相同的信息权值, 窗口大小随着时间的推移不断向前滑动。对于处在滑动窗口大小范围之外的数据流对象, 滑动窗口模型将它们的信息权值当做 0, 将它们丢弃。

$$D_{T_c} = \{x_{\max(T_i, T_c - WTS)}, \dots, x_{T_i}, \dots, x_{T_c}\} (\max(T_i, T_c - WTS) < T_i < T_c) \quad (4-3)$$

$$D_{T_c} = \{x_{T_1}, \dots, x_{T_c}\} (0 \leq T_k \leq T_c, \text{ and } |D_{T_c}| \leq WN) \quad (4-4)$$

(3) 衰减窗口模型。衰减窗口模型处理的数据流对象为数据流开始时间戳 T_i 到当前时间戳 T_c 的所有数据流对象, 这与界标模型处理的数据流对象相同, 但是, 衰减窗口模型中的数据流对象的信息权值不相同, 较早到达的数据流对象具有低的信息权值, 而最新到达的数据流对象具有高的信息权值, 因此, 数据流对象对数据流的进化趋势影响具有差异性。对于数据流 D_{T_c} , 对于 D_{T_c} 中的每个数据流对象 $x_{T_i} (T_i \leq T_c)$, 其信息权值 (记作 $W(x_{T_i}, T_c)$) 依据衰减函数随着时间不断衰减, 衰减函数可以是指数函数, 如式 4-5 和式 4-6 所示。从 4-5 和 4-6 可以看出, 对于每一个新到达的数据流对象 x_{T_c} , 其信息权值是 1; 而对其它的数据流对象, 距离当前时间戳 T_c 越远, 则其信息权值越小。通常, 信息权值衰减函数可以表示为式 4-7 所示形式, 其中 b 称为衰减基, η 称为衰减生命。

$$W(x_{T_i}, T_c) = 2^{-\lambda(T_c - T_i)} (\lambda > 0) \quad (4-5)$$

$$W(x_{T_i}, T_c) = e^{-\lambda(T_c - T_i)} (\lambda > 0) \quad (4-6)$$

$$W(x_{T_i}, T_c) = b^{-\lambda\eta} (b > 1, \lambda > 0, \eta \geq 0, T_i \leq T_c) \quad (4-7)$$

4.1.4 典型的数据流聚类分析算法

典型的数据流聚类分析算法包括 STREAM、CluStream、AIN-Stream、DenStream、GD DS 等, 本节将阐述它们的基本原理, 并对它们的摘要数据结构 (或聚类模型) 做详细的分析和讨论。

(1) CluStream 算法[37]。CluStream 为聚类进化数据流 (clustering evolving data streams) 提供了一个框架模型, 其更关注数据流随时间推移的进化趋势。CluStream 算法将数据流聚类过程划分为两个阶段, 在线部分 (online component) 和离线部分 (offline component)。在线部分使用微簇结构 (Micro-clusters) 维护数据流对象的统计信息, 微簇是对相邻的一组数据流对象的描述, 微簇结构是在分层聚类分析算法 BIRCH 的聚类特征向量 (cluster feature vector) 中加入时间属

性。对于 d 维数据空间，微簇 C_i 中包含的数据流对象为 x_{T_1}, \dots, x_{T_n} ，其中 $x_{T_i} (i=1, \dots, n)$ 表示 T_i 时刻到达的数据流对象，则微簇 C_i 的特征向量 CF_i 可以描述为式 4-8 所示， CF_i 是一个 $2d+3$ 元组， n 表示微簇 C_i 中包含的数据点的数目， $k=1, 2, \dots, d$ 表示 k^{th} 维。Micro-clustering 阶段主要完成对数据流统计信息的收集，为离线部分的基于时间范围的宏聚类 and 进化分析准备统计信息。微聚类阶段根据 pyramidal time frame (倾斜式时间框架或金字塔式时间框架) 将微簇信息以快照 (snapshots) 的形式存储，金字塔式时间框架实现了不同粒度的微簇集的快照。同其它的数据流聚类分析方法类似，CluStream 算法也提供了对聚类模型的剪枝策略，包括对微聚类的删除和合并，合并策略是将距离最近的两个微簇合并。离线部分根据用户的输入信息，如时间范围 (time horizon) 和聚类粒度 (the granularity of clustering) 等，采用 macro-clustering 技术，对微簇集快照信息进行处理，并反馈结果给用户。

$$CF_i = (CF1^x, CF2^x, CF1', CF2', n)$$

$$(CF1^x)^k = \sum_{i=1}^n (x_{T_i})_k, (CF2^x)^k = \sum_{i=1}^n (x_{T_i})_k^2, k=1, 2, \dots, d \quad (4-8)$$

$$CF1' = \sum_{i=1}^n T_i, CF2' = \sum_{i=1}^n T_i^2$$

(2) AIN-Stream 算法[39]。AIN-Stream 是基于人工免疫系统 (artificial immune system) 的数据流聚类分析算法。它基于人工免疫系统的原理，将 B 细胞 (B-cell) 作为微簇，数据流对象作为 B 细胞的抗原 (antigen)。B 细胞识别区域内的所有数据流对象表示微簇中包含的数据流对象。AIN-Stream 算法的输入是数据流，而输出是由 B 细胞所构成的 B 细胞网络所表示的概要数据结构。B 细胞用 B 细胞特征向量 (B-cell feature vector, 记作 BCF) 来描述。对于 d 维数据空间，假定 B 细胞 B_j 的识别区域内包含的数据流对象集合为 x_{T_1}, \dots, x_{T_n} ， $T_i (i=1, 2, \dots, n)$ 表示时间戳， T_c 表示当前时间戳，则 B 细胞 B_j 的特征向量 $BCF(B_j)$ 可以用式 4-9 所示的形式来描述，其中 $k (k=1, 2, \dots, d)$ 表示 k^{th} 维， $Sim_{(x_{T_i}, B_j)}$ 表示 B 细胞 B_j 的识别区域内的数据流对象 $x_{T_i} (i=1, 2, \dots, n)$ 对 B 细胞 B_j 的激励度， $d_{(x_{T_i}, B_j)}$ 表示 B 细胞 B_j 的识别区域内的数据流对象 $x_{T_i} (i=1, 2, \dots, n)$ 与 B 细胞 B_j 之间的欧式距离， δ_j 表示 B 细胞 B_j 的识别区域的半径， Sim 表示 B_j 的

激励度、 AW 表示 B_j 的信息权值, T_s 表示 B_j 的创建时间, T_u 表示 B_j 的更新时间, 即 B 细胞 B_j 最后一次识别到的抗原所对应的时间戳。AIN-Stream 算法中聚类模型的剪枝策略通过对 B 细胞网络的压缩来实现, B 细胞网络的压缩主要是剔除两类 B 细胞, 分别是剔除冗余 B 细胞和激励度 Sim 最小的 B 细胞。在聚类结果输出阶段, 通过合并 B 细胞来实现聚类。

$$\begin{aligned}
 BCF(B_j) &= \langle AF1^x, AF2^x, Sim, AW, T_s, T_u \rangle \\
 (AF1^x)^k &= \sum_{i=1}^n e^{-\alpha(T_c - T_i)} (x_{T_i})_k \quad (\alpha > 0, k = 1, 2, \dots, d) \\
 (AF2^x)^k &= \sum_{i=1}^n e^{-\alpha(T_c - T_i)} (x_{T_i})_k^2 \quad (\alpha > 0, k = 1, 2, \dots, d) \quad (4-9) \\
 AW &= \sum_{i=1}^n e^{-\alpha(T_c - T_i)} \quad (\alpha > 0) \\
 Sim &= \sum_{i=1}^n Sim_{(x_{T_i}, B_j)}, \text{ and } Sim_{(x_{T_i}, B_j)} = e^{-\frac{d(x_{T_i}, B_j)^2}{\delta_j^2}} \cdot e^{-\alpha(T_c - T_i)}
 \end{aligned}$$

(3) GDDS 算法[48]。GDDS (Grid and Density clustering algorithm for analyzing Data Stream) 算法是一种基于网格和密度的数据流聚类分析算法。它将数据空间划分为网格, 然后对稠密的网格单元进行聚类。GDDS 算法的处理过程也分为在线处理和离线处理两个阶段, 在线处理阶段维护网格单元的统计信息, 离线阶段根据网格单元的统计信息进行聚类。对于 d 维数据空间, 当前时间戳为 T_c , 假定网格单元 G_k 中包含的数据流对象的集合为 x_{T_1}, \dots, x_{T_n} , $x_{T_i} (i = 1, 2, \dots, n)$ 表示时间戳为 $T_i (i = 1, 2, \dots, n)$ 时到达的数据流对象, 则网格单元 G_k 的结构 GS_k 如式 4-10 所示, 其中 $G_k[j]$ 表示网格 G_k 在 $j^{th} (j = 1, 2, \dots, d)$ 维的索引, $DF1^x$ 是一个 d 维向量, $(DF1^x)^j (j = 1, 2, \dots, d)$ 表示 $DF1^x$ 的 $j^{th} (j = 1, 2, \dots, d)$ 分量, $(DF2^x)^j$ 同理, $Count$ 表示网格 G_k 的密度 (即数据点的数目)。 GS_k 是一个 $3d + 3$ 维的向量。GDDS 算法使用哈希树 (Hash Tree) 来索引网格。离线处理过程基于在线过程的网格统计信息, 根据用户输入的参数 (如时间戳或时间戳范围), 得到数据流在某一个时间戳时的聚类结果或者某一时间范围内的聚类结果。

$$\begin{aligned}
GS_k &= \langle ID, Count, DF1^x, DF2^x, DF1^t, DF2^t \rangle \\
ID &= \{G_k[j]\} (j=1, 2, \dots, d) \\
(DF1^x)^j &= \sum_{i=1}^n (x_{T_i})_j, (DF2^x)^j = \sum_{i=1}^n (x_{T_i})_j^2 (j=1, 2, \dots, d) \quad (4-10) \\
DF1^t &= \sum_{i=1}^n T_i, DF2^t = \sum_{i=1}^n T_i^2
\end{aligned}$$

4.2 GC-Stream 算法的总体框架

GC-Stream 算法是基于网格的数据流聚类分析算法，其采用衰减窗口模型，旨在分析数据流随时间推移的演化趋势。该算法首先将数据空间量化为互不重叠的网格单元的集合；然后，建立基于 LSP-Tree 的空间索引结构，并维护网格单元的摘要统计信息，网格单元的摘要统计信息随着时间的推移不断演化；最后，聚类输出阶段依据基于网格的聚类方法对网格单元聚类并输出聚类结果，GC-Stream 算法的总体框架如 Algorithm 4-1 所示。

GC-Stream 算法的实现过程可以分为三个步骤，包括聚类模型的维护和更新阶段，聚类模型的剪枝阶段，以及聚类结果输出阶段。它们的详细描述如下：

(1) 聚类模型的维护和更新阶段。对应步骤 (1) - (5)。*Initialize_LSP_Tree* 过程根据数据流中的第一个数据流对象 x_{T_1} 初始化 LSP-Tree。*Maintain_LSP_Tree* 过程将数据流中新到达的每个数据流对象 $x_{T_i} (T_i \leq T_c)$ 插入到 LSP-Tree 中，*Update_LSP_Tree* 过程根据网格单元信息衰减策略对数据流对象 $x_{T_i} (T_i \leq T_c)$ 所对应的网格单元进行信息衰减。

(2) 聚类模型的剪枝阶段。对应步骤 (6) 和 (7)。当 LSP-Tree 的结构不能适应内存大小的限制时，*Pruning_LSP_Tree* 过程将对 LSP-Tree 进行剪枝操作。剪枝过程主要是从网格单元集合中剔除两类网格，一类是“过时”的网格，即相对于当前时间 T_c 而言，网格单元的最后更新时间与 T_c 之间的时间范围大于阈值 τ ；另一类是稀疏网格，即网格单元的密度小于给定的密度阈值 $minPts$ 。通过对“过时”网格和稀疏网格的剔除，将达到压缩 LSP-Tree 的目的。

(3) 聚类结果输出阶段。对应步骤 (8) 和 (9)。当用户要求输出聚类结果时，*Output_clusters* 过程将扫描 LSP-Tree，并对核心网格单元进行聚类，输出代表簇的网格单元集合。

Algorithm 4-1: GC-Stream (D_{T_c})
Input: data Stream D_{T_c} for timestamp T_c .
Output: cluster cells CC .
Procedure:

- (1) **IF** data stream object x_{T_i} ($T_i \leq T_c$) is first object **THEN**
- (2) $LSP_Tree \leftarrow Initialize_LSP_Tree(x_{T_i});$
- (3) **END**
- (4) **FOR** each new data stream object x_{T_i} ($T_i \leq T_c$) **DO BEGIN**
- (5) $Maintain_LSP_Tree(LSP_Tree, x_{T_i});$
- (6) $Update_LSP_Tree(LSP_Tree, x_{T_i}, T_i);$
- (7) **IF** (LSP_Tree is too big) **THEN**
- (8) $Pruning_LSP_Tree(LSP_Tree, \tau, minPts);$
- (9) **END**
- (10) **IF** (given the time stamp T_k ($T_k \leq T_c$)) **THEN**
- (11) $Output_clusters(LSP_Tree, CC);$
- (12) **END**
- (13) **END**

4.3 GC-Stream 算法的详细描述

在 4.2 节中，对算法的总体框架进行了描述，将 GC-Stream 算法的实现过程分为三个处理步骤。本节将对 GC-Stream 算法的网格单元结构、LSP-Tree 的维护和更新策略，LSP-Tree 的剪枝策略，以及聚类输出进行详细的描述。

4.3.1 网格单元特征向量描述

在 GC-Stream 算法中，对于 d 维数据空间，将根据网格的间隔长度 m 划分数据空间，将整个数据空间划分为互不重叠的网格单元集合 G ， $|G|$ 的计算公式如式 4-11 所示，其中 DL_i 表示 i^{th} ($i=1,2,\dots,d$) 维的长度。基于网格单元集合 G 构建空间索引结构 LSP-Tree，LSP-Tree 将所有非空网格单元按照它们的空间位置关系组织起来。对于数据流中的每个数据流对象，将它们投影到对应的网格单元中，网格单元中包含了它们的统计信息，用这些统计信息来代表数据流对象。在 GC-Stream 算法中，假设网格单元 G_i 中包含的数据流对象为 $x_{T_1}, x_{T_2}, \dots, x_{T_n}$ ， x_{T_i} ($i=1,2,\dots,n$) 表示时刻 T_i 到达的数据流对象，则网格 G_i 的特征向量 GF_i 如式 4-12 所示， GF_i 是 $d+4$ 维向量，其中 $DFCI^x$ 表示 G_i 的质心， $tightness$ 表示网格 G_i

的紧凑度, $counts$ 表示 G_i 中包含的对象的数目, $clusterID$ 表示 G_i 所属的簇, T_u 表示 G_i 中最近一次到达的数据流对象所对应的的时间戳, $(DFCI^x)^k (k=1,2,\dots,d)$ 表示 $DFCI^x$ 中 k^{th} 分量, $(x_{T_i})_k (k=1,2,\dots,d)$ 表示数据流对象 x_{T_i} 在 k^{th} 维的坐标, $\|x_{T_i} - x_{T_j}\|$ 表示数据流对象 x_{T_i} 和 x_{T_j} 之间的距离。

$$|G| = \prod_{i=1}^d \left\lceil \frac{D_{G_i}}{m} \right\rceil \quad (4-11)$$

$$GF_i = \langle DFCI^x, tightness, counts, clusterID, T_u \rangle$$

$$(DFCI^x)^k = \frac{\sum_{i=1}^n (x_{T_i})_k}{n} \quad (k=1,2,\dots,d) \quad (4-12)$$

$$tightness = \sqrt{\frac{\sum_{i=1}^n \sum_{j=1}^n \|x_{T_i} - x_{T_j}\|^2}{n(n-1)}} \quad (n \geq 2)$$

4.3.2 LSP-Tree 维护和更新过程

在 GC-Stream 算法中, 将实时到达的数据流对象投影到网格单元中, 网格单元的特征向量描述了网格单元中包含的数据流对象的摘要统计信息, 网格单元通过 LSP-Tree 结构组织。LSP-Tree 中主要包括非叶子节点 (non-leaf node) 和叶子节点 (leaf node) 两类, 根节点也属于非叶子节点。非叶子节点的结构如图 4-1 所示, 非叶子节点由 $GDI-list$ 构成, $GDI-list$ 是由 $GDI-Node$ 构成的链表结构, 链表结构中的 $GDI-Node$ 集合根据 $GDI-Node$ 中元素 GDI 的值升序排列, 以方便将新的网格单元快速地插入到 LSP-Tree 中。 $GDI-Node$ 包含三个元素, GDI 表示网格单元在 LSP-Tree 中当前层所对应的维上的索引, $pNextLayer$ 是一个指针, 指向下一层节点, Ptr 也是指针, 指向 $GDI-list$ 中的邻近节点。叶子节点的结构如图 4-2 所示, 叶子节点中的每一个元素与网格单元的特征向量中的每个特征值相对应。图 4-3 以二维空间为例, 给出了 LSP-Tree 的基本结构。

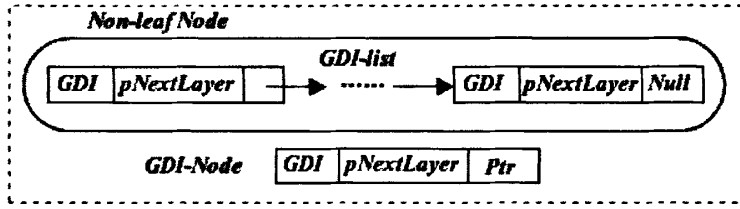


图 4-1 LSP-Tree 中非叶子节点 (Non-leaf Node) 结构示意图

Fig. 4-1 the structure of Non-leaf Node in LSP-Tree



图 4-2 LSP-Tree 中叶子节点 (Leaf-Node) 结构示意图

Fig. 4-2 the structure of Leaf-Node in LSP-Tree

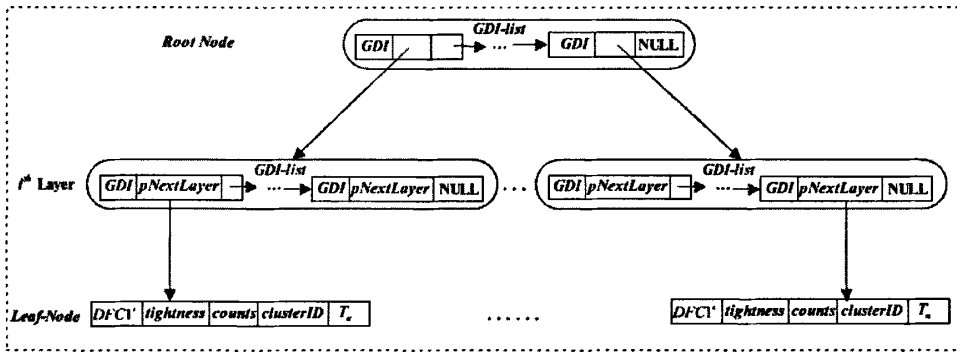


图 4-3 LSP-Tree 结构 (二维空间为例) 示意

Fig. 4-3 the structure of LSP-Tree (take 2-dimensional space for example)

当有新的数据流对象到达时, GC-Stream 算法将维护 LSP-Tree 结构, 即将新的数据流元素对应的网格单元或数据流对象插入到 LSP-Tree 中。

Maintain_LSP_Tree 过程的实现伪代码如 Algorithm 4-2 所示。

Update_LSP_Tree 过程将对数据流中到达的每个新的数据流对象 x_{T_i} , 更新其所对应的网格单元 (假定为 G_j) 的特征向量信息, 假定网格单元 G_j 对应的特征向量信息为 $GF_j = \langle DFCI_{pre}^x, tightness_{pre}, counts_{pre}, clusterID, (T_u)_{pre} \rangle$ 。在 GC-Stream 算法中, 将采用衰减窗口模型中的指数衰减策略来更新网格单元的特征向量信息, 具体的更新策略如式 4-13 所示, $GF_j = \langle DFCI^x, tightness, counts, clusterID, T_u \rangle$ 表示网格单元 G_j 对数据流对象 x_{T_i} 作更新之后的特征向量信息, $\alpha (\alpha > 0)$ 称为衰减因子, $k (k = 1, 2, \dots, d)$ 表示数据空间的维。

Algorithm 4-2: Maintain_LSP_Tree (Root, GDIS, x_T)

Input: the pointer *Root* pointed to root node, the set *GDIS* of grid index corresponding with data stream object x_T .

Output: LSP-Tree.

Procedure:

```

(1) PCurrent-NLN= Root; //points to non-leaf or leaf node
(2) PpreGDIN=NULL; // points to GDI-Node of prepare layer
(3) Flag=false;
(4) FOR each GDIS[i] in GDIS DO BEGIN
(5) IF PCurrent-NLN=NULL THEN
(6) PCurrent-NLN=Create_NLN (GDIS[i]);
(7) IF (Root=NULL) THEN
(8) Root= PCurrent-NLN;
(9) ELSE
(10) PpreGDIN→pNextLayer = PCurrent-NLN;
(11) END
(12) PpreGDIN= PCurrent-NLN;
(13) ELSE
(14) Flag=Find_exists_GDIL(PCurrent-NLN, GDIS[i], PpreGDIN);
(15) IF(Flag=False) THEN
(16) PpreGDIN=Insert_GDI_GDIL(PCurrent-NLN, GDIS[i]);
(17) END
(18) END
(19) PCurrent-NLN= PpreGDIN→pNextLayer ;
(20) Flag=false;
(21) END // FOR END
(22) IF (PCurrent-NLN=NULL) THEN
(23) PpreGDIN→pNextLayer=Create_Leaf_Node( $x_T$ );
(24) ELSE
(25) Update_Leaf_Node( $x_T$ );
(26) END

```

$$GF_j = \langle DFC1^x, tightness, counts, clusterID, T_u \rangle$$

$$GF_j.counts = counts_{pre} \times 2^{-\alpha(T_i - (T_u)_{pre})} + 1 (\alpha > 0)$$

$$GF_j.T_u = T_i \quad (4-13)$$

$$(GF_j.DFC1^x)^k = ((DFC1^x_{pre})^k + (x_{T_i})_k) / 2 (k=1, 2, \dots, d)$$

$$GF_j.tightness = (tightness_{pre} \times 2^{-\alpha(T_i - (T_u)_{pre})} + \|GF_j.DFC1^x - x_{T_i}\|) / 2 (\alpha > 0)$$

4.3.3 LSP-Tree 剪枝过程

由于数据流通常是潜在无限的，随着流数据集的增大，维护数据流对象的概要信息的聚类模型规模将持续增长。因此，数据流聚类分析算法必须采取剪枝操作来动态适应内存大小的限制。在 GC-Stream 算法中，剪枝操作主要实现对两种类型的网格单元的剪枝，一类是对“过时的”网格单元的剪枝，即相对于当前时

间而言，它们长时间没有得到更新，也即它们的持续时间 $T > \tau$ ，其中 τ 是用户设定的时间阈值；另一类是对“噪声”网络的剪枝，即它们包含的数据点的数目 $counts < minPts$ ， $minPts$ 是网格密度阈值参数。对这两种类型的网格单元的剔除，都要涉及到 LSP-Tree 的删除操作。*Pruning_LSP_Tree* 的具体实现过程如算法 4-3 所示。

Algorithm 4-3: Pruning_LSP_Tree (Root, Layer, PPre_GDI_Node) .
Input: Root points to current Non_Leaf Node,
 Layer is the layer of Non-Leaf Node pointed by Root.
 PPre_GDI_Node points to GDI_Node whose pNextLayer is equal to Root.
Output: LSP-Tree.
Procedure:
 (1) Size= the number of GDI_Node in GDI_list pointed by Root.
 (2) FOR ($i=1$; $i \leq size$; $i++$) DO BEGIN
 (3) IF ($Layer < d+1$) THEN
 (4) Pruning_LSP_Tree (Root.GDI_list[i] \rightarrow pNextLayer, Layer+1, Root.GDI_list[i]);
 (5) ELSE
 (6) IF (Is_Satisfy_Pruning_Condition (τ , minPts)) THEN
 (7) FREE (Root);
 (8) PPre_GDI_Node \rightarrow pNextLayer=NULL;
 (9) END
 (10) return;
 (11) END
 (12) END

Pruning_LSP_Tree 过程是一个递归的过程，它递归地遍历 LSP-Tree 中的每个叶子节点，对网格单元的特征向量中的特征值进行计算，如果 $T_c - T_u > \tau$ 或 $counts < minPts$ ，则从 LSP-Tree 中删除这些网格单元所对应的叶子节点。

4.3.4 LSP-Tree 聚类过程

在 GC-Stream 算法中，当需要输出聚类结果时，将对 LSP-Tree 进行遍历，基于网格密度可达的概念（见 3.2 节定义 3.8）寻找最大的网格密度连通（见 3.2 节定义 3.9）的核心网格单元集合，并将它们识别为簇，直到所有的核心网格单元都被处理。

4.4 GC-Stream 算法性能分析

在 d 维数据空间中，假定数据流 D_{T_c} 中共包含 N 个数据流对象，并且数据空间中每个维被划分为 m 个单元。对于 GC-Stream 算法而言，LSP-Tree 的维护和

更新过程，主要是将数据流对象对应的网格单元插入到 LSP-Tree 中，由于每一维的比较次数是 $\log_2 m$ ，则总共的比较次数是 $d \log_2 m$ ，那么 N 个数据流对象总共的比较次数是 $Nd \log_2 m$ ，因此，维护和更新过程的时间复杂度是 $o(Nd \log_2 m)$ 。由于 d 和 m 都是常数，因此，其可被近似为 $o(N)$ ，是线性的时间复杂度。剪枝阶段的时间复杂度是 $o(m^d)$ 。对于聚类输出阶段，在最坏情况下（即所有网格单元都是核心网格），邻居网格单元的数目是 $2d \times m^d$ ，而每个邻居网格单元的查找时间是 $d \log_2 m$ ，因此，聚类输出阶段的时间复杂度是 $o(2d^2 m^d \log_2 m)$ 。由于 $\lim_{m \rightarrow \sqrt[N]{N}} m^d = N$ ，因此，其可近似为 $o(N)$ 。总体上来说，GC-Stream 算法的时间复杂度是 $o(N)$ ，是线性的。

GC-Stream 算法利用 LSP-Tree 结构来组织非空网格单元集合，空间复杂度是 $o(m^d)$ ，由于 $\lim_{m \rightarrow \sqrt[N]{N}} m^d = N$ ，因此，在最坏情况下，空间复杂度可近似为 $o(N)$ 。

而实际情况下，往往是 $m^d \ll N$ 。

从上面的分析可知，无论是从时间复杂度还是从空间复杂度来说，GC-Stream 算法都能较好地满足数据流聚类分析算法的实时性要求和受内存大小的限制要求。

4.5 GC-Stream 算法实验结果与性能评价

GC-Stream 算法的实验运行环境如下：Microsoft Windows XP Professional 2002 操作系统，Intel (R) Core (TM) 2 Duo CPU 2.66GHz，2GB 内存。编程环境是 Microsoft Visual C++ 6.0，利用 C++ 语言实现。

数据集采用 Pelleg 和 Moore 人工合成的 Gaussian 数据集[49]，该数据集共包含 2000 个数据点，11 个簇。图 4-4 给出了 Gaussian 数据集的原始分布情况。图 4-5 给出了 GC-Stream 算法在 Gaussian 数据集上的聚类分析结果。从二者的对比可以看出，GC-Stream 算法的聚类结果具有较好的准确性。

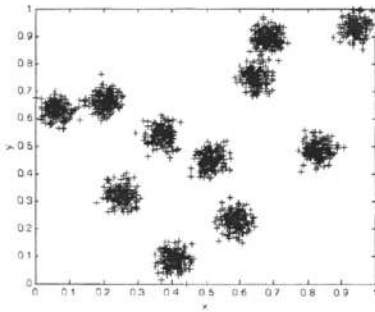


图 4-4 Gaussian 数据集初始分布情况
Fig. 4-4 The initial distribution of gaussian dataset

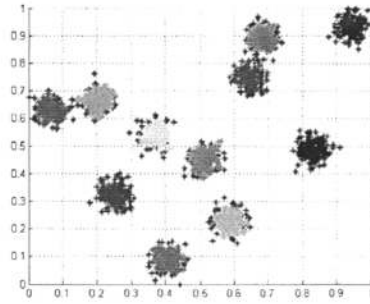


图 4-5 GC-Stream 算法的聚类结果
Fig. 4-5 The clustering results of GC-Stream algorithm in Gaussian dataset

图 4-6 给出了 GC-Stream 算法和 CluStream 算法的运行时间的对比结果。从图中可以看出，随着数据流的持续到达，GC-Stream 算法的运行时间少于 CluStream 算法，这说明 GC-Stream 算法比 CluStream 算法具有高的执行效率，这是因为在 GC-Stream 算法中使用了效率较高的 LSP-Tree 结构来维护数据流对象的摘要统计信息。此外，GC-Stream 算法的运行时间随着数据流的增长线性增长的速度变慢，这是由于随着 LSP-Tree 结构中保存的网格单元的数目增多，当插入新的数据流对象时，减少了往 LSP-Tree 中插入 GDI-Node 的次数，因此，总的运行时间线性增长的速度减缓。

图 4-7 给出了在不同的 m 取值情况下，GC-Stream 算法的空间大小随数据流大小的变化情况。 m 表示数据空间中每一维的网格单元的数目，对于 d 维数据空间，总的网格单元的数目是 m^d 。从图中可以看出，当 m 的值比较大时，GC-Stream 算法运行时占用的空间较大，这是因为 m 比较大时，网格的划分粒度比较细，网格数目规模大，这样就导致 LSP-Tree 的频繁维护，不断地插入 GDI-Node，占用大量的内存空间。此外，从图中还可以看出，GC-Stream 算法的空间大小随着数据流的增长首先呈现快速增长的趋势，接着在下降后呈现平稳的趋势。这是由于初始时，LSP-Tree 结构规模较小，保存的网格单元数目较少，因此，需要频繁地维护 LSP-Tree，以便将新的数据流对象对应的网格单元插入到 LSP-Tree 中；当数据流大小增加到 60K 时，算法的空间大小迅速下降，这是由于对 LSP-Tree 进行了剪枝操作，下降趋势的程度受到噪声网格和“过时”网格的数目的影响；随着数据流的继续增长，GC-Stream 算法的空间大小趋于平稳，这时 LSP-Tree 结构规模稳定，维护操作减少，而只进行更新操作。

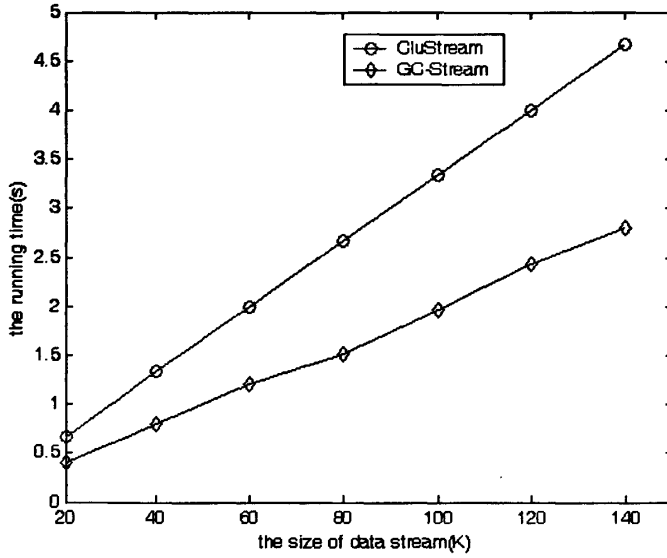


图 4-6 GC-Stream 算法和 CluStream 算法的时间对比结果

Fig. 4-6 The comparison of running time for GC-Stream and CluStream algorithm

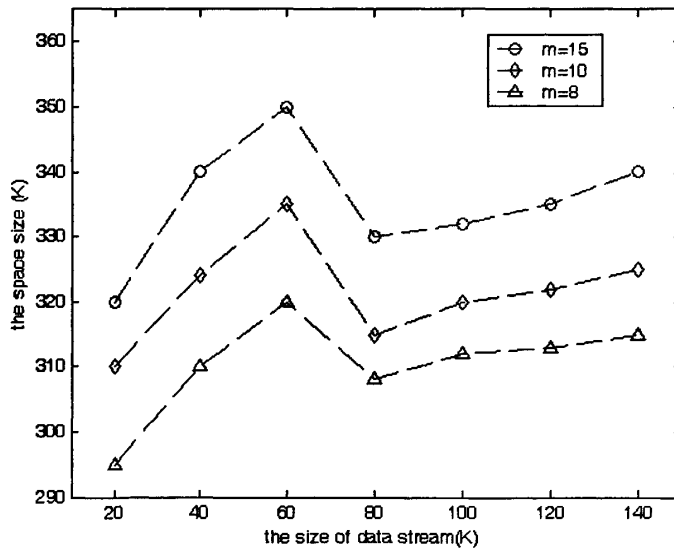


图 4-7 在不同的 m 值下，GC-Stream 算法的空间大小随数据流大小的变化

Fig. 4-7 The variation of space size with the size of data stream in different m

从上面的分析可以看出，GC-Stream 算法利用 LSP-Tree 结构来组织保存数据流对象的摘要统计信息的网格单元，实现了较好的聚类准确性，并很好地满足了数据流聚类分析算法的实时性要求和动态适应内存大小的特性。

4.6 本章小结

本章主要分析和讨论了基于网格的数据流聚类分析算法 GC-Stream。首先,介绍了数据流的概念,分析了数据流聚类分析算法的特性及窗口模型,重点分析了典型的数据流聚类分析算法 CluStream、AIN-Stream 及 GDDS 的聚类模型、窗口模型以及聚类方法。其次,描述了 GC-Stream 算法的总体框架,主要包括聚类模型的维护和更新阶段、聚类模型的剪枝阶段以及聚类结果输出阶段。接下来,对 GC-Stream 算法的关键实现过程和技术进行了描述,包括网格单元的特征向量结构, LSP-Tree 的叶子节点和非叶子节点结构, LSP-Tree 的维护和更新策略, LSP-Tree 的剪枝策略,以及聚类结果的输出。最后,从时间复杂度和空间复杂度两个方面分析了 GC-Stream 算法的性能,并给出了算法的实验结果和性能评价结果。

第五章 总结与展望

不同的应用对聚类分析算法提出了特定的要求,本文介绍了聚类分析算法的一般要求,并在分析基于网格的聚类分析算法的优势和典型的基于网格的聚类分析算法的基本原理和特性的基础上,提出了基于网格的并行的聚类分析算法 PGMCLU,该算法侧重于解决聚类分析算法面临的数据海量性、高维性,及对多密度数据集的聚类能力的困难。理论分析和实验验证都表明 PGMCLU 算法具有较好的聚类能力,能够处理海量的、高维的、带噪声的、包含任意形状簇的、多密度的数据集,并且在加速比和效率方面表现较好。

数据流聚类分析是数据流挖掘中的一个重要研究方向,并具有广泛的应用领域。本文介绍了数据流的概念、数据流聚类分析算法的特性及三种典型的窗口模型,并重点分析了典型的数据流聚类分析算法的聚类模型、衰减策略以及聚类方法,提出了基于网格的数据流聚类分析算法 GC-Stream,该算法关注数据流的动态变化,侧重于分析数据流中蕴含的演化趋势。理论分析和实验结果都表明 GC-Stream 算法能够很好地满足数据流聚类分析算法的实时性要求以及内存空间的限制性。

本章将对论文的主要研究工作进行总结,并对未来可行的进一步的研究工作进行展望。

5.1 论文总结

本文主要的研究工作包括以下几个方面:

(1) 提出了网格紧凑度、网格密度直接可达、网格密度可达及网格密度连通等概念。网格紧凑度度量了网格单元中数据点之间的紧密程度,为较好地识别不同密度的簇奠定了基础。

(2) 针对集群系统和基于网格的空间划分的特性,提出了基于参考维的数据分区方法。该方法基于“分而治之”的思想,通过确定每个节点处理的网格单元集合在参考维的索引范围,可以实现将数据分配到不同的节点,各个节点独立地对不同的数据分区中的数据点进行处理,提高了 PGMCLU 算法的时间效率,

以及对海量数据集的处理能力。

(3) 提出利用 SP-Tree 结构来组织非空网格单元集合, 以及利用网格特征值来描述网格单元的摘要统计信息, 并具体描述了网格单元的特征值结构。聚类分析算法通过扫描 SP-Tree 结构, 并分析网格单元的特征值信息便可以获得聚类所需要的信息, 提高了聚类算法的时间效率。

(4) 实现了网格密度阈值 $minPts$ 的自动设置。 $minPts$ 参数是根据数据的分布特征自动确定的, 提高了识别核心网格及稀疏网格的能力和质量。

(5) 提出了对边界网格的处理策略。通过计算边界网格与邻居核心网格的距离, 确定边界网格应该归属于哪个簇, 提高了聚类结果的精度。

(6) 提出了基于网格密度连通的局部聚类方法。在 PGMCLU 算法中, 簇定义为最大的网格密度连通的网格单元集合。通过遍历 SP-Tree 空间索引结构, 并根据网格单元直接密度可达的定义在邻居网格中迭代地寻找直接密度可达的网格单元来识别簇, 这样可以很好地识别任意形状的簇, 以及具有不同密度的簇。

(7) 提出了基于簇密度和簇相似性的局部聚类的合并策略。簇密度较好地度量了簇中包含的网格单元的平均密度, 簇相似性度量的提出扩展了度量簇之间的相似度的方法。通过在节点之间交换局部聚类结果的簇信息, 并根据簇相似度量来合并簇和更新簇的组编号, 可以有效地实现将局部聚类结果合并为全局的聚类结果。

(8) 提出了将数据空间划分为网格, 并用网格单元的特征向量来描述数据流对象统计信息的方法。网格单元的特征向量很好地反映了数据流对象的特征, 并较好地支持了数据流对象的增量更新和衰减。

(9) 提出了基于 List 的改进的 SP-Tree 结构 LSP-Tree, 以及基于该结构的维护和更新策略。LSP-Tree 形成了 GC-Stream 算法的聚类模型。LSP-Tree 的叶子节点保存了网格单元的特征向量, 而非叶子节点是一个 List 结构, List 结构中的元素按照升序排列, 这种结构较好地保证了 LSP-Tree 的维护和更新过程的执行效率。

(10) 提出了网格单元信息衰减方法和剪枝方法。GC-Stream 算法采用了基于相对时间的指数衰减方法, 可以对网格单元信息进行衰减, 以便分析数据流的演化趋势。剪枝过程通过对噪声网格和“过时”网格的处理较好地抑制了 LSP-Tree 的增长规模, 并提高了 LSP-Tree 的维护效率。

5.2 工作展望

本文在分析典型聚类算法研究现状及基本原理的基础上,提出了基于网格的并行的聚类分析算法 PGMCLU,以及基于网格的数据流聚类分析算法 GC-Stream。PGMCLU 算法重点研究了数据分区、构建 SP-Tree 空间索引结构、局部聚类及聚类合并方法。GC-Stream 算法则基于衰减窗口模型,重点研究了新的聚类模型 LSP-Tree,以及模型维护、更新和剪枝策略。在上述算法的研究过程中,取得了许多研究成果。在这些成果的基础上,下一步的研究工作可以从以下几个方面展开:

(1) SP-Tree 空间索引结构依据网格单元的空间位置关系,实现了对基于网格的聚类分析算法中网格单元集合的高效组织,极大地提高了邻居网格的查找效率。但是,数据空间中网格单元的数目随维数的增长呈指数增长趋势,这样就会增加 SP-Tree 的规模大小。因此,可以研究利用属性子集选择方法和维度规约方法(如小波变换和主成分分析)来降低数据集的维度,从而减少网格划分过程中产生的网格单元的数目。

(2) GC-Stream 算法采用了衰减窗口模型来适应数据流的动态变化,以便分析数据流的演化趋势。但是基于衰减窗口模型的数据流聚类分析算法通常只能反映当前时间戳时数据流的演化(或进化)规律,不能正确地反映数据流在任意时间段内的演化规律。因此,可以采用滑动窗口模型和快照技术来改进算法以分析数据流在任意时间段内的演化趋势。

(3) LSP-Tree 空间索引结构支持对多个 LSP-Tree 的合并操作,因此,对于高速的批量到达的数据流对象可以考虑研究并行的数据流聚类算法,各个节点独立地建立自己的 LSP-Tree 结构,当用户需要输出某一时间戳时的聚类结果时,由离线过程完成多个 LSP-Tree 的合并操作并输出聚类分析结果。

(4) 针对特定的数据流(如气象数据流等),对 GC-Stream 算法进行进一步的完善,使其可以实时、精确、高效地对数据流进行分析,提供有价值的聚类结果供用户参考。

参考文献

- [1] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281-297.
- [2] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. John Wiley and Sons, New York, November 1990.
- [3] Ng R T, Han J. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994, pp. 144-155.
- [4] Zhang T, Ramakrishnan R, Livny M. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 1996, pp.103-114.
- [5] S. Guha, R. Rastogi, K. Shim. ROCK: A robust clustering algorithm for categorical attributes [J]. *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, 512-521.
- [6] S. Guha, R. Rastogi, K. Shim. CURE: an efficient clustering algorithm for large database[J]. *Information Systems*. 2001, 26(1): 35-58.
- [7] G. Karypis, E. H. Han, V. Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling [J]. *IEEE COMPUTER*, August 1999, 32(8): 68-75.
- [8] Ester M, Kriegel H-P, Sander J, et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, August 1996. pp. 226-231.
- [9] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering Points to Identify the Clustering Structure. In *Proc. of the ACM SIGMOD'99 International Conference on Management of Data*, Philadelphia, PA, 1999, pp. 49-60.
- [10] A. Hinneburg and D.A. Keim, An Efficient Approach to Clustering in Multimedia Databases with Noise. In *Proc. of the 4th International Conference on Knowledge Discovery and Data Mining*, New York, 1998, pp. 58-65.
- [11] Agrawal R, Gehrke J, Gunopulos D, et al. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, 1998, pp. 94-105.
- [12] Wang W, Yang J, Muntz R. STING: A Statistical Information Grid Approach to Spatial Data Mining. In *Proceedings of the 23rd VLDB Conference*, Athens, Greece, 1997, pp. 186-195.
- [13] Sheikholeslami G, Chatterjee S, Zhang A. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In *Proceedings of the 24th VLDB Conference*, New York, USA, 1998, pp. 428-439.
- [14] Ji Zhang, Wynee Hsu, Mong Li Lee. Clustering in dynamic spatial databases. *Journal of intelligent information systems*, 2005, 24(1), 5-27.
- [15] Aggarwal C C, Wolf J L, Yu P S, et al. Fast Algorithms for Projected Clustering. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, 1999, pp. 61-72.
- [16] T. Kohonen. Self-organized formation of topologically correct feature maps [J]. *Biological Cybernetics*, 1982, 43(1):59-69.
- [17] D. Fisher. Improving inference through conceptual clustering. In *Proc Int. Conf on AAAI*, Seattle, 1987, 461-465.
- [18] Ozge Uncu, William A. Gruver, Dilip B. Kotak. GRIDBSCAN: GRId Density-Based Spatial Clustering of Applications with Noise. In *IEEE International Conference on Systems, Man and Cybernetics*, Taipei, Taiwan, 2006, pp. 2976-2981.
- [19] Chen Xiaoyun, Min Yufang, Zhao Yan, Wang Ping. GMDBSCAN: Multi-Density DBSCAN Cluster Based on Grid. In *IEEE International Conference on e-Business Engineering*, Xi'an, China, 2008, pp. 780-783.
- [20] César S. de Oliveira, Paulo Igor Godinho, Aruanda S. G. Meiguins. EDACluster: An Evolutionary Density and Grid-Based Clustering Algorithm. In *Seventh International Conference on Intelligent Systems Design and Applications*, 2007, pp. 143-148.

- [21] Shaaban Mahran, Khaled Mahar. Using grid for accelerating density-based clustering. In *8th IEEE International Conference on Computer and Information Technology*, July 2008, pp.35 – 40.
- [22] 刘敏娟, 柴玉梅. 基于网格的共享近邻聚类算法 [J]. 计算机应用, 2006, 26 (7): 1673-1675.
- [23] Chih-Ming Hsu, Ming-Syan Chen. Sunspace clustering of high dimensional spatial data with noises. PAKDD, 2004, pp. 31-40.
- [24] John T. Rickard, Ronald R. Yager, Wendy Miller. Mountain clustering on non-uniform grids using P-tree [J]. Fuzzy optimization and decision making, 2005 (4): 87-102.
- [25] ZHAO Yanchang, SONG Junde. GDILC: A Grid-based Density-Isoline Clustering Algorithm. IEEE 2001, pp. 140-145.
- [26] Chen ning, Chen an, Zhou longxing. An incremental grid density-based clustering algorithm [J]. Journal of software, 2002, 13(1), 1-7.
- [27] ZHOU Bing, SHEN Junyi, PENG Qinke. Parallel Clustering Algorithm for PCs Cluster. Computer Engineering, Vol.30, NO.4, February 2004, pp. 4-6.
- [28] 李肯立, 杨进, 彭成斌, 秦云川. 基于 MPI+OpenMP 混合模型的并行地震数据处理支持库的研究 [J]. 计算机工程与科学, 2007, 29 (12): 136-139.
- [29] 周兵, 冯中慧, 王和兴. 集群环境下的并行聚类算法之研究 [J]. 计算机科学, 2007, 34 (10): 195-199.
- [30] L. Ertöz, M. Steinbach, V. Kumar. A New Shared Nearest Neighbor Clustering Algorithm and its Applications. In *Workshop on Clustering High Dimensional Data and its Applications, Proc. of Text Mine '01, First SIAM Intl. Conf. on Data Mining*, Chicago, IL, USA, 2001.
- [31] L. Ertöz, M. Steinbach, V. Kumar. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. In *Proc. of the 2003 SIAM Intl. Conf. on Data Mining*, San Francisco, May 2003.
- [32] Panagiotis Antonellis, Christos Makris, Nikos Tsiarakis. Algorithms for clustering clickstream data. Information Processing Letters, 109 (2009), pp. 381 – 385.
- [33] Shi Zhong. Efficient streaming text clustering. Neural Networks, 18 (2005), pp. 790–798.
- [34] Jae Woo Lee, Nam Hun Park, Won Suk Lee. Efficiently tracing clusters over high-dimensional on-line data streams. Data & Knowledge Engineering, 68 (2009), pp. 362 – 379.
- [35] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta. Clustering distributed data streams in peer-to-peer environments. Information Sciences, 2006, 176(14), pp. 1952–1985,
- [36] Guha S, Meyerson A, Misher A, et al. Clustering data streams: theory and practice [J]. IEEE Transactions on knowledge and engineering, January 2003, 3(15), 515-547.
- [37] C. C. Aggarwal et al. Fast Algorithms for Projected Clustering. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. 1999, pp.61-72.
- [38] C. C. Aggarwal, J. W. Han, J. Y. Wang and P. S. Yu. A Framework for Projected Clustering of High Dimensional Data Streams. In *Proceedings of the 30th VLDB Conference*. Toronto, Canada, 2004, pp.852-863.
- [39] 王述云. 数据流频繁项挖掘与聚类分析的研究 [D]. 上海: 复旦大学, 2008.
- [40] Nam Hun Park, Won Suk Lee. Cell trees: An Adaptive Synopsis Structure for Clustering Multi-dimensional On-line Data Streams. Data & Knowledge Engineering. 2007, 63(2): 528-549.
- [41] 曾东海. 基于网格密度和空间划分树的聚类算法研究 [D]. 广州: 厦门大学, 2006.
- [42] G. Milligan. An algorithm for creating artificial test clusters. Psychometrika, 50(1), 1985, pp.123–127.
- [43] M. Garofalakis, J. Gehrke, R. Rastogi. Querying and mining data streams: you only get one look. In *The Tutorial Notes of the 28th International Conference on Very Large Databases*, Hong Kong, China, August 2002.
- [44] Joong Hyuk Chang, Won Suk Lee. Finding frequent itemsets over online data streams. Information & Software Technology, 2006, 48 (7), pp. 606 – 618.
- [45] M. Datar, A. Gionis, P. Indyk, R. Motawi. Maintaining stream statistics over sliding window, In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 2002.
- [46] Mohamed Medhat Gaber, Arkady B. Zaslavsky, Shonali Krishnaswamy. Mining data streams: a review, *SIGMOD*, 2005 , 34 (2), pp.18–26.
- [47] 杨霞玲. 多维数据流聚类算法的分析与实现 [D]. 北京: 北京工业大学, 2009.

- [48] 高永梅, 黄亚楼. 一种基于网格和密度的数据流聚类算法[J]. 计算机科学, 2008, 35(2), 134-137.
- [49] Dan Pelleg, Andrew Moore. X-means: Extending K-means with Efficient Estimation of Number of Clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

在学期间的研究成果

参与的科研项目：

1. 课题名称：空间数据挖掘的高性能计算研究。课题来源：甘肃省计算中心科研项目。项目编号：2007JS0001。
2. 项目名称：基于 RFID 技术的嵌入式数据库系统及相关产品的开发。项目计划类别：广东省教育部产学研结合项目。项目编号：2007A090302117。

已发表的学术论文：

- [1] CHEN Xiaoyun, CHEN Yi et al. PGMCLU: A Novel Parallel Grid-based Clustering Algorithm for Multi-density Datasets. *The 1st IEEE Symposium on Web Society (SWS 2009)*, August 23-24, 2009 Lanzhou, China. (20094812511845)
- [2] Kaiyin Huang, Pengfei Chen, Yi Chen et al. The Research for Embedded Active Database Based on ECA Rule and Implementation in SQLite Database. *2009 International IEEE Workshop on Database Technology and Applications (DBTA2009)*, April 25-26, 2009 Wuhan, China. (20094712477190)
- [3] Xiaoyun Chen, Youli Su, Yi Chen et al. GK-means: An Efficient K-means Clustering Algorithm Based On Grid. *2009 International Symposium on Computer Network and Multimedia Technology (CNMT 2009)*, December 18-20, 2009 Wuhan, China. (20101212778879)
- [4] 黄楷胤, 陈毅等. 基于 Web Service 的 RFID 系统集成应用[J]. 微计算机信息, 2009, 25(35):15-17.

致谢

时光荏苒，三年的研究生阶段的学习和生活已经进入尾声。三年的时光，无数美丽的身影在温暖和感动着我！三年的时光，奋斗与喜悦永驻，迷惘与释然交织，温暖与幸福辉映！三年的时光，镌刻了每一个美妙的时刻，凝聚了每一次蜕变的阵痛，记录了每一次轻盈的跳跃！三年的时光，灵魂在次次荡涤中，散发着高贵的芬芳气息！

真诚地感谢我的导师陈晓云教授，陈老师您辛苦了！三年的时光中，在陈老师的贴心的关怀和悉心的指导下，我在点滴进步中走向成长，完成每一次成功的跳跃！陈老师一丝不苟、勇于创新的科研精神，深深地影响着我的科研工作，鼓励我走近科研、了解科研、相识科研，品味科研的苦于涩，体验科研的欣与喜！陈老师为人师表的高尚人格，乐观的人生态度，锲而不舍的钻研精神，以“润物细无声”的伟大滋润着我的心，伴我度过坎坷，走向成功！

感谢实验室同学陈鹏飞，苗圣法，李龙杰，岳敏，何艳珊，张鑫，刘国华，苏有丽，姚毓凯，霍萌萌！他们在科研上给予了我积极地支持，在生活上给予温暖的关怀！

感谢我的舍友！感谢他们对我的关怀，为我提供了一个温馨的宿舍，让我在这里度过温暖和快乐的三年时光！

感谢我的父母、妹妹和亲人！感谢伟大的父母二十多年来默默无闻，一如既往的支持！感谢可爱的妹妹给这个家带来的欢乐与笑声，感谢她对哥哥力所能及的支持！感谢每一个可亲可爱的亲人对我的鼓励和支持！

感谢支持和影响我的每一个人，没有他们（她们）影响我，也就没有我影响世界！真诚地祝福他们（她们）在未来的日子里平安幸福！“路漫漫其修远兮，吾将上下而求索”，我将用我不懈的执着与奋斗，在生命的空地上，绘制精彩的图案，奏响生命的华章！

2010年4月