

硕士学位论文

基于树结构子空间划分的网格聚类算法研究

Research on Grid-based Clustering Based on Tree
Structure Subspace Partitioning

专业学位类别

电子信息

专业领域

计算机技术

作者姓名

樊超

指导教师

冯启龙 教授

2023 年 5 月

中图分类号 TP301

学校代码 10533

UDC 004

学位类别 专业学位

硕士学位论文

基于树结构子空间划分的网格聚类算法研究

Research on Grid-based Clustering Based on Tree Structure Subspace Partitioning

作者姓名

樊超

专业学位类别

电子信息

专业领域

计算机技术

研究方向

数据挖掘和大数据处理

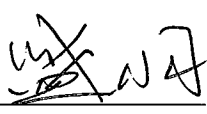
二级培养单位

计算机学院

指导教师

冯启龙 教授

论文答辩日期 2023.5.19

答辩委员会主席 

中 南 大 学

2023 年 5 月

学位论文原创性声明

本人郑重声明，所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了论文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得中南大学或其他教育机构的学位或证书而使用过的材料。与我共同工作的同志对本研究所作的贡献均已在论文中作了明确的说明。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

学位论文作者签名：樊超 日期：2023 年 5 月 19 日

学位论文版权使用授权书

本学位论文作者和指导教师完全了解中南大学有关保留、使用学位论文的规定：即学校有权保留并向国家有关部门或机构送交学位论文的复印件和电子版，允许本学位论文被查阅和借阅。本人授权中南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和公开传播，可以采用复印、缩印或其它手段保存和汇编学位论文。本人同意按《中国优秀博硕士学位论文全文数据库出版章程》规定享受相关权益。本人保证：毕业后以学位论文内容发表的论文作者单位注明中南大学；学位论文电子文档的内容和纸质学位论文的内容相一致。

延缓公开论文延缓到期后适用本授权书，涉密论文在解密后适用本授权书。

本学位论文属于：(请在以下相应方框内打“√”)

☒ 公开

☐ 延缓公开，延缓期限（__ 年 __ 月 __ 日至 __ 年 __ 月 __ 日）

学位论文作者签名：樊超

指导教师签名：冯志超

日期：2023 年 5 月 19 日

日期：2023 年 5 月 19 日

(填写阿拉伯数字)

基于树结构子空间划分的网格聚类算法研究

摘要：聚类分析是一种无监督学习方法，被广泛地应用于数据挖掘和机器学习等领域。密度聚类是聚类分析领域的热点研究方向，其独特优势在于能够识别任意形状类并有效地处理噪声数据。然而，由于密度聚类算法的时间复杂度普遍较高，限制了其在大数据分析中的应用场景。网格聚类作为密度聚类的一种重要的衍生方法，通过对数据空间进行网格划分后对网格聚类，能够有效地提高聚类效率。但是，网格聚类会带来聚类质量的下降，且其聚类效率通常会随着维度的增加而急剧下降，这使得网格聚类算法难以高效地处理高维度大数据集。因此，提高网格聚类质量和其在高维大数据集下的运行效率，是密度聚类研究领域的一个重要课题。

(1) 针对网格聚类质量下降的问题，本文提出了一种新的网格聚类模型。该模型对网格划分、网格密度计算、网格距离度量和网格聚类规则等核心因素进行了重新定义和设计，从而显著提高了网格聚类的质量。该模型重新定义了网格密度，考虑了相邻网格之间的相互影响，并设计了新的网格密度计算方法，使得网格密度的度量更加合理。其次，该模型重新设计了网格距离，通过在网格内保留点的摘要信息，能够计算不同网格内点的距离来替代原有的网格距离，从而使得网格聚类结果更加准确。在网格模型的设计基础上，本文还提出了一种适用于低维大数据的聚类算法 GMCLU。该算法在低维空间上能够达到 $O(n)$ 的时间复杂度，为网格聚类在处理大规模低维数据时提供了有效的解决方案。

(2) 针对网格聚类在处理高维度大数据时所面临的效率问题，本文提出了一种基于树结构的子空间划分以及子聚类高效合并的聚类框架，并依据该框架设计了适用于高维大数据聚类分析的 GTCLU 算法。GTCLU 算法的核心理念在于采用分治策略降低聚类网格的规模，从而提升算法的运行效率。通过构建一棵树形结构，该算法将网格空间划分为多个较小的子空间，并将这些子空间存放在叶子节点中。GTCLU 针对叶子节点中的子空间采用高效的网格聚类算法进行快速聚类，然后在非叶子节点自底向上地快速合并各个子聚类结果，最终在根节点获得完整的聚类。本文在实验中验证了 GTCLU 算法在提升

网格聚类运行效率方面的有效性,表明了其适用于高维大数据集的聚类分析。

图 15 幅, 表 6 个, 参考文献 95 篇

关键词: 密度聚类; 网格聚类; 子空间划分; 大数据

分类号: TP301

Research on Grid-based Clustering Based on Tree Structure Subspace Partitioning

Abstract: Density-based clustering is a fundamental research area in clustering analysis, possessing unique benefits such as identifying arbitrarily shaped clusters and effectively managing noisy data. However, the high time complexity inherent to density-based clustering algorithms restricts their application in big data analysis. Grid-based clustering, a significant extension of density-based clustering, aims to enhance clustering efficiency by dividing the data space into grids and clustering them. Nonetheless, grid-based clustering often reduces clustering quality, and its efficiency tends to decline markedly with increasing dimensions, making it challenging to efficiently process high-dimensional large datasets. Consequently, improving the clustering quality and the high-dimensional clustering efficiency of grid-based models presents a crucial challenge in density-based clustering research.

(1) To address the issue of declining clustering quality, this thesis proposes a new grid-based clustering model. By redesigning essential components such as space division, grid density calculation, grid distance measurement, and grid clustering rules, this model substantially enhances the quality of grid clustering. The model reconsiders grid density by accounting for the mutual influence between neighboring grids and devises a new grid density calculation method, yielding a more logical grid density measurement. Furthermore, the model restructures grid distance by preserving summary information of points within the grid, facilitating the computation of distances between points in distinct grids as an alternative to the original grid distance, thereby increasing the accuracy of grid clustering results. Based on the proposed model, this thesis also presents the GMCLU algorithm, suitable for low-dimensional large data, which achieves a time complexity of $O(n)$ in low-dimensional spaces, providing an effective solution for grid clustering when processing large-scale low-dimensional data.

(2) To address the efficiency challenges associated with grid-based clustering in high-dimensional large data, this thesis proposes a clustering framework based on a tree structure for subspace partitioning and efficient merging of subclusters. Additionally, this study develops the GTCLU algorithm base on the proposed framework to cluster high-dimensional datasets. The central principle of the GTCLU algorithm is to employ a divide-and-conquer strategy to reduce the scale of clustering grids, thereby improving the algorithm's performance efficiency. By constructing a tree structure, the algorithm partitions the grid space into smaller subspaces and stores these subspaces in leaf nodes. GTCLU utilizes an efficient grid clustering algorithm for rapid clustering the grids within the leaf nodes' subspaces and subsequently merges the subclusters from bottom to top in non-leaf nodes, ultimately achieving a comprehensive clustering result at the root node. Through systematic experiments, this thesis demonstrates the effectiveness of the GTCLU algorithm in enhancing the performance efficiency of grid-based clustering.

Keywords: Density-based Clustering; Grid-based Clustering; Subspace Partitioning; Big Data

Classification: TP301

目 录

第 1 章 绪论.....	1
1.1 研究背景及意义.....	1
1.2 国内外研究现状.....	2
1.3 本文研究内容.....	4
1.4 论文组织结构.....	6
第 2 章 相关理论基础.....	7
2.1 聚类分析概述.....	7
2.1.1 划分聚类.....	7
2.1.2 层次聚类.....	8
2.1.3 密度聚类.....	9
2.2 密度聚类原理.....	9
2.2.1 DBSCAN 算法	10
2.2.2 OPTICS 算法.....	11
2.3 网格聚类原理.....	12
2.3.1 GIRDCLUS 和 BANG 算法	12
2.3.2 CLIQUE 算法.....	14
2.4 聚类质量评价.....	16
2.4.1 调兰德指数.....	16
2.4.2 调互信息.....	17
2.5 本章小结.....	18
第 3 章 一种新的网格聚类模型及 GMCLU 算法	19
3.1 引言.....	19
3.2 网格聚类模型设计.....	20
3.2.1 空间的网格划分.....	20
3.2.2 网格密度的定义.....	22
3.2.3 网格聚类原则.....	22
3.3 GMCLU 算法实现	24
3.3.1 算法实现.....	24

3.3.2 算法复杂度分析.....	27
3.4 实验分析.....	28
3.4.1 实验环境.....	28
3.4.2 实验数据集.....	28
3.4.3 聚类质量分析.....	29
3.4.4 聚类效率分析.....	31
3.5 本章小结.....	36
第 4 章 基于树的空间划分和子聚类合并的网格聚类算法.....	38
4.1 引言.....	38
4.2 GTCLU 框架设计	39
4.2.1 空间划分和树的构建.....	39
4.2.2 基于空间划分树的聚类.....	41
4.2.3 并行化计算及 GTCLU 的扩展应用	43
4.3 GTCLU 算法实现	43
4.3.1 算法实现.....	43
4.3.2 算法复杂度分析.....	47
4.4 实验分析.....	48
4.4.1 实验环境.....	48
4.4.2 实验数据集.....	48
4.4.3 聚类质量分析.....	50
4.4.4 聚类效率分析.....	51
4.5 本章小结.....	57
第 5 章 总结与展望.....	58
5.1 研究工作总结.....	58
5.2 展望.....	59
参考文献.....	60
攻读学位期间主要的研究成果.....	68
致 谢.....	69

第1章 绪论

1.1 研究背景及意义

在当今信息化时代,数据已逐渐成为至关重要的资产之一。大数据作为这一时代的显著特征,表现为海量的、多样化的、高速增长的数据。这些数据来自传感器、社交媒体、移动设备、应用程序和物联网等多样化的设备,其中涵盖了个人信息、行为习惯、购物偏好、社交网络和地理位置等多方面的内容。有效的利用大数据可以为人们带来了许多益处。企业和组织可以通过分析大数据深入了解客户需求、市场趋势和业务运营状况,从而制定更精准的战略和决策。公共服务机构可以利用大数据改善交通规划、医疗保健和教育质量等民生工程,提高公共服务的质量和效率。科研机构可以通过分析海量数据更好地理解复杂现象,发现潜在规律,从而推动各领域的知识进步。因此,如何利用有限的计算资源从大数据中提取出有价值的信息,已经成为了国内外近年来的研究重点^[1-3]。

聚类分析作为机器学习和数据挖掘领域的重要研究方向,在大数据分析和研究中具有广泛的应用价值。聚类分析是一种无监督学习方法,其算法能够自动地将无标签数据集划分为若干个类,其目标是使被聚在同一类中的数据尽可能相似,而被聚在不同类中的数据尽可能相异^[4,5]。在过去的几十年里,国内外学者针对聚类分析进行了大量的研究,提出了许多聚类算法。根据数据相似性的不同度量方法以及聚类过程的不同设计思想,传统的聚类算法可以大致划分为以下几类经典模型:以 k -means^[6] 为代表的划分聚类模型,以 SLINK^[7] 为代表的层次聚类模型,以及以 DBSCAN^[8] 为代表的密度聚类模型^[9-11]。

密度聚类作为聚类分析领域的一个热门研究方向,因其具有能够识别任意形状的类、有效的识别噪音数据以及可以自动探测类的数量等优势而在现实场景中得到了广泛的应用。众多知名的机器学习工具库,如 Scikit-learn^[12]、ELKI^[13]、Weka^[14]、Spark^[15] 和 R^[16] 等,均实现了多种经典密度聚类算法。密度聚类定义一个点的密度为其 ϵ 半径邻域内点的数量,将紧密相连的高密度点聚在同一个类中,将低密度的离群点视为噪音点。尽管密度聚类方法能够广泛的适用于不同分布形态的数据集并得到良好的聚类效果,但其算法的时间复杂度普遍较高,导致其在大数据集聚类分析中的应用受限^[17-21]。

网格聚类是密度聚类的一种重要的衍生方法,旨在提高密度聚类的计算效率。网格聚类将原始的数据空间划分为由网格单元构成的网格空间,以网格及网格密度代替原始空间中的点和点密度,进而基于网格密度和网格距离实现聚类。相较于传统密度聚类方法,网格聚类利用单个网格单元替代多个点,可以有效的降低

聚类样本集的规模,从而提高聚类速度^[22-27]。然而,与密度聚类相比,网格聚类通常会导致聚类质量下降。网格聚类质量通常受网格划分、网格密度设计、网格距离度量及网格聚类策略等多种因素影响,因此设计合理的网格模型成为了一项艰巨的任务。此外,网格模型的聚类速度与数据维度密切相关。数据维度增加会导致网格数量急剧上升,从而降低其聚类速度。因此,在大数据应用背景下,如何在保持高聚类质量的同时,进一步提高网格模型在高维大数据集中的运行效率,仍是当前持续讨论的热门研究课题^[28-31]。

鉴于密度聚类在大数据应用中所面临的以上诸多挑战,本文的主要关注点在于提高网格模型的聚类质量以及在高维大数据下的聚类效率。为了改善网格模型的聚类质量,本文提出了一种新的网格聚类模型,该模型对网格划分、网格密度、网格距离和网格聚类原则等核心问题进行了重新定义和改进,从而有效提高了网格模型的聚类质量。此外,在此模型基础上,本文提出了适用于低维数据的 GMCLU 算法,该算法在低维空间下能够达到 $O(n)$ 的时间复杂度。为了提高网格模型在高维数据空间中的聚类效率,本文提出了一种基于树的空间划分及子聚类高效合并的聚类框架,并在此框架基础上实现了 GTCLU 网格聚类算法。该算法通过分治策略有效降低了网格聚类的计算规模,在保证聚类质量的同时,能够高效的完成高维大数据集的聚类分析。

1.2 国内外研究现状

密度聚类在数据挖掘和机器学习等领域已得到广泛应用。密度聚类的概念最早由 Ester 等人^[8] 在 DBSCAN 算法中提出。该算法需要输入两个参数 ϵ 和 $minPts$, 并定义一个点的密度为以该点为中心、 ϵ 为半径的邻域内点的数量,定义密度大于或等于 $minPts$ 的点为核心点,小于 $minPts$ 的点为边缘点。DBSCAN 的聚类过程主要围绕数据集中的核心点展开。根据 ϵ -邻域内的核心点归属于同一个类的原则,该算法首先将数据集中的核心点划分为若干个类,然后将数据集中的边缘点归入其 ϵ -邻域内核心点所在的类中,最后未被聚类的边缘点被视为噪音点。DBSCAN 具有自动探测数据集中类的数量、识别任意形状类以及有效处理噪音点等优势。

自 DBSCAN 提出以来,该方法受到了广泛关注,许多学者在密度聚类方面进行了大量研究。Hinneburg 等人^[32] 提出了 DENCLU 算法,引入了“影响力函数”概念,通过高斯混合模型对数据集的密度函数进行建模,将密度函数的峰值作为潜在的聚类中心,并通过将数据点向聚类中心点聚合的方法来完成聚类。Ankerst 等人^[33] 提出的 OPTICS 是基于 DBSCAN 的一种扩展算法,主要解决了 DBSCAN 无法适用于不同密度分布的数据集的问题。为了解决这一问题,

OPTICS 的核心思想是不生产一个明确的聚类结果,而是创建一个关于密度聚类的层次结构,其中每一层表示不同密度参数下的聚类结果,最后通过图形化帮助分析以选取更合理的聚类结果。Campello 等人^[34]改进了 OPTICS,结合层次聚类提出了 HDBSCAN,该算法用一棵最小生成树将重要的类组建成层次的结构,然后通过一种衡量类稳定性的方法,从这个层次结构中提取出最佳的聚类结果。HDBSCAN 算法对输入的参数不敏感,适用于不同密度分布的数据集,但需要 $O(n^2)$ 的时间复杂度。Zhou 等人^[35]提出了 SDBSCAN 算法,该算法通过引入采样技术,有效地提高了密度聚类的聚类速度,并降低了其内存空间的占用。但 SDBSCAN 没有解决 DBSCAN 不适用于不同密度分布数据集的问题。Borah 等人^[36]同样提出了基于采样技术的密度聚类算法 IDBSCAN 来加速 DBSCAN 算法的运行速度。Tsai 等人^[37]将 k -means 算法与 IDBSCAN 算法相结合,提出了 KIDBSCAN。该算法首先通过 k -means 算法找到 k 个高密度中心点及其周围邻近点作为采样集,然后在该采样集中调用 IDBSCAN 算法生成聚类结果。与此类似,Gholizadeh 等人^[38]结合 k -means++与密度聚类提出了 K-DABSCAN。该算法在 k -means++生成的类中应用 DBSCAN 进行密度聚类,可以有效降低密度聚类的数据规模,提高其运行效率。Ertöz 等人^[39]结合共享最近邻算法(Shared Nearest Neighbor, SNN)和密度聚类,提出了 SNN-DBSCAN。该算法不需要预设半径和密度阈值等参数,而是通过 SNN 计算数据点的相似性并建立点之间的相似性矩阵,然后通过相似性矩阵对数据集进行聚类。SNN-DBSCAN 可以有效解决 DBSCAN 无法处理密度分布变化的数据集的缺陷,但由于相似性矩阵计算复杂度太高,导致该算法难以被应用于大数据集的聚类。为了进一步提高密度聚类的速度,分布式计算被应用到了密度聚类中,该方案虽然能在一定程度上解决密度聚类在大数据集上的应用问题,但需要更多的计算资源支撑^[40-45]。数据流处理是当前数据挖掘领域的一个热门研究课题,密度聚类算法也被大量应用于数据流分析中,这些研究尝试将密度聚类与在线学习、增量式更新等技术相结合,以适应动态变化的数据流场景^[46-52]。

网格聚类是密度聚类的一个衍生方向,可以极大的加快密度聚类的聚类速度,自提出以来得到了大量学者的研究发展,逐渐演变成密度聚类最重要的分支之一。Schikuta 等人^[53]首先提出了网格聚类算法 GRIDCLUS,该算法将原始的数据空间划分成为由超立方体网格组成的网格空间,将数据点插入到其所属的网格中并根据网格内点的数量计算网格的密度。随后,算法根据网格密度从大到小对网格进行排序,从最大密度网格开始以合并邻近网格的方法完成网格聚类。为了解决 GRIDCLUS 算法在网格大小设计、网格管理维护及邻近网格搜索等方面的低效问题, Schikuta 等人^[54]随后又提出了改进版算法 BANG。BANG 算法的设计思

想与 GRIDCLUS 一致,但聚类效率更高,更适合大数据集聚类。Agrawal 等人^[55]依据先验原理提出了 CLIQUE 算法。先验原理描述了高维空间与其直接映射的低维空间之间的关系,在高维空间上的密集网格在其直接映射低维空间上依然是密集网格,在高维空间上属于一个类中的网格在其直接映射低维空间上依然属于同一个类。基于该原理,CLIQUE 算法从原始空间直接映射的低维空间开始聚类,并逐渐从低维空间向高维空间过渡,直到在原始网格空间中生成最终的聚类结果。Wang 等人^[56]结合统计技术提出了 STING 算法,该算法在网格中保存了均值、标准差和数据分布等统计信息,并将这些信息应用于网格搜索和聚类。基于 STING 算法,Wang 等人^[57]进一步改进,又提出了 STING+算法以应对动态数据集的聚类分析。Hinneburg 等人^[58]对网格划分方法进行优化,提出了 OPTIGRID 算法,能够更好的应对高维数据的网格聚类问题。Sheikholeslami 等人^[59]提出了 WaveCluster 算法,利用小波变换的多分辨率特性在高维空间进行聚类。该算法对带噪声的数据集具有较强的鲁棒性,但在高维空间上聚类速度有所减缓。Aggarwal 等人^[60]提出的 IPCLUS 算法允许人工干预聚类过程,可以得到更符合人类直觉的聚类结果,但依赖不准确的直觉进行干预可能会导致较差的聚类结果。Loh 等人^[61]提出的 GSCAN 算法利用网格技术减少了冗余的点距离计算,并利用 GPU 计算进一步加速聚类过程。Chen 等人^[62]的 D-Stream 算法将网格聚类应用于数据流的处理。该算法分为线上处理和线下处理两个阶段,线上算法利用网格对数据流进行持续统计,而线下算法对线上的网格进行聚类以获取当前数据流的聚类结果。此后,许多基于网格密度技术的数据流聚类算法被提出以应用于实时数据流的在线挖掘,如 StreamSW^[63]、FGCH^[64]、EXCC^[65]、Mudi-Stream^[66]、GRIDEN^[67]、DGStream^[68]、ESA-Stream^[69]等。

1.3 本文研究内容

聚类分析在机器学习和数据挖掘等领域得到了广泛应用。在众多聚类方法中,密度聚类因具备自动识别类的数量、识别任意形状的类以及良好探测噪音数据等优势,成为聚类分析领域中一个重要的研究方向。然而,密度聚类算法普遍存在时间复杂度较高的问题,导致其难以应用于大数据集的聚类分析。

网格聚类作为密度聚类的衍生方向,能有效地提高其聚类速度。尽管如此,网格聚类仍存在两个明显的缺陷:其一,与密度聚类相比,网格模型的聚类质量普遍较低;其二,网格模型的聚类速度随着数据维度的增长而急剧下降,通常无法应用于高维度的大数据集。为解决网格聚类所面临的这两个问题,本文将从以下两个方面进行探讨:(1)针对网格聚类的质量下降问题进行研究;(2)研究网格聚类在高维空间下的聚类效率问题。

(1) 网格聚类质量问题研究与低维空间网格聚类

本文设计了一种新的网格聚类模型,对网格的划分、网格密度、网格距离及网格聚类规则等关键因素进行了重新定义和设计,可以有效地提高网格模型的聚类质量。在此网格模型设计的基础上,本文同时提出了一种适用于低维大数据集的高效网格聚类算法 GMCLU,该算法在低维空间下能够达到 $O(n)$ 的时间复杂度和 $O(m)$ 的空间复杂度,其中 $m \ll n$ 表示非空网格的数量。

网格聚类质量受到多重因素的共同影响,包括网格划分、网格密度设计、网格距离设计以及网格聚类规则等。为了更好地将网格聚类与传统密度聚类相匹配,本模型采用了密度聚类常用的 ϵ 半径阈值与 *minPts* 密度阈值两个参数。模型根据 ϵ 半径参数自动计算出网格宽度来将原始数据空间网格化,同时保证了每个网格内最远两点之间的距离不超过 ϵ 半径阈值。在设计网格密度时,除了计算网格内点的数量,本模型还考虑了其周围邻近网格对该网格密度所产生的影响,并设计了一个影响力函数来计算网格与其周围邻近网格的综合密度。这一设计使得网格密度的计算更加合理,能够有效地防止核心网格被误判为边缘网格。本模型在网格中生成了点的位置摘要信息,能够通过计算两个网格内点的距离来替代网格距离,使得网格之间的聚类更加精确。在以上模型的设计基础上,本文提出了适用于低维空间的 GMCLU 聚类算法,并对其进行了系统的实验测试,证明了该网格模型能有效地提高网格聚类质量,且该算法能够高效地对低维大数据集完成聚类分析。

(2) 高维空间下的网格聚类效率问题研究

本文提出了一种基于树的空间划分和子聚类合并的网格聚类框架,并在此框架基础上设计了 GTCLU 聚类算法。通过系统性的实验测试,本文证明了 GTCLU 算法在保持高质量聚类结果的同时,能够显著地提高网格聚类在高维空间下的运行效率,为高维大数据的聚类分析提供了一种可行且有效的解决方案。

网格模型的聚类效率受网格数量的直接影响,而网格空间的大小与维度呈指数增长关系。在众多网格聚类算法中,寻找网格邻居的过程通常耗时最长,尽管通过 *kd-tree* 等索引结构能够一定程度上提高该过程的计算效率,但对于高维大数据集来说,其所需时间仍然是不可接受的。对高维数据进行降维处理是一种普遍的解决策略,但这将导致数据的信息损失和聚类质量下降。为了解决高维空间下的网格聚类问题,本文基于分治的思想,提出了一种基于树的空间划分和子聚类合并的网格聚类框架,并在此框架的设计基础上实现了 GTCLU 聚类算法。GTCLU 通过一棵二叉树将原本的网格空间划分成为众多更小的子网格空间,并将这些子空间存储在树的叶子节点中,树的非叶子节点只需保存少量的网格信息用于后续的子聚类合并。通过在各个叶子节点上进行网格聚类,可以快速得到

各个子空间上的网格聚类结果。然后在树的非叶子节点上由底至上地对这些子聚类进行快速合并，最终在根节点得到整个网格空间的聚类结果。GTCLU 在非叶子节点上的聚类合并过程非常迅速，只需要根据节点上保存的有限信息进行快速计算即可完成子聚类的合并过程。此外，GTCLU 在各个叶子节点上的聚类过程以及在树的同一层上的各个非叶子节点的子聚类合并过程均是相互独立的，因此可以轻易地采用并行化计算来进一步加速 GTCLU 的聚类过程。本文对 GTCLU 进行了系统的实验测试，结果表明该算法能够有效地加速高维空间下的网格聚类过程，同时还能获得比传统网格模型更高的聚类质量。

1.4 论文组织结构

本文共分为五章，文章组织结构如下：

第 1 章：首先介绍聚类分析的研究背景和应用场景，接着讨论密度聚类和网格聚类的优势及存在的问题，并详细阐述了密度聚类和网格聚类的研究现状。最后针对网格模型聚类质量下降和在高维大数据中应用受限等问题，提出了本文的解决方案。

第 2 章：首先介绍聚类分析的定义和三个经典的聚类模型。接着结合实际算法详细阐述了密度聚类和网格聚类的相关原理，并分析了各自的优缺点。最后介绍了一些常用的聚类质量评价指标。

第 3 章：首先分析导致传统网格算法聚类质量不佳的原因，然后提出了本文的解决方案。接着从网格空间划分、网格密度定义、网格聚类原则三个维度详细介绍了本文的网格聚类模型设计。最后，根据新模型设计了适用于低维大数据的聚类算法 GMCLU，并对其进行了系统的实验验证。

第 4 章：首先讨论网格聚类模型在高维大数据集下面临的聚类效率问题，然后提出了 GTCLU 解决方案。接着从空间划分及其树的构造、基于树的聚类、并行化计算及其他可能的扩展性应用三个维度详细阐述 GTCLU 聚类框架。最后介绍了 GTCLU 算法的详细实现过程并对其进行了系统的实验验证。

第 5 章：首先介绍了本文关于网格模型的聚类质量和聚类效率提升等方面的相关研究内容及主要贡献，然后展望了后续可能的进一步研究方向。

第 2 章 相关理论基础

本章介绍聚类分析相关的理论基础。首先介绍聚类的定义及几种典型的聚类模型，然后结合具体的算法详细介绍密度聚类和网格聚类的相关原理，最后介绍几种常用的聚类评价指标。

2.1 聚类分析概述

聚类分析是一种无监督学习方法，是统计数据分析中的一种常用技术，被广泛的应用于模式识别、信息检索、生物信息学、图像分析、数据挖掘和机器学习等诸多领域。

聚类分析可以认为是一种统计数据的思想，不能简单地将其归纳为一个精确的、唯一的定义，这也是为什么存在如此多不同类别的聚类模型的原因。但是聚类分析有一个共同的思想，即根据数据的相似性对数据集进行分类。本文将这个聚类思想概括为：根据数据的相似性，利用聚类算法将一个给定的数据集自动分为若干个类，其目标是使被分在同一个类中的数据尽可能相似，而被分在不同类中的数据尽可能相异。

作为一种无监督学习方法，聚类分析无需训练数据集，且研究者通常无法提前知道被聚类数据集中类的数目，聚类算法能够自动地在数据集中得到聚类结果。不同的聚类算法在数据相似性、聚类规则等方面有不同的设计，且在同一个数据集下通常会得到不同的聚类结果。根据数据相似性的测量方式以及聚类规则的不同设计，可以大致将众多的传统聚类算法归纳为划分聚类、层次聚类和密度聚类三个经典的聚类模型^[70-72]。

2.1.1 划分聚类

划分聚类(Partition Clustering)又称作基于中心聚类(Center-based Clustering)，是一种非常高效的聚类模型。划分聚类通常定义一个明确的目标函数以衡量聚类结果的优劣，并通过最优化目标函数来实现聚类目标。划分聚类又可以分为 k -means、 k -medians、 k -medoids 等经典问题。本文将通过经典的 k -means 问题来阐述划分聚类的基本思想。

$$F(S) = \sum_{i=1}^k \sum_{x \in S_i} \|x - \mu_i\|^2, \quad \text{其中 } \mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x \quad (2-1)$$

k -means 问题可以定义为：给定一个 d 维的数据集 $D = \{x_1, \dots, x_n\}$ 和一个参

数 k , 目标是将数据集 D 划分称为 k 个类 $S = \{S_1, \dots, S_n\}$, 使得目标函数 $F(S)$ 最小。目标函数 $F(S)$ 表示各个点到其所属类中心的距离之和, 其定义如公式(2-1)所示。

Lloyd 算法^[6] 是解决 k -means 问题最经典的方法, 其算法流程可概括为以下几个步骤:

- (1) 从数据集 D 中随机选取 k 个点作为 k 个类的初始中心。
- (2) 对于任意点 $x \in D$, 在 k 个点中选择距离它最近的点作为该点的类中心, 并将该点划分到对应的类中。
- (3) 重新计算 k 个类中所有点的均值并作为新的 k 个类中心。
- (4) 重复步骤 (2) 和步骤 (3), 一直到 k 个类的中心点收敛不变为止。

划分聚类具有实现简单、聚类速度快和能够得到理论保证等优势。然而划分聚类需要输入 k 个初始中心值, 而在实际应用中, 对 k 值进行合理判断通常具有挑战性。同时, 其聚类结果对初始中心点的选择非常敏感, 而选取合适的 k 个中心值同样困难。此外, 划分聚类实际上假设了类的形状为球体, 对于包含非球体形状类的数据集, 划分聚类通常会得到较差的聚类结果。划分聚类对噪声数据非常敏感, 噪声数据的存在也会极大的影响聚类结果的质量。因此, 尽管划分聚类具有较高的聚类效率, 但其实际应用场景非常受到数据集分布特征的限制^[73,74]。

2.1.2 层次聚类

层次聚类 (Hierarchical Clustering) 的核心思想是认为数据集中距离较近的点比距离较远的点更具有相似性。层次聚类首先将数据集中的每个点视为一个独立的类, 然后根据距离更近更相似的原则, 迭代地将数据集中距离最近的类进行合并, 并在合并的过程中生成一棵树状的结构, 直到整个数据集在树的根节点合并为一个类。层次聚类不输出单一的聚类结果, 而是生成一个树状层次结构。树的每一层表示在特定距离阈值下的聚类结果, 用户可以根据需求从树中选择某一层的聚类作为目标结果。层次聚类可进一步细分为单链接聚类 (Single-linkage Clustering)、全链接聚类 (Complete-linkage Clustering)、平均链接聚类 (Average-linkage Clustering) 等经典问题。本文将通过单链接聚类中著名的 SLINK^[7] 算法来阐述层次聚类的基本原理。

SLINK 算法定义两个类之间的距离为它们之间最近两点的距离。首先, 将初始数据集中的每个点视为一个独立的类, 计算各个类之间的距离并生成一个 $n \times n$ 的二维距离矩阵 M 。接下来的算法步骤如下:

- (1) 将初始的 n 个类作为树的叶子节点。
- (2) 查询距离矩阵 M , 找出距离最小的两个类并在树中将它们合并为新的

类。然后更新距离矩阵：从矩阵中删除被合并类的行和列，并插入合并后的新类。同时，记录当前层的聚类结果。

(3) 重复步骤(2)，记录每一层的聚类结果，直到所有的点被合并到同一个类中。

以上的聚类结果可以根据其合并过程生成一个可视化的单链聚类树状图，便于研究者对聚类结果做进一步的分析。

层次聚类不需要像划分聚类那样预判类的数量，并且可以通过一个可视化的树状聚类图来描述类之间的关系，便于对数据进行更深入的分析。但是层次聚类的算法时间复杂度较高，难以被应用于大数据集的聚类分析。且当树状聚类结构过于复杂时，从树状图中准确判断聚类结果可能会变得困难^[75,76]。

2.1.3 密度聚类

密度聚类(Density-based Clustering)的核心概念在于认为同一类中的数据点应分布在连续的高密度区域内，而不同类之间应该存在密度差异或距离间隔。密度聚类主要包含点密度聚类和网格密度聚类两个方向。点密度聚类通常定义一个点的密度为其给定的半径邻域内点的数量，并将具有较高密度且距离较近的数据点归为同一个类。网格聚类是点密度聚类的一个重要的衍生方向。网格聚类将原本的数据空间划分称为超立方体网格组成的网格空间，并根据网格内点的数量来定义网格密度，最后对高密度的网格进行聚类。

密度聚类不需要像划分聚类那样预判类的数量，可以通过算法自动的得到若干个类。同时密度聚类对数据的分布没有要求，可以探测任意形状类，且其对噪音数据不敏感，能够有效的识别噪音点。

为了详细的介绍密度聚类的原理，后文中将介绍两个经典的密度聚类算法 DBSCAN^[8] 和 OPTICS^[33]，以及两个被广泛使用的网格聚类算法 BANG^[54] 和 CLIQUE^[55]，并将它们作为后续实验的对比算法。

2.2 密度聚类原理

许多知名的聚类算法需要对数据的分布形态做出假设，只能满足特定的数据集聚类。如 k -means 等划分算法实际上假设数据集中的类是符合高斯分布的超球体，并且需要对类的数量做出预判。而现实世界的数据集分布形态各异，且包含大量噪音数据，同时研究者无法预知数据集中类的数量。因此，现实世界需要一个能够自动探测类的数量，能够识别任意形状类且对噪音数据不敏感的聚类算法。密度聚类的提出则满足了这些现实世界数据的聚类需求。

密度聚类通常定义一个点的密度为以该点为中心的局域内点的数量，并根据给定的密度阈值将各点划分称为高密度点和低密度点。密度聚类的核心思想是认

为同一个类应该由距离较近的高密度点组成,而不同类之间应该存在较大的密度差异或距离差距。密度聚类根据其聚类思想自动地将数据集划分为若干个类,且能够良好的识别任意形状的和噪音数据。密度聚类模型拥有许多经典算法,本文从中总结出密度聚类算法设计需要解决的一些关键问题:

- (1) 如何定义并计算点的密度?
- (2) 应该基于怎样的规则对高密度区域进行聚类?
- (3) 如何提升密度聚类的聚类效率?

不同的算法针对以上问题提出了不同的解决方案。接下来,本文将通过 DBSCAN 和 OPTICS 两个经典算法来详细介绍密度聚类的原理。

2.2.1 DBSCAN 算法

DBSCAN (Density-based Spatial Clustering of Applications with Noise) 是一种经典的密度聚类算法,由 Ester 等人^[8]首次提出。自提出以来, DBSCAN 一直受到持续关注和研究,并被广泛应用于数据挖掘实践。2014 年, DBSCAN 获得了知名数据挖掘会议 SIGKDD 颁发的时间测试奖项。Schubert 等人^[77]于 2020 年发表了一篇文章,强调了 DBSCAN 在数据挖掘实践中的重要作用,认为 DBSCAN 仍然是最优秀的聚类算法之一,该文章的观点受到了广泛的认同。因此,本文将以 DBSCAN 为例详细介绍密度聚类算法原理,并将其作为实验中的对比算法之一。

DBSCAN 算法需要输入两个参数: ϵ 和 $minPts$ 。定义点 p 的密度为以 p 为中心、 ϵ 为半径的局域内点的数量,并将这个局域称为点 p 的 ϵ -邻域。基于点的密度和 ϵ -邻域的概念, DBSCAN 提出了以下定义:

核心点: 如果一个点 p 的密度大于或等于 $minPts$, 则将这个点定义为核心点。

边缘点: 如果一个点 p 的密度小于 $minPts$, 则将这个点定义为边缘点。

密度直达: 如果一个核心点 q 的 ϵ -邻域内包含另一个核心点 p , 则称点 p 可密度直达点 q 。密度直达要求两个点都是核心点。

密度可达: 如果存在一个核心点 p 和任意点 q , 且存在一条密度直达的路径 p, p_1, \dots, p_n , 若点 q 在点 p_n 的 ϵ -邻域内, 则称点 p 密度可达点 q 。其中的密度直达路径意味着这条路径上任意两个相邻的点 p_{k-1} 、 p_k 都是核心点, 且 p_k 在 p_{k-1} 的 ϵ -邻域内。密度可达要求一个点是核心点, 另外一个点可以是边缘点。

DBSCAN 要求一个类中至少包含一个核心点。对于一个核心点, 它所有密度可达的点都应该与它被聚在同一个类中。如果一个边缘点不能从任意的核心点密度可达, 则将这个边缘点视为噪音点。如图 2-1 所示, 当 $minPts$ 设置为 4 时,

红色点的密度均大于或等于 4，被定义为核心点，其他点被定义为边缘点。从核心点 a 出发，将所有从 a 密度可达的点都与 a 归为同一个类，即图中左边区域的红色点和蓝色点。同理，将右边区域的红色点和蓝色点归在同一个类中。边缘点 c 因为从任意核心点出发都密度不可达，因此被归为噪音点。

根据以上的聚类思想，DBSCAN 的算法可以归纳如下：

- (1) 计算数据集中每个点密度，并根据其密度区分核心点和边缘点。
- (2) 从任意未被聚类的核心点出发，开启一个新的类，并将它所有密度可达的点聚在这个类中。
- (3) 重复步骤 (2)，直到所有核心点已被聚类。
- (4) 最后将没有被聚类的边缘点标记为噪音点。

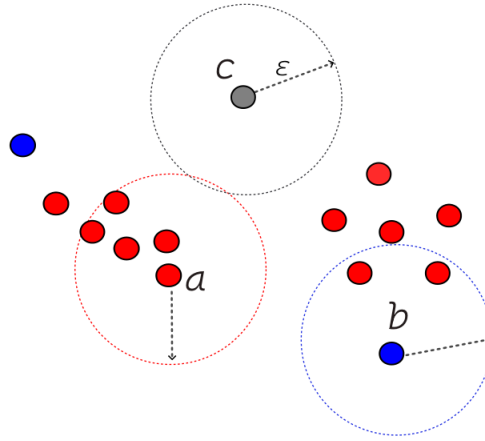


图 2-1 DBSCAN 算法示意图 ($minPts = 4$)

附注：红色的点表示核心点，其他颜色的点表示边缘点。灰色的 c 点从任意核心点都密度不可达，被视为噪音点。

2.2.2 OPTICS 算法

OPTICS (Ordering Points to Identify the Clustering Structure) ^[33] 是一种从 DBSCAN 扩展而来的密度聚类算法，主要解决了 DBSCAN 对于 ϵ 参数过于敏感的问题，使其更适用于包含不同密度分布的数据集。

与 DBSCAN 一样，OPTICS 需要输入 ϵ 和 $minPts$ 两个参数，并根据这两个参数定义点的密度、核心点、边缘点和密度直达等概念。在此基础上，OPTICS 还引入了核心距离 (core-dist) 和可达距离 (reachability-dist) 两个新概念。

核心距离：核心距离仅针对核心点而言，边缘点没有核心距离。如果一个点 p 是核心点，则将点 p 的核心距离定义为以点 p 为中心绘制超球体局域的最小半径，使得该局域内恰好包含 $minPts$ 个点，记为 $core-dist(p)$ 。

可达距离：仅当点 p 是一个核心点时，点 p 才存在可达距离。设存在一个核心点 p 和任意点 o ，则定义点 p 到点 o 的可达距离为点 p 的核心距离与 p 到 o 的欧式距离中的较大值，即 $\max(\text{core-dist}(p), \text{dist}(p, o))$ 。

给定两个参数 ϵ 、 minPts 以及数据集 D ，OPTICS 算法首先计算每个点的密度并标记核心点和边缘点。接下来，OPTICS 的算法步骤可以分为以下几个阶段：

(1) 初始化有序队列和结果队列两个空队列。有序队列用于存储核心点及其密度直达的点，并将其中的点按可达距离的升序维持排序。结果队列中用来存储已经处理完成的点。

(2) 从数据集中任选取一个未处理的核心点，待处理。若所有核心点均已处理完成，则算法结束。

(3) 对于被选取的核心点，找到其所有密度直达的点，并将这些点（包括被选取的核心点）中未处理且不在有序队列中的点按可达距离的升序排列加入到有序队列中。

(4) 若有序队列为空，返回步骤 (2)。否则取出有序队列中的第一个点，将其标记为已处理状态并放入结果队列中，返回步骤 (3)。

在算法结束后，可以根据结果队列并以横坐标为处理点的顺序排列、纵坐标为各点的可达距离绘制一个聚类直方图。通过该直方图，可以辅助判断选取一个可达距离的阈值来输出最终的聚类结果。这种方法使得 OPTICS 算法能够适应不同密度分布的数据集，提高聚类效果的准确性和可靠性。

2.3 网格聚类原理

网格聚类可以视为一种扩展型密度聚类模型，其相较于传统密度聚类具有更高的聚类效率。该方法将原始数据空间划分为有限数量的网格组成的网格空间，并对这些网格进行聚类。与密度聚类类似，网格聚类依据网格内点的数量计算网格密度，将高密度且紧密相连的网格划分为同一类，而不同类之间的网格则存在密度差异或距离间隔。网格聚类主要需要解决以下问题：

- (1) 如何对数据空间进行网格化，以及用什么方法来表示网格？
- (2) 如何定义网格密度以及根据网格密度区分重要网格？
- (3) 如何依据网格密度及网格距离进行聚类？同一类或不同类别之间的网格应满足何种条件？

不同的算法针对上述问题提出了不同的解决方案。本节将通过几个经典的网格聚类算法来介绍网格聚类的基本原理。

2.3.1 GIRDCLUS 和 BANG 算法

GRIDCLUS (Grid-based Hierarchical Clustering) 算法^[53] 是众多网格模型中

最具代表性的算法之一，通过对该算法进行分析，可以深入理解网格聚类中网格划分、网格密度、聚类规则等核心问题的基本原理。

GRIDCLUS 采用一种名为网格文件（Grid File）的数据结构将原始的数据空间进行网格化。网格文件是一个嵌套列表结构，包含 d 个列表，每个列表包含了当前维度的划分信息。GRIDCLUS 在内存中维护一个网格结构，该网格结构由 d 个维度划分信息（Scales）、一个网格簿（Grid Directory）和一个数据块（Data Block）的集合组成。网格和数据块均为 d 维超矩形结构。数据块由一个或多个非空网格组成，且一个网格只能属于一个数据块。数据块集合包含了数据空间中的所有点。一个数据块必须包含至少一个点，且其所包含的点不应该超过一个上限，否则将这个数据块划分成为更小的数据块。图 2-2 表示了一个二维的网格结构。

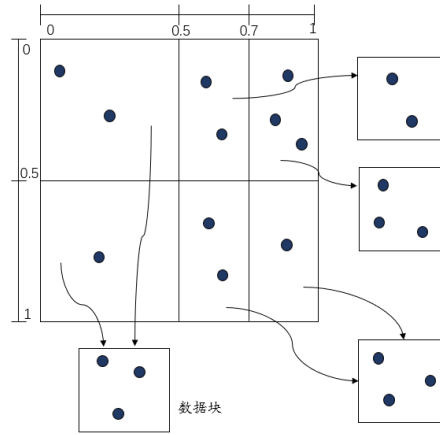


图 2-2 二维网格结构示意图

GRIDCLUS 通过数据块进行聚类，并将一个数据块的密度定义为其包含的点的数量与数据块体积的比值，如公式（2-2）所示：

$$density_b = \frac{p_b}{V_b} \quad (2-2)$$

其中 p_b 表示数据块 b 内点的数量， V_b 表示数据块的体积。

GRIDCLUS 算法对数据块集合按密度降序排列，每次选取密度最大的数据块作为类的中心块进行聚类。若存在多个密度相等的最大密度数据块，它们将全部被选取。随后，算法根据这些中心块以合并邻居的方式进行聚类。算法采用多层聚类策略，每一层代表在特定密度阈值下的聚类结果，该密度阈值随着层次增加而递减。其算法步骤可描述如下：

(1) 将原始数据集转换为网格结构，并按密度降序对数据块排序。设置聚类层次为第 0 层。目前所有数据块均被标记为未激活且未聚类状态。

(2) 如果所有数据块均已被激活，则算法结束。否则，从未激活的数据块中取出密度最大的多个数据块并激活它们。

(3) 从这些最大密度数据块中取出一个未被聚类的数据块，开启一个新的类并将其标记为已聚类状态。如果不存在这样的块，跳到步骤 (5)。

(4) 对于这个新的类，搜索类中每个数据块已被激活且未被聚类的邻居块，将它们标记为已聚类状态并加入到这个类中，直到不存在这样的邻居块为止。返回步骤 (3)。

(5) 将所有未激活的数据块都设置为新类。此时，已得到当前层的聚类结果并将其保存。

(6) 将所有数据块标记为未聚类状态，将层次标记加 1，返回步骤 (2) 以进入下一层聚类。

可以看出，GRIDCLUS 最终生成了类似于层次聚类的结果，一次性得到了多层聚类，可以根据需求提取特定层次的聚类作为最终结果。

BANG 算法^[54] 从 GRIDCLUS 扩展而来，其聚类原理与 GRIDCLUS 一致。BANG 主要对 GRIDCLUS 在网格划分、邻居搜索、数据块管理等方面存在的低效性问题做出了改进，提高了 GRIDCLUS 在大数据集上的聚类效率。

2.3.2 CLIQUE 算法

CLIQUE (Clustering in Quest) 算法^[55] 是一种将网格聚类与子空间聚类相结合的方法，该算法能够自动地对数据集的内嵌子空间进行聚类，获得感兴趣的子空间结果，并能从低维子空间聚类传导到更高维的空间聚类。在这里，内嵌子空间是指从高维直接映射到低维的空间，而无需对维度进行任何变换。

对于一个 d 维空间的数据集，CLIQUE 首先根据一个输入值 l 将每个维度平均划分成 l 等份，则原始的数据空间被划分成由 l^d 个超立方体网格组成的网格空间。对于任意的点，可以根据坐标值将其加入到相应的网格中。CLIQUE 将网格密度定义为网格内点的数量，若一个网格的密度超过给定的密度阈值，则将该网格定义为密集网格。

CLIQUE 利用密集网格对网格空间进行聚类。如果两个密集网格 a 和 b 存在共同的切面，则 a 和 b 被分到同一个类中。类似地，如果一个密集网格 a 与密集网格 b 有共同的切面，且 b 与密集网格 c 有共同切面，则 a 与 c 也应该被分到同一个类中。基于此原理，对于任意的子网格空间，可以利用贪心的方法对其完成聚类：从任意一个未被聚类的密集网格开始，开启一个新的类，利用贪心的方法

遍历当前类中网格的邻居网格，并将其全部加入到这个类中，直到不存在这样的邻居网格为止。对每一个未被聚类的密集网格重复上述过程，即可完成对该网格空间的聚类。

CLIQUE 通过最小描述 (Minimal Description) 来表示一个类。对于一个类 C ，如果存在一个超矩形空间 $R \subseteq C$ ，则称 R 为一个区域 (Region)，若同时不存在更大区域 U 满足 $U \supset R$ ，则称 R 为一个最大区域 (Maximal Region)。如果存在 x 个最大区域刚好可以覆盖一个类的全部网格，且任意 $x-1$ 个最大区域均无法覆盖整个类，则将这 x 个最大区域所组成的集合称为这个类的最小描述。如图 2-3 所示，其中的阴影网格构成了一个类， A 和 B 区域为这个类的两个最大区域， $A \cup B$ 即为这个类的最小描述。

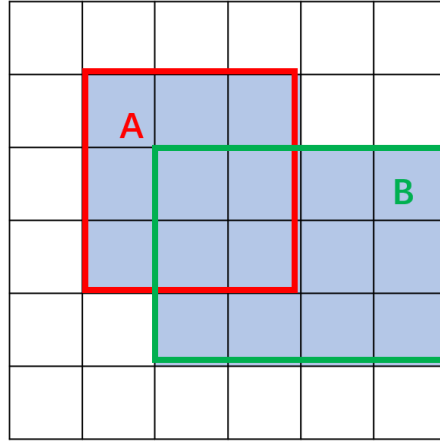


图 2-3 最大区域和最小描述示意图

附注：阴影部分的网格组成一个类， A 和 B 表示这个类的两个最大区域， $A \cup B$ 则为这个类的最小描述。

CLIQUE 算法并非直接对 d 维网格空间进行聚类，而是根据先验原理，从低维子空间开始聚类，然后逐渐过渡到高维空间。先验原理可以表述为：

- 如果一个网格在 d 维空间上是密集网格，则该网格在任意 $d-1$ 维内嵌子空间上一定是密集网格。
- 如果一个网格在任意 $d-1$ 维内嵌子空间上不是密集网格，则该网格在 d 维空间上一定不是密集网格。
- 如果有 m 个网格在 d 维空间上属于同一个类，则它们在任意 $d-1$ 维内嵌子空间上一定属于同一个类。
- 如果有 m 个网格在任意 $d-1$ 维内嵌子空间上不属于同一个类，则它们在 d 维空间上一定不属于同一个类。

根据先验原理，CLIQUE 从一维子空间开始，逐渐过渡到 d 维空间完成聚类。

对于任意 $i-1$ 维内嵌子空间到 i 维空间的过渡过程可描述为：根据若干个 $i-1$ 维内嵌子空间的聚类最小描述，找到它们在 i 维空间上相交的密集网格作为候选网格，用贪心的方法对这些候选网格完成聚类，并生成这些类在 i 维空间上的最小描述。

2.4 聚类质量评价

想要对聚类质量做出精准的评价是困难的。聚类不像分类等有监督学习方法有一个精准的目标，能够通过精确度、召回率等指标来判别其分类的好坏。聚类是一种广泛定义的思想，核心目的是将相似的点聚在同一个类中，不相似的点聚在不同的类中。但对于数据的相似性，不同的聚类模型有着不同的定义，因此难以统一评价。如 k -means 等划分聚类通常有唯一的目标函数，可以通过其目标函数来判别聚类质量的好坏。然而，密度聚类没有明确的目标函数，因此无法通过简单的目标函数进行评价。

目前对于聚类的质量评价主要有两个方向。如果被聚类的数据集包含真实标签，则通过计算聚类标签与真实标签的相似性来评价聚类质量，常用的指标有 Rand Index^[78]、Mutual Information^[79]、V-Measure^[80] 等。如果被聚类的数据集不包含真实标签，则需要根据聚类的思想来判断，通常通过计算同一个类中点的紧密程度与不同的类之间的分离程度来评估聚类质量，常用的指标有 Silhouette Coefficient^[81]、Davies-Bouldin Index^[82]、Calinski-Harabasz Index^[83] 等。

本文采用有标签的数据集进行实验，因此下文详细介绍两个最常用的有标签评价指标。

2.4.1 调兰德指数

兰德指数 (Rand Index, RI) 是一种常用的聚类质量评价指标，可以用来评价数据的真实标签集与聚类结果标签集的相似性。 RI 的取值范围为 $[0, 1]$ ，数值越大表示两个集合的相似度越高。当取值为 1 时表示聚类标签集与真实标签集完全相同，为 0 时则表示两个集合完全不相似。

但是 RI 指标没有考虑到一个随机分配的聚类结果也可能与真实标签集存在一定的相似性，导致随机分配的聚类结果也可能得到较高的 RI 指数。为了消除这种随机性给聚类评价带来的误判，调兰德指数 (Adjusted Rand Index, ARI)^[84] 在 RI 的基础上进行了改进，排除了结果的随机性。

ARI 的取值范围为 $[-1, 1]$ ，取值 -1 时表示两个集合完全不相似，取值为 1 时表示两个集合的聚类标签完全相同，为 0 时则表示聚类结果与随机分配结果无差异。接下来详细介绍 ARI 的计算方法。

设集合一共有 n 个点，令 T 表示数据集的真实标签集， C 表示聚类结果的

标签集。设 a 表示在集合 T 和集合 C 中被分配在同一个类中的点对的数量, b 表示在集合 T 和集合 C 中被分配到不同类中的点对的数量, C_2^n 表示集合中所有点对的可能性, 则 RI 和 ARI 的计算如下所示:

$$RI = \frac{a + b}{C_2^n} \quad (2-3)$$

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]} \quad (2-4)$$

其中 $E(RI)$ 为 RI 的期望, 表示随机分配聚类下 RI 的取值。

2.4.2 调互信息

互信息 (Mutual Information, MI) 是信息论中的一个重要概念, 可以用来衡量两个随机变量之间的相互依赖性, 其值越高说明两个变量的关联性越强。在聚类中, MI 被用来衡量真实标签集与聚类标签集的一致性, 取值范围为 $[0, 1]$, 取值越高代表聚类效果越好。当取值为 1 时表示聚类结果与真实标签集完全一致, 取 0 时则表示完全不一致。

与 ARI 类似, 调互信息 (Adjusted Mutual Information, AMI)^[85] 排除了 MI 的随机性结果, 使评价标准更加精准。 AMI 的取值范围为 $[-1, 1]$, 取值 -1 时表示聚类结果与真实标签集完全不一致, 取值为 1 时表示完全一致, 取 0 时则表示聚类结果与随机分配的结果无差异。下面介绍 AMI 的计算方法。

设数据集的大小为 n , 令 T 和 C 分别表示真实标签集和聚类标签集。将一个集合的信息熵定义为其聚类划分的不确定性, 则 T 和 C 的信息熵分别为:

$$H(T) = - \sum_{i=1}^{|T|} P(i) \log(P(i)) \quad (2-5)$$

$$H(C) = - \sum_{j=1}^{|C|} P'(j) \log(P'(j)) \quad (2-6)$$

其中 $P(i) = \frac{|T_i|}{n}$ 、 $P'(j) = \frac{|C_j|}{n}$ 分别表示随机选取一个点落入集合 T_i 和 C_j 中的概率。

集合 T 和 C 之间的 MI 指标计算如下:

$$MI(T, C) = \sum_{i=1}^{|T|} \sum_{j=1}^{|C|} P(i, j) \cdot \log \left(\frac{P(i, j)}{P(i)P'(j)} \right) \quad (2-7)$$

其中 $P(i, j) = \frac{|T_i \cap C_j|}{n}$ 表示一个点同时被分在类 T_i 和 C_j 中的概率。

结合 MI 的期望值 $E[MI]$ ，则 AMI 计算如下：

$$AMI = \frac{MI - E[MI]}{avg(H(T), H(C)) - E[MI]} \quad (2-8)$$

2.5 本章小结

本章主要介绍了密度聚类相关的基础原理与聚类质量评价方法。首先结合划分聚类、层次聚类、密度聚类三大经典聚类模型介绍了聚类分析的相关概念。然后通过 DBSCAN、OPTICS 两个经典算法详细介绍了密度聚类的理论基础和算法设计思路。并从经典的密度聚类模型引申出了网格聚类模型，结合 GRIDCLUS、CLIQUE 等经典算法详细介绍了网格聚类模型的理论基础和算法设计思想。最后介绍了如何对聚类质量做出合理的评价，并详细介绍了 ARI 和 AMI 两个常用聚类质量评价指标的原理及计算方法。

第3章 一种新的网格聚类模型及 GMCLU 算法

本章研究网格聚类相较于密度聚类带来的聚类质量下降问题,针对传统的网格聚类算法在网格空间划分、网格表示、网格密度定义、网格聚类原则等方面存在的问题展开了讨论,并对网格聚类模型进行了重新的设计,最终使得网格聚类的聚类质量得以提升至密度聚类的水平。基于网格聚类模型的重新设计,本章还提出了一种适用于低维大数据的网格聚类算法 GMCLU (Grids Merging CLUstering)。该算法在低维空间上能够达到 $O(n)$ 的时间复杂度,且仅需要 $O(m)$ 的内存空间,其中 n 为数据集中数据点的数量, $m \ll n$ 为非空网格的数量。

3.1 引言

密度聚类算法(如 DBSCAN)能够在任意形状的数据集上实现良好的聚类效果,并有效地识别噪音数据,因此得到了广泛应用。然而,传统的密度聚类算法时间复杂度较高,难以处理大数据集。网格聚类作为密度聚类的一个主要分支,通过将数据空间网格化后对网格进行聚类,可以有效地降低聚类数据的样本量,达到提高聚类效率的目的。经过近几十年的发展,网格聚类产生了许多优秀的算法和多种不同的解决方案。概括来说,一个典型的网格聚类算法主要包括以下四个步骤^[21-24]:

(1) 根据给定的参数,对每个维度进行均匀划分,将原始数据空间划分为由大量超立方网格组成的网格空间。然后遍历数据集,将每个数据点根据其坐标加入到所属的网格中。

(2) 根据每个网格中的点的数量来计算其网格密度。如果一个网格的密度大于或等于一个给定的阈值,则将其定义为核心网格,否则定义为边缘网格(或称为稠密网格和稀疏网格)。

(3) 从任意一个未被访问过的核心网格开始,将这个网格标记为已访问状态并开启一个新的类,然后将这个核心网格的所有非空邻居网格加入到这个类中。重复的遍历这个类中未被访问过的核心网格,将其标记为已访问状态并将其邻居网格加入该类中,直到处于这个类中的所有核心网格均已被标记为访问状态为止。至此,一个完整的网格类被划分完成。

(4) 重复步骤(3),直到网格空间中的所有核心网格均被标记为已访问状态。至此,整个数据集跟随着网格被划分成了若干个类。对于没有被加入到类中的边缘网格,将其视为噪音网格。

网格聚类通过对数据集进行网格化处理并用单个网格代替其中的数据点进行聚类,从而有效降低了聚类的数据规模,使得聚类效率得到显著的提升。然而,

这一策略也导致了聚类误差的增加,使得网格聚类相较于密度聚类的质量有所下降。本文认为网格聚类质量下降的原因主要有以下两点:

(1) 在定义网格密度时,仅考虑了网格内数据点的数量,而没有考虑该网格周围的其它稠密网格对其产生的影响,这可能导致核心网格被误判为边缘网格。由于算法的最终目的是对数据点进行聚类,网格化过程会导致原本相邻的数据点被人为地分离到不同的网格之中。一个稀疏网格周围可能会充满大量的稠密网格,则这个稀疏网格内的点周围也可能分布着大量的点,但这些点被人为的划分到了其周围的邻近网格之中。从密度聚类的角度来看,这些稀疏网格中的数据点应被视为核心点,但典型的网格聚类会将这些稀疏网格中的点误判为边缘点,这就是因为没有考虑到邻近网格之间的相互影响的原因。

(2) 在对网格进行聚类合并时,典型的网格聚类算法仅考虑了网格之间的距离(或仅考虑邻居网格),而未考虑不同网格中数据点的实际距离,从而容易产生较大的聚类误差。例如,存在两个相邻网格,这两个网格中的数据点都密集地分布在距离对方网格最远的对角位置附近,这意味着这两个网格中的数据点实际距离较远,因此它们不应被分到同一类中。然而,典型的网格聚类算法没有考虑到这一因素,可能会错误地将它们划分为同一类。

为了充分利用网格聚类的速度优势,同时确保其聚类质量,本文需要对网格模型进行重新设计。针对上述问题,本章在第3.2节提出了一种新的网格聚类模型,该模型在网格划分、网格密度、网格距离及网格聚类规则等方面进行了重新定义和改进,并在第3.3节基于该模型提出了网格聚类算法 GMCLU。最终的实验结果表明,本文的网格聚类模型能够有效提升聚类质量,且 GMCLU 算法能够高效地对低维大数据集进行聚类分析。

3.2 网格聚类模型设计

在进行网格聚类模型设计之前,本文首先假设所有数据集均已经过归一化处理。首先,经过归一化后,可以通过设定一个固定的网格宽度轻松地将数据空间划分为由超立方体网格构成的网格空间,而无需考虑数据点在各个维度上取值范围的差异。其次,归一化后的网格空间更便于计算,可以通过简单的数学计算找到一个网格周围的邻居网格。最后,归一化可以使数据分布更均匀,更有利于数据在网格中的均匀分布。

3.2.1 空间的网格划分

空间的网格划分是网格聚类的重要步骤,网格划分的质量将直接影响最终的聚类结果。在典型的网格聚类模型中,通常根据一个给定的维度划分参数来对每个维度进行等距划分,从而实现欧式数据空间的网格化。然而,本文希望网格聚

类输入的关键参数能与 DBSCAN 等密度聚类算法完全匹配，即仅包含 ϵ 和 $minPts$ 两个参数，从而更好地使网格的聚类思想与密度聚类相匹配。因此，有必要对空间网格化的方法进行重新设计。

对于一个给定的 ϵ 参数，通过公式 (3-1) 计算得到网格的宽度 $width$ ，其中 d 为数据的维度。

$$width = \frac{\epsilon}{\sqrt{d}} \quad (3-1)$$

通过公式 (3-1) 的设计，可以确保一个超立方体网格内最远两点之间的距离不超过 ϵ 范围，即保证了网格内所有的点彼此为 ϵ -邻居。

在确定网格宽度之后，将空间的每个维度均等地划分为 l 等份，从而原始的数据空间被划分为由 l^d 个超立方体组成的网格空间。 l 的计算方法如公式 (3-2) 所示，由于数据集经过了归一化处理，因此每个维度的数据取值范围均为 $[0, 1]$ 。

$$l = \left\lceil \frac{1}{width} \right\rceil \quad (3-2)$$

对于一个网格 g ，本文用一个四元组来表示，如公式 (3-3) 所示。其中， pos 为一个 d 维数组，表示网格的坐标； ls 为一个 d 维数组，表示网格内所有点坐标的线性和； iw 为网格内点的数量，表示网格的内在密度； ow 为网格的外在密度，表示该网格的邻近网格对其密度产生的影响。本文将在下一小节对密度的定义进行详细介绍。

$$g = (pos, ls, iw, ow) \quad (3-3)$$

$$ls_{g_i} = sum(x_j), \quad x_j \in g_i \quad (3-4)$$

$$iw_{g_i} = |g_i| \quad (3-5)$$

$$dist(g_i, g_j) = \sqrt{\sum_{k=1}^d \left(\frac{ls_{g_i}^k}{iw_{g_i}} - \frac{ls_{g_j}^k}{iw_{g_j}} \right)^2} \quad (3-6)$$

通过网格的 ls 和 iw 两个属性，可以计算得到网格内点的质心坐标。因此本文定义两个网格之间的距离为其网格内点的质心距离，如公式 (3-6) 所示。

3.2.2 网格密度的定义

如前文所述, 本文在定义网格密度的时候, 不仅考虑了网格内点的数量, 还考虑了周围邻近网格对其密度所产生的影响。本文将这个影响称为网格的外在密度, 用符号 ow 表示。

对于一个网格 g_j , 将其对另一个网格 g_i 的密度影响称为网格 g_i 从网格 g_j 获得的流入密度, 用符号 $ow_{g_i \leftarrow g_j}$ 表示, 其计算方法如公式 (3-7) 所示。

一个网格的外在密度为其 ϵ 范围内的所有邻近网格的流入密度之和, 如公式 (3-8) 所示, 其中 $N(g_i)$ 表示网格 g_i 的所有 ϵ -邻居网格组成的集合。

$$ow_{g_i \leftarrow g_j} = \begin{cases} \min\left(\frac{0.5 \cdot \epsilon}{dist(g_i, g_j)}, 1\right) \cdot g_j, & \text{if } dist(g_i, g_j) \leq \epsilon \\ 0, & \text{otherwise} \end{cases} \quad (3-7)$$

$$ow_{g_i} = \sum_{g_j \in N(g_i)} ow_{g_i \leftarrow g_j} \quad (3-8)$$

最终, 定义一个网格的密度为其内在密度与外在密度之和, 如公式 (3-9) 所示。内在密度为这个网格内点的数量, 反映了网格内点的密集程度, 而外在密度则反映了该网格周围的邻近网格对其密度所产生的影响。

$$den_g = iw_g + ow_g \quad (3-9)$$

3.2.3 网格聚类原则

根据网格划分和网格密度的设计, 能够将原本的欧式空间划分成为网格空间, 并计算出每个的网格密度以及网格之间的距离。按照密度聚类的思想, 本文将这些网格根据其密度划分成为核心网格和边缘网格。

给定一个参数 $minPts$, 本文对核心网格和边缘网格的定义如下 (注: 大部分密度聚类与网格聚类限定 $minPts$ 为整数, 而本文的 $minPts$ 可以设置为小数, 因为本模型的网格密度可以为小数)。

核心网格: 如果一个非空网格的密度 den_g 大于或等于给定的参数 $minPts$, 则将这个网格定义为一个核心网格, 记为 cg 。

边缘网格: 如果一个非空网格的密度 den_g 小于给定的参数 $minPts$, 则将这个网格定义为一个边缘网格, 记为 bg 。

核心网格集: 将全部核心网格组成的集合定义为核心网格集, 记为 G_c 。

边缘网格集：将全部边缘网格组成的集合定义为边缘网格集，记为 G_b 。

得到核心网格集和边缘网格集后，就能对网格进行聚类。对于给定的参数 ϵ 和 $minPts$ ，本文对网格的聚类原则设计如下：

- (1) 一个类中至少应该包含一个核心网格。
- (2) 网格距离小于或等于 ϵ 的核心网格应该被归于同一个类中。
- (3) 核心网格具备传导性质，如果核心网格 cg_1 与核心网格 cg_2 在 ϵ 距离之内，且核心网格 cg_2 与核心网格 cg_3 在 ϵ 距离之内，则 cg_1 与 cg_3 应该被归在同一个类中。
- (4) 边缘网格应该被归于距离它 ϵ 范围内核心网格的类中。如果一个边缘网格的 ϵ 范围内有多个核心网格，则归于其中密度最大的核心网格的类中。
- (5) 如果一个边缘网格的 ϵ 范围内没有核心网格，则将这个边缘网格视为噪音网格。

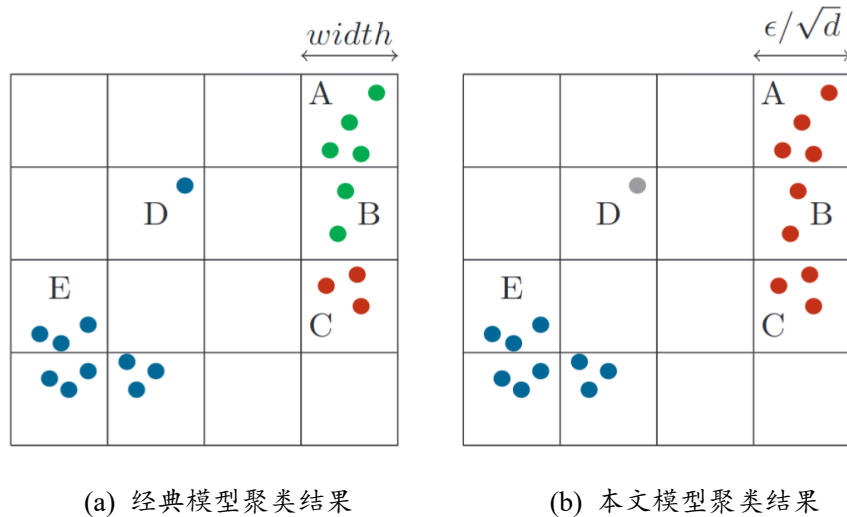


图 3-1 经典模型与本文模型的聚类对比图 ($minPts$ 设置为 3)

至此，本文完成了对网格模型的设计。现在结合图 3-1 详细解释为什么本文的模型能够显著地提高网格聚类质量：

(1) 经典的网格聚类模型通过直接输入网格宽度或每个维度的切分等份来划分网格空间，而本文将网格参数与密度聚类直接对应，通过 ϵ 参数来计算网格宽度，并保证了网格内的任意两点均为彼此的 ϵ -邻居。

(2) 经典模型将网格内点的数量定义为网格密度，容易错误的将核心网格计算为边缘网格。而本文考虑了网格周围的邻近网格对其密度产生的影响，并设计了合理的公式计算出该影响值，得到了新的密度，从而避免了此类误判事件的发生。如图中的网格 B ，当 $minPts$ 设置为 3 的时候，经典模型会将网格 B 视为一个边缘网格，从而将网格 A 、 B 、 C 划分成了 $\{A, B\}$ 和 $\{C\}$ 两个类，而显然这三

个网格被聚在同一个类中更加合理。本文的设计由于考虑了网格 A 和 C 对网格 B 的密度影响,通过计算能够正确地将 B 判断为核心网格,从而正确的将 $\{A, B, C\}$ 聚在了同一个类中。

(3) 经典模型在网格聚类的过程中,简单地将相邻网格合并以完成聚类,以网格距离来代替了点的距离,会导致较大的聚类误差。如边缘网格 D ,经典模型会将其归于其邻居核心网格 E 所在的蓝色类中。而显然 D 网格中的点离 E 网格中的点较远,是一个孤立点,应当被视为噪音点。本文在设计网格时,在网格中保存了点的坐标摘要信息,能够计算出网格内点的质心位置,并用网格内点的质心距离来表示网格距离,能更好地反映不同网格中点的相似性。本模型能够计算出 D 网格与 E 网格中点的质心距离超出了 ϵ 范围,因而更合理地将 D 网格归类为噪音网格。

(4) 在处理边缘网格时,本模型更加细致。经典模型将边缘网格归类于其邻居任一核心网格的类中,而本文将其归于最大密度的 ϵ -邻居网格所在的类中,使边缘网格的聚类更加准确。

3.3 GMCLU 算法实现

本文在 3.2 节中设计了一个新的网格聚类模型,提高了网格的聚类质量。本节将依据该网格聚类模型,实现一种基于网格广度优先搜索的聚类算法,称为 GMCLU (Grids Merging Clustering)。该算法在低维空间下能够达到 $O(n)$ 的时间复杂度和 $O(m)$ 的空间复杂度,其中 m 表示非空网格的数量且远小于点的数量 n 。最后实验证明了 GMCLU 能够高效地处理低维度大数据集。

3.3.1 算法实现

GMCLU 算法基于本文所提出的网格聚类模型,接受两个参数 ϵ 、 $minPts$ 和数据集 D ,具体实现过程如算法 3-1 所示,其主要步骤可以概括如下:

(1) 对数据集 D 进行归一化处理。

(2) 根据公式 (3-1) 和 (3-2) 的计算,将原始数据空间划分为网格空间。然后遍历数据集,将数据点添加到对应的网格中。GMCLU 利用哈希表来表示网格空间,且哈希表中只需要存储包含了数据点的非空网格,其实现过程如算法 3-2 所示。

(3) 遍历全部非空网格,对每个网格,遍历其邻居网格,根据公式 (3-7)、(3-8)、(3-9) 计算网格密度,并根据 $minPts$ 将所有网格分为核心网格集和边缘网格集。其实现过程如算法 3-3 中第 1 到第 10 行所示。

(4) 对核心网格集进行广度优先搜索,根据网格聚类原则,将网格距离小于或等于 ϵ 的核心网格聚在一个类中。此步骤可以将核心网格集聚为若干个类。

其实现过程如算法 3-3 中第 11 到第 23 行所示。

(5) 遍历边缘网格, 将边缘网格加入其 ϵ 范围内密度最大的核心网格所在的类中。没有被聚类的边缘网格被视为噪音网格。其实现过程如算法 3-3 中第 24 到第 29 行所示。

(6) 按照网格的聚类结果为每个数据点打上类的标签, 并输出结果。其实现过程如算法 3-4 所示。

算法 3-1 GMCLU 聚类算法

输入 数据集 $D = \{p_1, \dots, p_n\}$, 半径距离 ϵ , 密度阈值 $minPts$
输出 聚类标签集 $Labels = \{label_1, \dots, label_n\}$
 1 $D' \leftarrow$ 将数据集 D 做归一化处理
 2 $G \leftarrow$ 调用算法 3-2 生成网格空间
 3 调用算法 3-3 对网格聚类, 给 G 中的网格打上类的标签
 4 $Labels \leftarrow$ 调用算法 3-4, 根据网格聚类标签给数据点打上类的标签
Return $Labels$

算法 3-2 网格空间构造

输入 归一化的数据集 D' , 半径距离 ϵ
输出 网格空间 G
 1 将每个维度均等的切分为 $l \leftarrow \left\lceil \frac{\sqrt{d}}{\epsilon} \right\rceil$ 等份, 得到网格空间
 2 $G \leftarrow$ 创建一个空的哈希表来表示网格空间
 3 **foreach** $p \in D$ **do**
 4 $pos \leftarrow \lfloor p \cdot l \rfloor$ // 计算出网格坐标
 5 **if** $pos \in G$ **then** // 将点加入对应网格中
 6 $G[pos].iw \leftarrow G[pos].iw + 1$
 7 $G[pos].ls \leftarrow G[pos].ls + p$
 8 **else** // 创建一个新的网格
 9 $G[pos] \leftarrow Grid(pos = pos, ls = p, iw = 1, ow = 0, label = -1)$
Return G

在计算网格密度的步骤 (3)、聚类核心网格的步骤 (4) 以及聚类边缘网格的步骤 (5) 中, 都需要获取每个网格的 ϵ -邻居网格, 此过程是影响聚类速度的关键部分。为了加速这一过程, 本文在 GMCLU 的算法实现中将网格坐标整数化, 即数据点 p 所属的网格坐标通过公式 (3-10) 计算, 其中 l 表示原始欧式空间的每个维度被切分成 l 等份。在此设计下, 寻找一个网格的所有邻居网格只需要在各个维度上进行加减 1 计算即可完成, 一个网格的邻居网格共有 $3^d - 1$ 个。

如一个二维网格的坐标是 $[1,1]$ ，则其所有的邻居网格集为 $\{[0,1], [0,-1], [1,0], [1,-1], [-1,-1], [-1,1], [1,1], [1,-1]\}$ 。尽管这种设计无法确保找到所有的 ϵ -邻居网格，但它在牺牲了少量聚类精度的同时，极大地提高了聚类效率。

$$g_{pos} = \lfloor p \cdot l \rfloor \quad (3-10)$$

算法 3-3 网格聚类

输入 网格空间 G ，半径距离 ϵ ，密度阈值 $minPts$

输出 无显示输出；根据聚类结果对 G 中的网格打上类的标签

```

1 // 1. 计算网格的外部密度，并将网格空间  $G$  划分为核心网格集和边缘网格集
2  $C, B \leftarrow$  新建空的核心网格集和边缘网格集
3 foreach  $g \in G$  do
4     foreach  $g'$  of  $g$  的邻居网格 do
5         if  $g' \in G$  and  $dist(g, g') \leq \epsilon$  do
6              $g.ow = g.ow + \min\left(\frac{0.5 \cdot \epsilon}{dist(g, g')}, 1\right) \cdot g'$ 
7         if  $g.iw + g.ow \geq minPts$  do
8             将网格  $g$  加入到核心网格集  $C$  中
9         else
10             将网格  $g$  加入到边缘网格集  $B$  中
11 // 2. 对核心网格进行聚类
12  $Q \leftarrow$  新建一个空队列
13  $labelId \leftarrow 0$  // 初始化类标签
14 foreach  $c \in C$  and  $c$  未访问 do
15     标记核心网格  $c$  为已访问状态且将  $c$  加入队列  $Q$ 
16      $c.label \leftarrow labelId$ 
17     while  $Q$  不为空时 do
18          $c \leftarrow$  从  $Q$  中取出队首网格
19         foreach  $g$  of  $c$  的非空邻居网格 and  $g$  未访问 do
20             if  $g$  是核心网格 and  $dist(c, g) \leq \epsilon$  do
21                 标记  $g$  为已访问且把  $g$  加入  $Q$  的队尾
22                  $g.label \leftarrow labelId$ 
23              $labelId = labelId + 1$ 
24 // 3. 对边缘网格进行聚类
25 foreach  $b \in B$  do
26     if  $b$  的邻居网格中包含  $dist(b, c) \leq \epsilon$  的核心网格  $c$  do
27          $b.label \leftarrow$  将符合条件且密度最大的核心网格的标签赋予  $b$ 
28     else
29          $b$  的初始标签为-1，代表噪音网格

```

算法 3-4 生成聚类标签

输入 归一化的数据集 D' , 已聚类的网格 G

输出 聚类标签集 $Labels$

```

1   $Labels = []$  // 初始化一个空的标签集
2  foreach  $p \in D'$  do // 找到点  $p$  所在的网格, 并将该网格的类标签赋予  $p$ 
3       $pos \leftarrow \lfloor p \cdot l \rfloor$ 
4       $label \leftarrow G[pos].label$ 
5       $Labels.Append(label)$ 
Return  $Labels$ 

```

3.3.2 算法复杂度分析

为了阐述 GMCLU 算法在处理低维度大数据集时的有效性, 本节将对算法的时间复杂度和空间复杂度进行详细分析。最终证明了在低维空间下, GMCLU 能够达到 $O(n)$ 的时间复杂度和 $O(m)$ 的空间复杂度, 其中 n 表示数据集点的数量, $m \ll n$ 表示非空网格的数量。

时间复杂度分析: GMCLU 算法在 $O(1)$ 的时间内计算出网格宽度, 然后遍历数据集将各点加入其所属的哈希网格表中, 对每个点只需要 $O(1)$ 的时间进行处理, 因此建立网格空间总共需要 $O(n)$ 的时间。接下来, 算法需要计算每个网格的密度并将其划分为核心网格集和边缘网格集。为了计算网格的外部密度, 需要访问其所有邻居网格。一个网格周围可能有 $3^d - 1$ 个邻居网格, 对每个邻居网格只需要 $O(1)$ 的时间进行判断和计算, 则全部网格的密度计算需要 $O(m \cdot 3^d)$ 的时间。然后, 对核心网格和边缘网格进行聚类, 该过程采用广度优先搜索合并的方式完成。每个非空网格只需要访问一次, 同样每次访问需要遍历它的 $3^d - 1$ 个邻居网格, 因此网格的聚类时间也需要 $O(m \cdot 3^d)$ 。最后, 再遍历一次数据集对给每个点打上标签, 需要 $O(n)$ 的时间。综上所述, 整个算法的时间复杂度为 $O(n + m \cdot 3^d)$ 。在低维空间下, 数据集能够被良好的网格化, 非空网格的数量 m 远远小于数据点的数量 n , 且 3^d 是一个小常数, 因此 $m \cdot 3^d$ 小于 n 或至少与 n 在一个量级, 则得到 GMCLU 在低维空间下的时间复杂度为 $O(n)$ 。

空间复杂度分析: 网格模型是对网格进行聚类, 因此只需要从硬盘中顺序读取数据存入网格中, 而不需要花费 $O(n)$ 的内存空间对原始数据集进行存储。一个网格由四元组 (pos, ls, iw, ow) 表示, 其中 pos 和 ls 是一个 d 维的数组, 而 iw 和 ow 是一个常数, 因此一个网格需要 $O(d)$ 的存储空间。GMCLU 一共需要存储 m 个非空网格, 因而总共需要 $O(m \cdot d)$ 的空间。由于对数据维度 d 的处理是不可避免的, 因而本文在讨论时间复杂度和空间复杂度时隐藏了维度的信息, 则 GMCLU 的空间复杂度为 $O(m)$ 。如前文所述, 当维度很低时, m 远远小于 n , 因此 GMCLU 在低维大数据集下可以节省大量的内存空间。

3.4 实验分析

为了评估 GMCLU 算法的性能, 本文从聚类质量和聚类效率两个维度对 GMCLU 算法进行实验分析。实验选取了 DBSCAN、OPTICS 两个密度聚类算法以及 BANG、CLIQUE 两个网格聚类算法作为 GMCLU 的对比算法。这四个算法是被广泛应用的、具有代表性的经典算法, 在大量文献中被作为对比算法出现。同时, 这四个算法已被大多数聚类的机器学习库所实现, 表明了它们在实践领域中的应用价值。相关对比算法的详细信息在本文第 2 章中已有过介绍。

3.4.1 实验环境

本章的所有实验都是在 ThinkBook 14p G2 笔记本电脑上进行的: 操作系统为 64 位的 Ubuntu 22.04.2 LTS 桌面版虚拟机; 处理器型号为 AMD Ryzen 7 5800H CPU@3.20 GHz, 16 核; 内存大小为 16GB; 算法运行环境为 Python 3.10 解释器。

为了公平的对比相关算法, 本实验的所有算法均由 Python 实现。本文的 GMCLU 算法通过 Python 3.10 编码实现。对比算法 DBSCAN、OPTICS、BANG 和 CLIQUE 均由 Python 的聚类算法库 PyClustering^[86] 实现, 版本为 0.10.1, 并关闭了该库 C 语言加速功能。

3.4.2 实验数据集

为了衡量各算法在聚类质量方面的表现, 本文选择了 13 个广泛用于聚类质量测试的基准数据集进行实验, 如表 3-1 中编号 1 至 13 的数据集所示。数据集 1 至 6 涵盖了不同分布形态的数据, 其中的类呈现出多种不同的形状。数据集 7 至 13 中的类则遵循高斯分布。各数据集详情如下: Compound 数据集来源于文献[87], 包含 399 个二维数据点, 共 6 个类; D31 和 R15 数据集均源自文献[88], D31 包含 3100 个二维数据点, 共 31 个类, R15 包含 600 个二维数据点, 共 15 个类; Flame 数据集来自文献[89], 包含 240 个二维数据点, 共 2 个类; Iris 数据集来自文献[90], 包含 150 个四维数据点, 共 3 个类; Pathbased 数据集来自文献[91], 包含 312 个二维数据点, 共 3 个类; 数据集 7 至 9 均来自文献[92], 包含 2048 个数据点, 分别为二维、四维和八维, 每个数据集包含 2 个类; 数据集 11 至 13 均来自文献[93], 每个数据集包含 5000 个二维数据点和 15 个类, 它们的主要区别在于各数据集中高斯分布的标准差不同。

为了评估各算法在低维度大规模数据集上的聚类性能, 本文选取了三个编号为 14 至 16 的低维大数据集来测试聚类算法的运行时间。这三个数据集均由 Scikit-learn 库生成, 其中的类均符合高斯分布, 其高斯分布的标准差均为 0.6。

具体而言,数据集 G10-3d-100k 由 10 万个三维数据点组成,共包含 10 个类;数据集 G20-3d-50k 包含 5 万个三维数据点,分为 20 个类;数据集 G10-4d-10m 则由一千万个四维数据点构成,包含了 10 个类。

表 3-1 测试数据集

编号	数据集	维度	类的数量	点的数量
1	Compound	2	6	399
2	D31	2	31	3100
3	Flame	2	2	240
4	Iris	4	3	150
5	Pathbased	2	3	312
6	R15	2	15	600
7	G2-2-30	2	2	2048
8	G2-4-30	4	2	2048
9	G2-8-30	8	2	2048
10	S1	2	15	5000
11	S2	2	15	5000
12	S3	2	15	5000
13	S4	2	15	5000
14	G10-3d-100k	3	10	100000
15	G20-3d-50k	3	20	50000
16	G10-4d-10m	4	10	10000000

3.4.3 聚类质量分析

本文采用了编号 1 至 13 的十三个基准数据集,以对比评估各算法的聚类质量。本实验选择了调兰德指数 (ARI) 和调互信息 (AMI) 作为衡量聚类质量的指标。这两个度量标准可以量化数据的真实标签集与聚类标签集之间的相似程度,并已在众多的聚类质量对比中得到广泛应用。 ARI 和 AMI 的值域在 $[-1,1]$ 之间,值越大意味着聚类质量越高。当指数达到 1 时,表明聚类结果与真实结果完全匹配。有关 ARI 和 AMI 的更多详细信息,请参见本文第 2.4 节。

为了获得各算法在各数据集上的最优聚类表现,本实验在每个数据集上对算法参数进行了大量迭代。最终选取在 ARI 达到最大值时的参数设置,并基于此参数计算相应的 AMI 值。各个算法在各数据集上的最佳聚类结果如表 3-2 所示。

从结果来看,DBSCAN 的 ARI 和 AMI 指标均在 8 个数据集上取得了最佳表现,两个指标在 13 个数据集下的平均值分别为 0.84 和 0.85。OPTICS 的 ARI 和 AMI 指标均在 5 个数据集上取得领先,其在 13 个数据集上的平均值分别为 0.84

和 0.85。BANG 算法只在 G2-8-30 一个数据集上取得了两个指标的领先，在 13 个数据集上 *ARI* 和 *AMI* 的平均值均为 0.75。CLIQUE 算法只在 Iris 一个数据集上取得了 *ARI* 和 *AMI* 的领先，其平均值分别为 0.75 和 0.79。本文的算法 GMCLU 在 *ARI* 上取得了 7 个数据集的领先，平均值为 0.85，在 *AMI* 上取得了 8 个数据集的领先，平均值为 0.87。

表 3-2 聚类质量对比

评价指标	<i>ARI</i> <i>AMI</i>				
测试数据集	DBSCAN	OPTICS	BANG	CLIQUE	GMCLU
Compound	0.93 0.89	0.93 0.89	0.91 0.74	0.92 0.79	0.96 0.93
D31	0.84 0.91	0.80 0.90	0.66 0.80	0.64 0.86	0.83 0.90
Flame	0.97 0.93	0.96 0.92	0.92 0.84	0.87 0.73	0.94 0.88
Iris	0.94 0.89	0.94 0.89	0.99 0.97	1.00 1.00	0.92 0.87
Pathbased	0.92 0.89	0.95 0.92	0.50 0.59	0.67 0.67	0.96 0.93
R15	0.99 0.99	0.98 0.98	0.90 0.95	0.82 0.89	0.97 0.98
G2-2-30	0.91 0.84	0.91 0.84	0.68 0.51	0.86 0.70	0.91 0.84
G2-4-30	0.99 0.98	0.99 0.98	0.85 0.69	0.97 0.94	0.99 0.98
G2-8-30	0.85 0.79	0.85 0.79	1.00 1.00	0.98 0.95	1.00 1.00
S1	0.97 0.97	0.97 0.97	0.88 0.88	0.89 0.92	0.97 0.97
S2	0.75 0.88	0.75 0.88	0.67 0.77	0.63 0.80	0.74 0.85
S3	0.42 0.61	0.42 0.61	0.37 0.50	0.29 0.57	0.46 0.65
S4	0.43 0.54	0.43 0.54	0.41 0.50	0.19 0.47	0.42 0.54
平均值	0.84 0.85	0.84 0.85	0.75 0.75	0.75 0.79	0.85 0.87

通过平均值分析，DBSCAN 和 OPTICS 两个密度聚类算法的聚类质量近乎相同，都取得了 0.84 的 *ARI* 值和 0.85 的 *AMI* 值，它们的聚类结果都远远的好于 BANG 和 CLIQUE 两个网格聚类，其中 BANG 的取值为 0.75 和 0.75，CLIQUE 的取值为 0.75 和 0.79。本文的 GMCLU 算法在平均结果上取得了全面的领先，*ARI* 得到了 0.85，比 DBSCAN 和 OPTICS 两个密度聚类提高了 1.2%，比 BANG 和 CLIQUE 两个网格聚类提高了 13.3%；*AMI* 得到了 0.87，比 DBSCAN 和 OPTICS 提高了 2.4%，比 BANG 提高了 16.0%，比 CLIQUE 提高了 10.1%。

根据对比可以得出，密度聚类算法普遍比网格聚类算法能够得到更好的聚类质量，而本文的 GMCLU 算法则能够显著地提高网格聚类的聚类质量，甚至在某些数据集上取得了比密度聚类更好的结果。值得一提的是，如 3.3 节所述，GMCLU 算法为了追求 $O(n)$ 时间的聚类速度，在实现上并没有完全履行 3.2 节的网格模型设计，造成了一定程度的聚类质量损失。而本文的网格模型实际上能够取得比 GMCLU 算法更好聚类结果，这一点将在第 4 章的聚类质量实验中得

以验证。

3.4.4 聚类效率分析

本实验从两个方面对各算法的聚类效率进行评估：首先，排除参数选择的影响，以聚类质量作为评判标准，测试各算法的 ARI 在数据集上达到某一阈值时的最短聚类时间。其次，通过设置不同参数，评估各算法在不同参数情况下的聚类时间。

(1) 以 ARI 为质量基准进行各算法的聚类效率对比分析

在第一个评估方法中，本文以调兰德指数 (ARI) 为聚类质量基准，选用数据集 G10-3d-100k 进行实验。通过对各算法参数的大量迭代测试，最终选择了在 ARI 分别达到 0.6、0.7、0.8、0.9 时各算法的最短聚类时间作为评价标准。实验结果如图 3-2 所示。

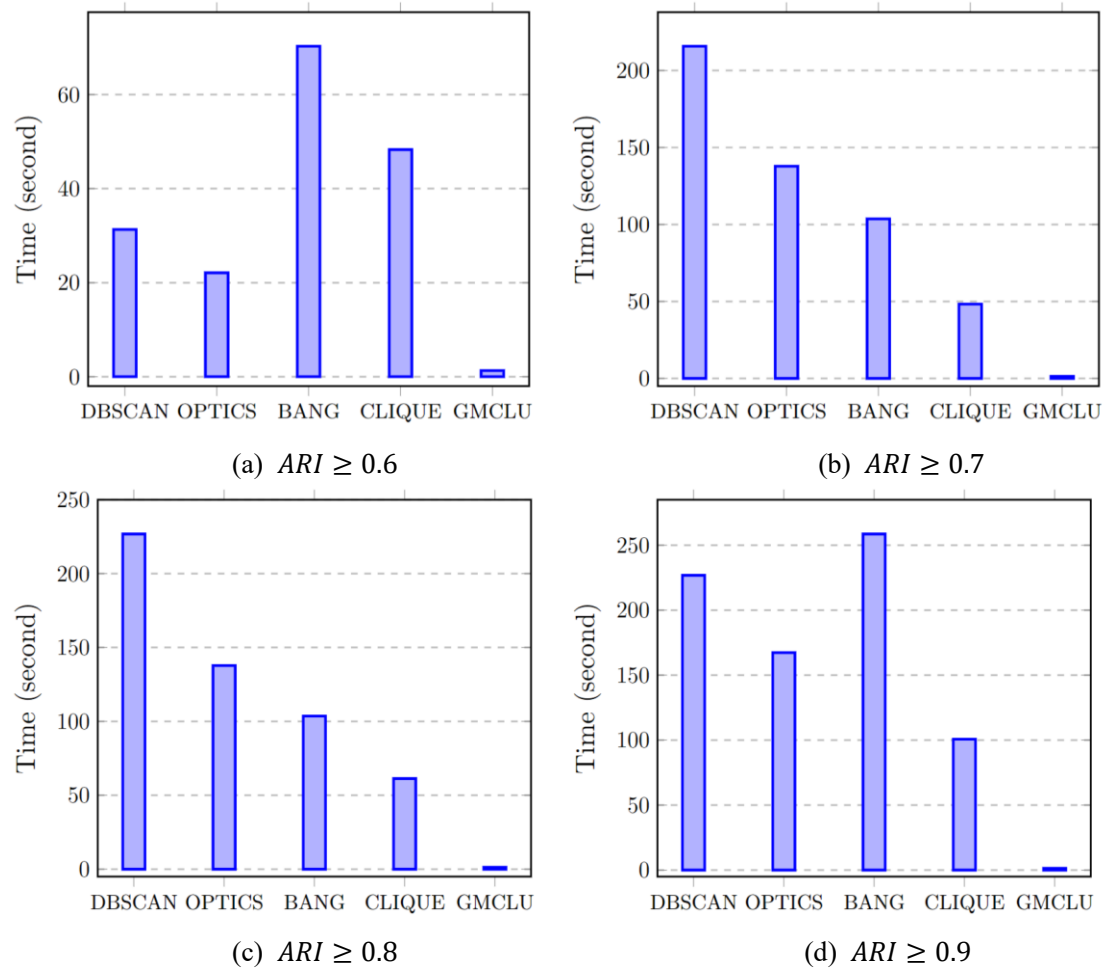


图 3-2 当 ARI 分别到达 0.6、0.7、0.8、0.9 时，各算法在 G10-3d-100k 数据集上的最快运行时间。

从结果中可以观察到,无论 ARI 阈值如何选择,GMCLU 算法在 G10-3d-100k 数据集上都能稳定且迅速地得到聚类结果,其所需时间最大不超过 1.4 秒。在 ARI 阈值为 0.6 时,GMCLU 比 DBSCAN 快了 24 倍,比 OPTICS 快了 17 倍,比 BANG 快了 54 倍,比 CLIQUE 快了 37 倍。在 ARI 阈值为 0.7 时,GMCLU 比 DBSCAN 快了 166 倍,比 OPTICS 快了 106 倍,比 BANG 快了 80 倍,比 CLIQUE 快了 37 倍。在 ARI 阈值为 0.8 时,GMCLU 比 DBSCAN 快了 174 倍,比 OPTICS 快了 106 倍,比 BANG 快了 80 倍,比 CLIQUE 快了 47 倍。在 ARI 阈值为 0.9 时,GMCLU 比 DBSCAN 快了 162 倍,比 OPTICS 快了 119 倍,比 BANG 快了 185 倍,比 CLIQUE 快了 72 倍。

在其他四个对算法中,CLIQUE 的表现相对稳定。当 ARI 值为 0.7、0.8 和 0.9 时,其性能均优于其他三个算法。DBSCAN 和 OPTICS 在 ARI 阈值较低时(取值 0.6)表现优于 BANG 和 CLIQUE,但随着 ARI 阈值的提高,其表现逐渐下降。BANG 在 ARI 阈值为 0.7 和 0.8 时,聚类速度优于 DBSCAN 和 OPTICS,但在 ARI 阈值为 0.6 和 0.9 时表现最差。

以 ARI 为基准的聚类效率评估只能作为评价标准之一。尽管它排除了参数选择的影响,但仍受到数据集中数据分布的影响,因为数据分布会影响算法参数的选取以及相应的 ARI 值。因此,本文还将在不同参数设置下对算法的聚类效率进行进一步的评估。

(2) 在多组参数下进行 GMCLU 与密度聚类的聚类效率对比分析

为了评估 CMCLU 与密度聚类算法的聚类效率,本实验选取了样本量为 5 万的 G20-3d-50k 数据集,分别测试了 GMCLU、DBSCAN、OPTICS 三个算法在多组参数下的聚类时间。其中 ϵ 参数取值范围为[0.02, 0.2],步长为 0.02; $minPts$ 参数分别选取为 5、50 和 100。实验结果如图 3-3 所示。

从结果中可以观察到,DBSCAN 的聚类时间受 ϵ 参数取值的显著影响,聚类时间随着 ϵ 参数的增大而急剧上升,很快超出了 4000 秒的时间限制。同时,DBSCAN 的聚类时间还受到 $minPts$ 参数的较大影响。当 $minPts$ 取值为 5 时,DBSCAN 在 ϵ 为 0.06 时就超出了时间限制,而当 $minPts$ 取 50 和 100 时,其聚类时间在 ϵ 为 0.1 时才超出时间限制。这是因为 DBSCAN 的聚类时间主要受到核心点数量的影响,且 ϵ 取值越大, $minPts$ 取值越小,核心点数量越多。OPTICS 与 DBSCAN 类似,其聚类时间随着 ϵ 的增长而上升。但是 OPTICS 相较于 DBSCAN 具有更高的聚类效率,受参数影响的波动也较小。当 ϵ 取值为 0.02 时,OPTICS 在 $minPts$ 为 5、50、100 时的聚类时间分别为 215.7 秒、116.2 秒、103.1 秒。当 ϵ 取值为 0.18 时,OPTICS 在 $minPts$ 为 5、50、100 时的聚类时间分别为 3364 秒、3554.0 秒、3289.4 秒。相较之下,本文提出的 GMCLU 算法时间波动

较稳定,且随着 ϵ 的增长呈下降趋势。当 ϵ 取值为 0.02 时 GMCLU 的聚类时间取得最大,在 $minPts$ 的取值为 5、50、100 时分别为 10.5 秒、13.9 秒、12.7 秒,比 DBSCAN 分别快了 32.0 倍、12.7 倍、14.2 倍,比 OPTICS 分别快了 20.5 倍、8.4 倍、8.1 倍。GMCLU 聚类效率的优势随着 ϵ 的增大而越发显著,当 ϵ 取值为 0.18 时,在 $minPts$ 的取值为 5、50、100 的聚类时间分别为 3.1 秒、3.2 秒、4.0 秒,比 OPTICS 分别快了 1085.2 倍、1110.6 倍、822.4 倍。当 ϵ 取值为 0.2 时, DBSCAN 和 OPTICS 都超出了时间限时,而 GMCLU 的聚类时间仅为 3.1 秒、3.0 秒、4.0 秒。

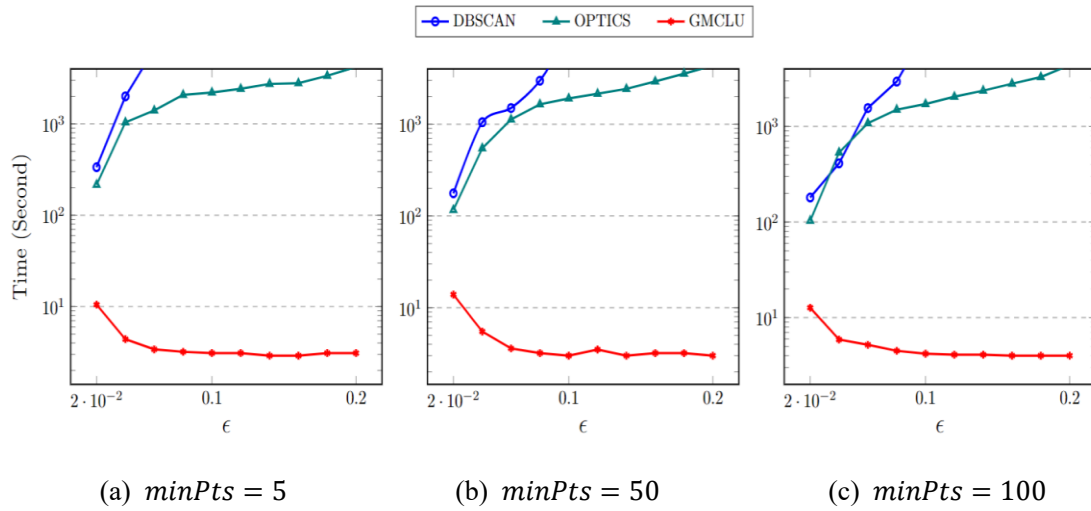


图 3-3 GMCLU、DBSCAN、OPTICS 在 G20-3d-50k 数据集上的聚类时间对比

(3) 在多组参数下进行 GMCLU 与网络聚类的聚类效率对比分析

CLIQUE 算法通过一个输入的参数 l 对每个维度进行等量划分来将原始的数据空间划分为网格空间。为了匹配 CLIQUE 的参数输入,本文在 GMCLU 的实验中同样使用参数 l 来生成网格空间,并通过 $\epsilon = \sqrt{d}/l$ 来计算出半径阈值。本实验在样本量为 10 万的数据集上对 CLIQUE 和 GMCLU 进行了聚类效率测试。其中 l 参数设置范围为[5, 50],步长为 5; $minPts$ 分别选取 5、50 和 100。最终的实验结果如图 3-4 所示。

结果表明,CLIQUE 和 GMCLU 的聚类时间都主要受到维度划分参数 l 的影响。CLIQUE 的聚类时间随着 l 的增大急剧增长,而 GMCLU 的聚类时间增长相对缓慢。在 l 取值为 5 时,CLIQUE 算法在 $minPts$ 为 5、50、100 时的聚类时间分别为 7.9 秒、9.2 秒、7.0 秒。而 GMCLU 在对应参数下的聚类时间为 1.2 秒、1.2 秒、1.3 秒,比 CLIQUE 分别快了 6.5 倍、7.6 倍、5.4 倍。当 l 取值为 40 时,CLIQUE 在对应 $minPts$ 下的聚类时间分别为 3133.5 秒、3136.5 秒、3102.0 秒,

而 GMCLU 在对应参数下的聚类时间为 1.4 秒、1.6 秒、1.6 秒，比 CLIQUE 分别快了 2238.2 倍、1960.3 倍、1938.8 倍。当 l 取值大于 40 后，CLIQUE 的聚类时间超出了 4000 秒的时间限制，而 GMCLU 在 l 取值为 50 时，聚类时间仍然不超过 2 秒。

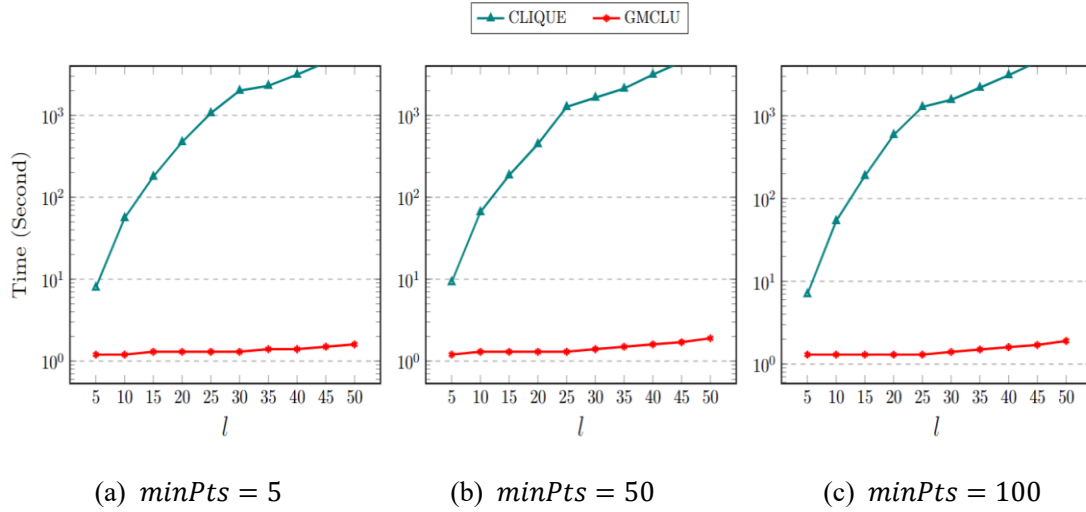


图 3-4 GMCLU 与 CLIQUE 在 G10-3d-100k 数据集上的聚类时间对比

BANG 算法的网格生成方式与 CLIQUE 与 GMCLU 不同，不能直接进行比较。BANG 通过一棵树状结构对空间进行网格划分，每次将一个较大的空间划分为两个子空间，最终在树的 $level$ 层停止划分，最终将原本的数据空间划分为网格数量为叶子个数的网格空间。为了将 GMCLU 与 BANG 在相似参数取值下进行聚类时间比较，本实验通过计算 BANG 和 GMCLU 分别在参数 $level$ 和 l 下的最大网格数量，并根据其最大网格数量将 $level$ 与 l 参数对应起来。根据此设计，最终拟定了在三维空间下，当 GMCLU 的 l 参数取值为 5 到 50 时，BANG 的 $level$ 参数取值为 8 到 18。同时，BANG 的另外一个 $threshold$ 表示当网格中点的数量小于或等于该数值时，将网格定义为噪音网格并不再进行划分。因此，本实验将 BANG 的 $threshold$ 参数设置为 GMCLU 的 $minPts - 1$ 。根据这些参数设定，本实验在样本量为 10 万的 G10-3d-100k 数据集上对 GMCLU 和 BANG 进行了聚类效率测试，其结果如表 3-3 所示。

从表中可以发现，BANG 的聚类时间受 $level$ 和 $threshold$ 的影响都较大，随着 $level$ 的增大而增大，随着 $threshold$ 的增大而减小。在 $level$ 取值为 8 时，BANG 在 $threshold$ 为 0、4、49 的聚类时间分别为 11.6 秒、9.7 秒、9.6 秒。相应的，GMCLU 在 l 取值为 5 时，对应 $minPts$ 为 1、5、50 的聚类时间都为 1.2 秒，比 BANG 分别快了 9.7 倍、8.1 倍、8.0 倍。随着网格数量的增加，GMCLU 聚类效率的优势进一步扩大。当 $level$ 取值为 18，对应的 l 取值为 50 时，BANG

在三个 *threshold* 下的聚类时间分别为 1064.2 秒、693.8 秒、347.8 秒，而 GMCLU 的聚类时间为 1.3 秒、1.6 秒、1.9 秒，比 BANG 分别快了 818.6 倍、433.6 倍、183.1 倍。

表 3-3 GMCLU 与 BANG 在 G10-3d-100k 数据集上的聚类时间对比（时间：秒）

参数: <i>l</i>	5	10	15	20	25	30	35	40	45	50
参数: <i>level</i>	8	11	13	14	15	16	17	17	18	18
GMCLU: <i>minPts</i> =1	1.2	1.2	1.2	1.2	1.2	1.3	1.3	1.3	1.3	1.3
BANG: <i>threshold</i> =0	11.6	36.3	113.9	166.8	274.9	432.8	689.3	689.3	1064.2	1064.2
GMCLU: <i>minPts</i> =5	1.2	1.2	1.3	1.3	1.3	1.3	1.4	1.4	1.5	1.6
BANG: <i>threshold</i> =4	9.7	31.2	69.4	124.4	171.4	262.3	435.3	435.3	693.8	693.8
GMCLU: <i>minPts</i> =50	1.2	1.3	1.3	1.3	1.3	1.4	1.5	1.6	1.7	1.9
BANG: <i>threshold</i> =49	9.6	25.3	58.0	87.0	117.4	141.6	259.2	259.2	347.8	347.8

（4）在样本量为千万的大数据集下测试 GMCLU 的聚类效率

为了更好地测试 GMCLU 在低维大数据集上的聚类效率，本文在样本量为 1 千万、维度为 4 的 G10-4d-10m 数据集上进行了实验。为了更直观的展示 GMCLU 的算法效率受网格数量的影响，本实验选择对每个维度的等量切分 *l* 与密度阈值 *minPts* 作为参数设定来进行实验，并根据 $\epsilon = \sqrt{d} / l$ 来计算出半径阈值。其中 *l* 的取值范围从 5 到 50，*minPts* 分别取值为 5、50 和 100。由于其他四个算法无法在限定时间内完成该大数据集的聚类，因此本文只展示 GMCLU 的实验结果，其结果如图 3-5 所示。

如结果所示，在千万级的 4 维大数据集下，GMCLU 在各个参数下均能迅速对其完成聚类。当 *minPts* 取 5 时，GMCLU 在 *l* 为 5 时只需要 117.4 秒即可完成聚类，在 *l* 为 50 时也仅需要 133.3 秒。当 *minPts* 取 50 时，GMCLU 在 *l* 为 5 时的聚类时间是 117.5 秒，在 *l* 为 50 时为 143.5 秒。当 *minPts* 取 100 时，GMCLU 在 *l* 为 5 时的聚类时间是 117.6 秒，在 *l* 为 50 时为 146.7 秒。可见，*minPts* 参数的改变对 GMCLU 算法的运行效率影响不大，这与 GMCLU 的算法设计相吻合。尽管曲线表明 GMCLU 的聚类时间会随着 *l* 的增长而增长（*l* 的增长代表网格数量的增长），但实际上的绝对时间增量并不大。如当 *minPts* 为 100 时，

GMCLU 在 l 为 50 时对比 l 为 5 的聚类时间只增长了 29.1 秒，而它们之间网格空间的大小相差了 $50^4 - 5^4 = 6249375$ ，这表明了 GMCLU 算法的聚类效率非常稳定，对参数设定并不敏感。

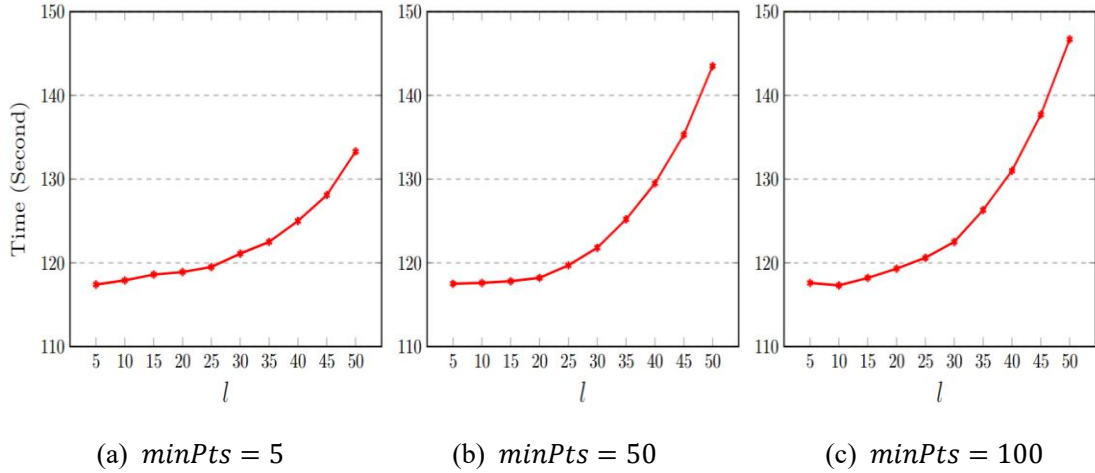


图 3-5 GMCLU 在 G10-4d-10m 数据集上的聚类时间

综上所述，本文从聚类质量和聚类效率两个维度对 DBSCAN、OPTICS、CLIQUE、BANG 和 GMCLU 五个算法进行了系统的对比实验。实验结果首先证明了本文的 GMCLU 算法在聚类质量上远优于 CLIQUE 和 BANG 两个网格聚类算法，在 13 个基准数据集下，其平均的 ARI 指标比 CLIQUE 和 BANG 提高了 13.3%，而 AMI 指标则分别提高了 10.1% 和 16.0%。对于 ARI 和 AMI 这两个指标而言，这样的提升是巨大的。实验显示 GMCLU 的聚类质量甚至比 DBSCAN 和 OPTICS 两个密度聚类算法也有微小的提升，在 ARI 和 AMI 两个指标上分别提高了 1.2% 和 2.4%。其次，实验在 G10-3d-100k、G20-3d-50k、G10-4d-10m 三个低维大数据集上进行了聚类效率的测试。实验证明了 DBSCAN 和 OPTICS 两个密度聚类的聚类效率最低，且其聚类时间受到参数的极大影响。BANG 和 CLIQUE 两个网格聚类算法虽然在效率上有所提升，但依旧难以满足大数据集的聚类需求。相较之下，GMCLU 算法在低维大数据集上的聚类效率远远超过了其他几个对比算法。随着半径阈值、网格数量、数据集样本量的增大，GMCLU 的聚类效率优势急剧增大，甚至在有限的测试范围内达到了几千上万倍的提升。同时，实验证明了 GMCLU 在低维大数据集上的聚类效率非常稳定，受参数选择的影响较小，因此非常适用于低维大数据集的聚类分析。

3.5 本章小结

本章提出了一种新的网格聚类模型，通过对网格划分、网格密度、网格距离和网格聚类原则等方面的重新设计，显著提升了网格模型的聚类质量。基于此模

型，本文提出了 GMCLU 算法，该算法在低维数据集上能够达到 $O(n)$ 的时间复杂度。最后，本文通过聚类质量和聚类效率两个维度的实验证明了 GMCLU 算法在显著提高聚类质量的同时，能够迅速完成低维大数据集的聚类分析。

第 4 章 基于树的空间划分和子聚类合并的网格聚类算法

本章主要研究密度聚类与网格聚类在高维空间下的聚类效率问题。在前一章中, 本文设计了一个新的网格聚类模型, 提升了网格聚类的质量, 并提出了一个适用于低维大数据集的聚类算法 GMCLU。但 GMCLU 的算法时间复杂度包含了 3^d , 因而无法应用于高维度大数据集。为了解决这一问题, 本章提出了一种基于树的空间划分和子聚类合并的网格聚类框架 GTCLU (Grid-partition Tree based CLUstering), 能够有效提高网格模型在高维空间下的聚类效率。

4.1 引言

密度聚类的时间花销主要来源于对每个点的 ϵ -邻居的查找过程, 在计算点的密度时, 以及进行密度聚类的过程中, 都需要花费 $O(n)$ 的时间来查找每个点的 ϵ -邻居, 因此典型的密度聚类算法 (如 DBSCAN) 需要 $O(n^2)$ 的时间才能完成聚类。虽然通过 kd-tree^[94]、ball-tree^[95] 等索引结构能够在一定程度上对密度聚类进行加速, 将寻找 ϵ -邻居的平均时间降低到 $O(\log n)$, 进而将整体的期望时间提升到 $O(n \cdot \log n)$ 。但该方案需要花费额外的 $O(n \cdot \log n)$ 时间和 $O(n)$ 空间来创建索引结构, 且其查询 ϵ -邻居的最坏时间依然是 $O(n)$ 。因此带有 kd-tree 等索引结构加速的密度聚类算法也难以被应用于大数据集的聚类分析。

网格聚类通过将原始的数据空间进行网格划分, 可以将聚类数据的规模由数据点的数量 n 降低到非空网格的数量 m , 进而加速聚类过程。特别是在低维空间下, 网格模型可以通过网格坐标的计算来寻找邻居网格, 而无需遍历整个网格集, 这使得网格的聚类速度得到进一步提升。但当处理高维数据时, 由于整个网格空间的大小与维度成 l^d 的指数关系 (其中 l 表示每个维度被切分成多少等份), 因此高维空间下的数据点更容易被分配到不同的网格当中。维度的增加直接带来了非空网格数量的急剧增长, 导致网格的聚类效率急剧下降。此外, 由于每个网格周围的邻居网格数量为 $3^d - 1$, 因此在高维空间下无法通过直接计算其邻居网格的方式寻找非空邻居网格, 而只能遍历整个非空网格集, 这进一步导致 GMCLU 等低维空间下的高效聚类算法无法被应用于高维空间。

为了解决上述问题, 本章基于第 3 章改进的网格模型, 采用分而治之的思想, 提出了一种基于树的空间划分和子聚类合并的网格聚类框架 GTCLU, 能有效地提高网格聚类在高维大数据集下的聚类效率, 且不会降低聚类质量。本章的主要贡献如下:

(1) 提出了一种易于实现的网格聚类框架 GTCLU, 该算法在保证聚类质量的同时, 极大地提高了网格模型在高维空间下的聚类效率。

(2) GTCLU 框架非常灵活, 可以被轻易地应用到其他的聚类模型中。

(3) 提出了在 GTCLU 上进行并行化计算的方法, 能够进一步提高其聚类速度。

4.2 GTCLU 框架设计

本文在第 3 章中已经介绍了网格空间的构建方法, 本节在假定已完成网格空间构建的基础上, 重点讨论如何加速网格的聚类过程。

如前文所述, 在高维空间下, 由于非空网格的数量急剧增长, 且通过数学方式快速计算邻居网格的方法不再适用, 导致网格聚类的搜索空间骤然增长, 进而使聚类速度急剧下降。那么一个自然的解决思路便是通过减小聚类搜索空间来加速网格聚类。

假设可以对原始网格空间进行进一步划分, 生成若干个较小的子网格空间, 能在这些小的子网格空间下应用任意的网格聚类算法快速地完成聚类, 再依赖有限的信息将这些子网格空间的聚类结果进行快速合并, 最终得到整个空间的聚类结果。那么这种方法将有效地减小网格的搜索空间, 最终加速网格模型的聚类速度。

基于这个思想, 本文设计了 GTCLU 聚类框架。GTCLU 通过一棵二叉树对网格空间进行进一步划分, 将最终得到的子空间存放在树的叶子节点中。同时, 在树的构建过程中, 将少量的必要信息保存在非叶子节点上用于子聚类结果的合并。在树构建完成时, 即完成了对网格子空间的划分过程。之后首先对叶子节点的子空间进行网格聚类, 然后在树的非叶子节点由底至上对其子节点的聚类结果进行快速合并, 最终在树的根节点得到整个网格空间的聚类结果。以上过程主要包含了两个步骤: (1) 空间划分和树的构建; (2) 基于树的网格聚类。

需要说明的是, GTCLU 聚类框架不会影响网格的聚类质量, 其聚类质量受网格模型设计的影响, 即网格空间如何构建和根据何种规则进行聚类。本框架是在网格模型的设计之上加速其聚类过程。由于本文在第 3 章中提出了一个新的网格模型, 该模型能够显著地提高网格聚类的聚类质量, 因此本章将以该网格模型作为基础, 通过 GTCLU 加速其在高维空间下的聚类速度。

4.2.1 空间划分和树的构建

定义 4.1 (切面邻居网格) 对于一个网格空间 G , 用一个切面 F 将其切分成两个子空间 G_1 和 G_2 , 与切面 F 存在相切面的网格称为切面 F 的邻居网格。其中落在子空间 G_1 中的邻居网格称为 G_1 的切面邻居网格, 落在 G_2 中的邻居网格称为 G_2 的切面邻居网格。

切面邻居网格即前文所提到的需要保存在非叶子节点中的少量必要信息, 根

据切面邻居网格可以快速地完成相邻子空间的聚类合并，其详细合并过程将在下一小节介绍。

空间划分和树的构建过程主要包含以下要点：

(1) 对于任意一个非叶子节点所代表的子网格空间 G ，先从这个空间中随机挑选若干个维度，并从其中选择当前的最长维度作为被划分维度。然后从被划分维度的中间位置将该网格空间划分为更小的两个子网格空间 G_1 和 G_2 ，并将 G_1 和 G_2 作为节点 G 的左右孩子节点。

(2) 如果一个非叶子节点 G 被切面 F 划分为两个子空间 G_1 和 G_2 ，那么在切分的过程中需记录下 F 的所有切面邻居网格，其中落在 G_1 中的切面邻居网格集记为 N_1 ，落在 G_2 中的切面邻居网格的集合记为 N_2 。同时将切面 F 的位置信息和 N_1 、 N_2 两个切面邻居网格集保存在节点 G 中，供后续合并聚类时使用。

(3) 对于落在切面邻居网格集 N_1 和 N_2 中的网格，在将其传递到子节点 G_1 和 G_2 中之前，需要先计算出其来自另一个集合中的外部密度。例如，对于网格 $g \in N_1$ ，需要在 N_2 中找到 g 的 ϵ -邻居网格集 $N_{g \leftarrow N_2}$ ，并算出 g 来自 $N_{g \leftarrow N_2}$ 的外部密度。

(4) 通过以上三点，可以完成对任意非叶子节点的划分。用根节点代表整个网格空间，并从根节点递归地对非叶子节点进行划分，直至满足递归终止条件为止，即可完成空间的划分和二叉树的构建。最终，整个空间中的网格被保存到各个叶子节点中，各个非叶子节点中仅保存切面信息和切面邻居网格。二叉树递归构建的终止条件通常可以设置为两个：其一，节点空间中的网格数量小于或等于给定阈值；其二，节点在树中的深度达到了给定的阈值。

在上述要点 (1) 中，通过引入随机性选择若干维度，并从中选择最长的维度来划分空间，可以在一定程度上避免空间划分不合理导致的聚类速度变慢。本文希望将一个空间中的非空网格尽可能平均地分布到两个子空间中，且切面邻居网格尽可能少，因此选择最大维度的中间进行划分。然而，网格的实际分配结果与数据的分布形态密切相关，无法预先知道哪种划分方法较好，因此引入了随机性以避免较差的结果发生。

在上述要点 (3) 中，需要预先计算两个切面邻居网格集中的网格来自对方的外部密度。按照第 3 章所述的模型，最终的聚类过程中需要计算每个网格的外部密度，且网格的外部密度来自于它的 ϵ -邻居。而被切分的两个子空间中实际上存在这样的 ϵ -邻居，且它们不会出现在同一个叶子节点当中被聚类，因此需要在空间划分时预先计算这部分外部密度。

至此，本文已经介绍了网格空间划分及其树构建过程的主要思想和方法。现在结合图 4-1 来更直观地解释该空间划分和树构建的过程。

图 4-1 用大写字母来表示网格空间，用小写字母来表示一个非空网格，其中图 (a) 表示网格空间的划分过程，图 (b) 表示对应空间树的构建过程。字母 A 在图 (a) 中表示整个网格空间，同时也在图 (b) 中表示该空间对应的根节点。空间 A 在纵轴的中间位置被划分成为两个子空间 B 和 C，并将 B 和 C 在树中作为节点 A 的孩子节点，同时在节点 A 中保存了其切面邻居网格集 $\{c\}$ 和 $\{k\}$ 。同理，继续递归地对空间 B 和 C 进行划分，将 B 划分为其子空间 D 和 E，将 C 划分为其子空间 F 和 G，并在 B 和 C 节点中保存了各自的切面邻居网格。本图的递归终止条件设为子空间的网格数量小于或等于 3，因此整个空间最终被划分为了 D、E、H、I、K、L、M 七个不能再分的子空间，同时这些子空间所包含的非空网格被存储在树叶子节点中。如叶子节点 D 中存储了 a、b、c 三个非空网格。至此，原始的网格空间 A 被划分完成并构建成了由若干个子空间所组成的一棵空间划分树。

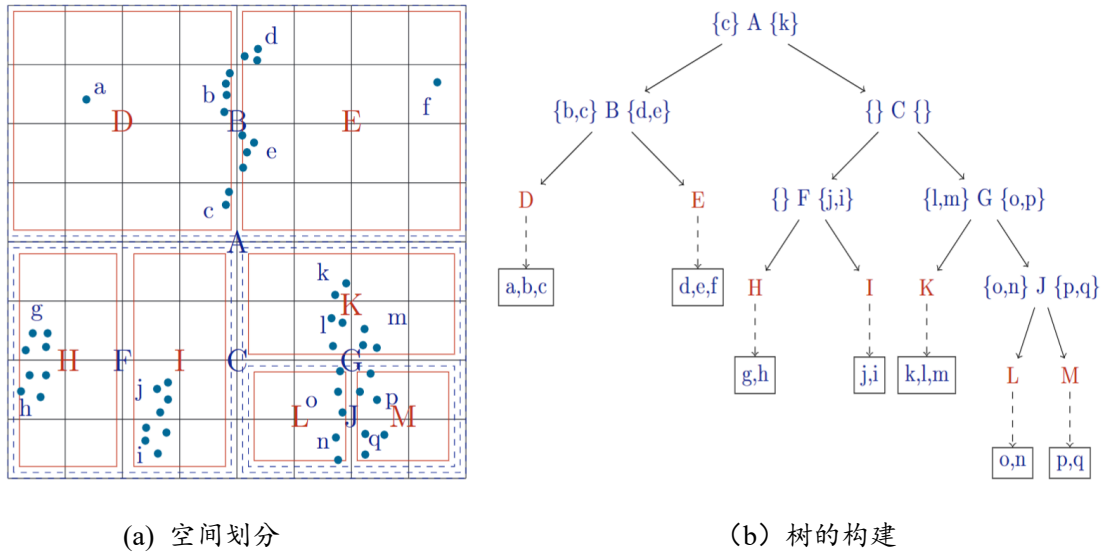


图 4-1 空间划分和树的构建过程

4.2.2 基于空间划分树的聚类

在上一小节中，本文介绍了空间划分树的构建过程，本节将探讨如何基于空间划分树对网格进行聚类。基于空间划分树的聚类主要分为两个步骤：

- (1) 在各个叶子上完成网格聚类，得到其对应子空间的聚类结果。
- (2) 由底至上，在非叶子节点上对其孩子节点的聚类结果进行合并，最终在根节点得到整个网格空间的聚类结果。

对于步骤 (1)，可以采用任意符合第 3 章所述的网格模型限制的聚类算法完成。其具体实现方法可参考算法 3-3，只需将其中通过计算得到邻居网格的方式改为遍历求 ϵ -邻居即可，本章将不再赘述该网格聚类算法的实现过程。

步骤 (2) 的合并过程主要基于非叶子节点所保存的切面邻居网格。设一个非叶子节点所代表的网格空间为 G ，他的两个孩子节点分别为 G_1 和 G_2 ，其对应的切面邻居网格集分别为 $N_1 \subset G_1$ 和 $N_2 \subset G_2$ 。假设 G_1 和 G_2 已经完成聚类，则在 G 上对其孩子节点的合并遵循以下原则：

- 假设网格 $g_1 \in N_1$ 、网格 $g_2 \in N_2$ ，且 g_1 、 g_2 均为核心网格。若这两个网格的距离满足 $\text{dist}(g_1, g_2) \leq \epsilon$ ，则将 g_1 和 g_2 所在的类合并为一个更大的类。
- 假设网格 $g_1 \in N_1$ 、 $g'_1 \in N_1$ 、 $g_2 \in N_2$ ，三个网格均为核心网格且 g_1 、 g'_1 不在同一个类中。若同时满足 $\text{dist}(g_1, g_2) \leq \epsilon$ 和 $\text{dist}(g'_1, g_2) \leq \epsilon$ ，则将 g_1 、 g'_1 、 g_2 分别所在的三个类合并为一个更大的类。同理，若有多个核心网格满足该条件，则将这多个类合并为一个类。
- 假设网格 $g_1 \in N_1$ 、 $g_2 \in N_2$ ，其中 g_1 为边缘网格且在 G_1 中被归为噪音网格， g_2 为核心网格。若满足 $\text{dist}(g_1, g_2) \leq \epsilon$ ，则将网格 g_1 加入到 g_2 所在的类中。
- 对于不满足以上三个条件的网格不进行合并操作。

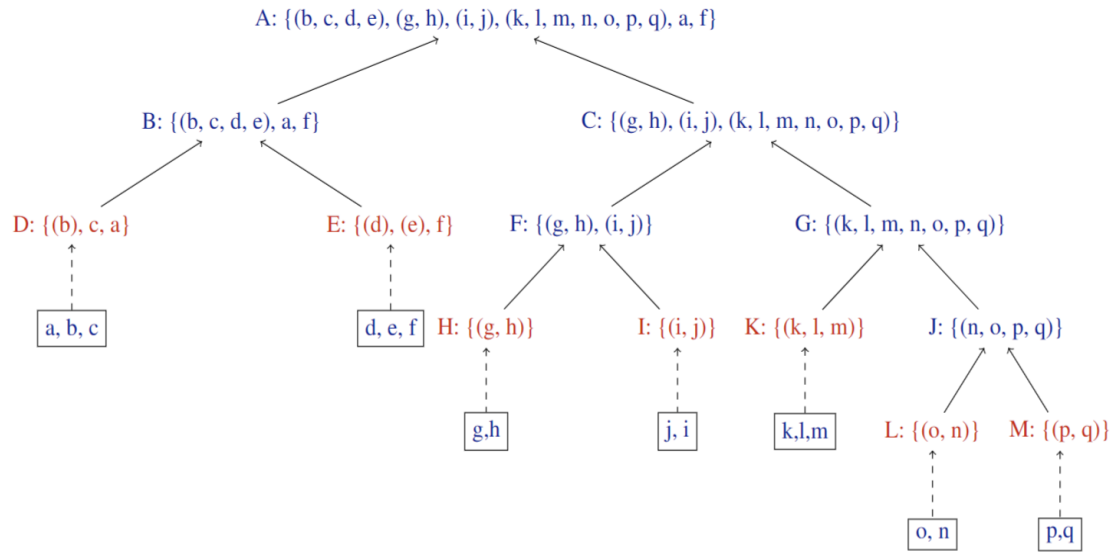


图 4-2 基于空间划分树的网格聚类过程

图 4-2 展示了对图 4-1 所构建的空间划分树的完整聚类过程。首先，通过调用一个网格聚类算法对图中所有的红色叶子节点完成聚类。例如，在叶子节点 D 上聚类得到了一个类 (b) 和两个噪音网格 c 、 a ，在叶子节点 E 上聚类得到了两个类 (d) 、 (e) 和一个噪音网格 f 。接下来，由底至上对非叶子节点进行聚类结果的合并。如在节点 B 上，根据保存在 B 中的切面邻居网格集 $\{b, c\}$ 、 $\{d, e\}$ 来对节点 D 、 E 的聚类结果进行，在节点 B 上得到了一个更大的类 (b, c, d, e) 和两个噪音网格 a 、 f 。待所有非叶子节点由底至上合并完成，最终在根节点 A 得到了整

个网格空间的聚类，其结果包含了四个类 (b, c, d, e) 、 (g, h) 、 (i, j) 、 (k, l, m, n, o, p, q) 和两个噪音网格 a 、 f 。

4.2.3 并行化计算及 GTCLU 的扩展应用

基于空间划分树的聚类过程可以轻松地扩展为并行计算。每个叶子节点的聚类过程是相互独立的，可以并行地对叶子节点进行聚类。类似地，同一层的非叶子节点的聚类合并过程也是相互独立的，可以进行并行化计算。因此，可以通过并行化进一步加速空间划分树的聚类过程。

此外，尽管本文将 GTCLU 应用于网格聚类，实际上 GTCLU 是一个易于灵活扩展应用的聚类框架。GTCLU 主要基于分治思想，如果某个聚类算法模型能够被划分到子空间计算，并能找到其合并子结果的方法，那么便可以轻易地将 GTCLU 扩展到该算法模型中。

4.3 GTCLU 算法实现

在 4.2 节中，本文完成了 GTCLU 框架的设计，介绍了该框架的主要实现方法和原理，并讨论了 GTCLU 的并行化计算和应用到其他算法模型中的可能性。本节中将详细介绍 GTCLU 在网格模型下的算法实现过程，并分析其时间复杂度和空间复杂度。

算法 4-1 GTCLU 聚类算法

输入 数据集 $D = \{p_1, \dots, p_n\}$ ，半径距离 ϵ ，密度阈值 $minPts$ ；递归停止条件：网格数量阈值 $maxNum$ ，节点深度阈值 $maxDepth$

输出 聚类标签集 $Labels = \{label_1, \dots, label_n\}$

- 1 $D' \leftarrow$ 将数据集 D 做归一化处理
- 2 $G \leftarrow$ 调用算法 3-2 生成网格空间
- 3 $Tree \leftarrow$ 调用算法 4-2 完成网格空间划分和空间划分树的构建
- 4 调用算法 4-4 完成空间划分树的网格聚类，给 G 中的网格打上类的标签
- 5 $Labels \leftarrow$ 调用算法 3-4，根据网格聚类标签给数据点打上类的标签

Return $Labels$

4.3.1 算法实现

对于一个给定的数据集 D ，输入两个密度聚类参数 ϵ 、 $minPts$ 和两个递归终止条件参数 $maxNum$ 和 $maxDepth$ ，GTCLU 的实现如算法 4-1 所示。其算法过程主要包含以下步骤：

(1) 对数据集 D 进行归一化处理，并结合参数 ϵ 生成由哈希表表示的网格空间 G 。

(2) 对网格空间 G 进行子空间划分, 并生成对应的空间划分树 $Tree$ 。

(3) 根据空间划分树 $Tree$ 对网格完成聚类, 其中包含叶子节点的网格聚类和非叶子节点的聚类结果合并等步骤, 最终在根节点得到整个网格空间的聚类结果, 并根据根节点的聚类结果对网格打上类的标签。

(4) 根据已经被标记了类标签的网格空间 G , 对原始的数据集打上类的标签, 最终返回该聚类标签集。

以上步骤 (1) 和步骤 (4) 在第 3 章已有实现, 详细过程可以参见算法 3-2 和算法 3-4。本章主要实现步骤 (2) 中关于网格空间划分及对应空间划分树的构建过程和步骤 (3) 中基于空间划分树的聚类过程。

算法 4-2 网格空间划分及对应空间划分树的构建过程

输入 网格空间表 G , 半径距离 ϵ , 密度阈值 $minPts$; 递归停止条件: 网格数量阈值 $maxNum$, 节点深度阈值 $maxDepth$

输出 空间划分树 $Tree$

- 1 $dimRange \leftarrow [[0, \dots, 0], [l, \dots, l]]$ // 新建一个二维数组, 表示当前空间的范围
- 2 $Tree \leftarrow$ 新建一棵空树, 由两个空的列表 $levels$ 和 $leaves$ 组成
- 3 $grids \leftarrow G.values()$ // 从网格空间表中提取出全部网格实例
- 4 $BuildTree(grids, 0, dimRange)$ // 调用算法 4-3 构建网格划分树, 将节点存入 $Tree$ 中

Return $Tree$

网格空间划分及对应空间划分树的构建过程如算法 4-2 和 4-3 所示。本文使用 $levels$ 和 $leaves$ 两个列表来表示一棵树。 $levels$ 是一个嵌套列表, 其中的每一个列表代表树的一层, 用于存储该层的所有非叶子节点。 $leaves$ 列表则存储所有的叶子节点。对于每个非叶子节点, 选取一个切面将节点空间划分为两个更小的子空间, 并将两个子空间作为该节点的孩子节点, 如算法 4-3 中第 5 到 9 行及 21、22 行所示。在空间划分的同时, 需要计算切面邻居网格的部分外部密度, 并将切面的信息和切面邻居网格保存在这个节点中供后续合并子聚类使用, 如算法 4-3 中第 10 到 16 行及 24 行所示。本文用一个二维数组 $dimRange$ 来表示当前空间在各个维度上的取值范围, 每次对节点的切割都需要更新 $dimRange$ 以适配其子空间的大小, 如算法 4-3 中第 21 行和 22 行所示。整个空间划分和建树的过程通过递归来完成: 如果一个节点达到了递归终止条件, 则将当前空间的网格存储在该节点中, 并将该节点作为叶子节点存储到树的 $leaves$ 列表中, 如算法 4-3 中第 1 行到第 3 行所示; 如果一个节点是非叶子节点, 则按上述的方法对其进行划分处理后, 将其存放到树的 $levels$ 中对应层的列表中, 如算法 4-3 中第 25 到 37 行所示。算法结束返回这个树, 树的所有叶子节点上保存了整个网格空间

的网格,而非叶子节点上保存了用于后续聚类合并的切面邻居网格和切面位置信息。

基于空间划分树的聚类过程如算法 4-4 和 4-5 所示,其中主要包含了对叶子节点进行网格聚类 and 自底向上对非叶子节点进行聚类合并两个步骤。叶子节点的聚类如算法 4-4 中第 1 到 2 行所示,首先从树的 *leaves* 列表中取出叶子节点,

算法 4-3 BuildTree(*grids*, *depth*, *dimRange*)

输入 当前空间的网格集 *grids*, 节点的深度 *depth*, 当前空间的大小 *dimRange*; 递归停止条件: 网格数量阈值 *maxNum*, 节点深度阈值 *maxDepth*

输出 树的节点 *node*

```

1  node  $\leftarrow$  新建一个空节点
2  if depth  $\geq$  maxDepth or grids.size  $\leq$  maxNum then
3      将grids保存在节点node中, 并将节点node加入Tree的叶子列表leaves中
4  else
5      splitDim  $\leftarrow$  随机选择几个维度, 将其中最长的维度作为划分维度
6      splitFace  $\leftarrow$   $\lfloor (\text{dimRange}[0][\text{splitDim}] + \text{dimRange}[1][\text{splitDim}]) / 2 \rfloor$ 
7      // 以splitFace为切面, 将该节点的网格分为两个子集, 并保存切面的邻居网格
8      lgrids  $\leftarrow$   $\{g \in \text{grids} \mid g.\text{pos}[\text{splitDim}] \leq \text{splitFace}\}$ 
9      rgrids  $\leftarrow$   $\{g \in \text{grids} \mid g.\text{pos}[\text{splitDim}] > \text{splitFace}\}$ 
10     ledges  $\leftarrow$   $\{g \in \text{lgrids} \mid g.\text{pos}[\text{splitDim}] = \text{splitFace}\}$ 
11     redges  $\leftarrow$   $\{g \in \text{rgrids} \mid g.\text{pos}[\text{splitDim}] = \text{splitFace} + 1\}$ 
12     // 根据这两个切面邻居网格集, 计算切面邻居网格来自对方的外部密度
13     foreach gi  $\in$  ledges do
14         // 其中Nredges(gi)表示gi在redges中的 $\epsilon$ -邻居网格集
15         
$$g_i.\text{ow} = g_i.\text{ow} + \sum_{g_j \in N_{\text{redges}}(g_i)} \min\left(\frac{0.5 \cdot \epsilon}{\text{dist}(g_i, g_j)}, 1\right) \cdot \text{iw}_{g_j}$$

16     对 redges 中的每个网格做以上同样的计算
17     // 更新两个子空间在被划分维度上的取值范围
18     ldimRange  $\leftarrow$  将dimRange[splitDim][1]设为splitFace
19     rdimRange  $\leftarrow$  将dimRange[splitDim][0]设为splitFace + 1
20     // 递归计算得到node的两个孩子节点
21     node.lchild  $\leftarrow$  BuildTree(lgrids, depth + 1, ldimRange)
22     node.rchild  $\leftarrow$  BuildTree(rgrids, depth + 1, rdimRange)
23     // 保存该节点的切面邻居信息, 并将该节点保存到树中对应层高的列表中
24     node.splitDim, node.ledges, node.redges  $\leftarrow$  splitDim, ledges, redges
25     while Tree.levels.size()  $\leq$  depth do
26         Tree.levels.add([ ])
27     Tree.levels[depth].add(node)
Return node

```

算法 4-4 基于空间划分树的网格聚类

输入 空间划分树 $Tree$, 半径距离 ϵ

输出 无显示输出; 根据网格聚类结果给网格打上类的标签

```

1  foreach  $node \in Tree.leaves$  do // 可以并行化计算
2       $node.clusters \leftarrow GridClustering(node.grids)$  // 对叶子节点进行网格聚类
3  for  $i \leftarrow Tree.levels.size() - 1$  to 0 do // 由低至上对各层非叶子节点合并聚类
4      foreach  $node \in Tree.levels[i]$  do // 同一层的节点可并行化计算
5           $node.clusters \leftarrow MergeChildren(node)$  // 调用算法 4-5 完成子空间聚类合并
6  // 根据根节点的聚类结果对各个网格打上类的标签
7   $clusters \leftarrow Tree.levels[0][0].clusters$ 
8  for  $i \leftarrow 0$  to  $clusters.size() - 1$  do
9      foreach  $g \in clusters[i]$  do
10          $g.label \leftarrow i$ 

```

算法 4-5 MergeChildren($node$)

输入 树的节点 $node$

输出 无显示输出; 对节点 $node$ 的孩子节点完成子聚类合并

```

1   $lclusters, rclusters \leftarrow node.lchild.clusters, node.rchild.clusters$ 
2   $ledges, redges \leftarrow node.ledges, node.redges$ 
3  // 1. 首先将切面邻居中的噪音网格合并到符合条件的类中
4  foreach  $g_l \in ledges$  and  $g_l$  是一个噪音网格 do
5      if  $g_r \in redges$  and  $g_r$  是核心网格 and  $dist(g_l, g_r) \leq \epsilon$  then
6          将网格  $g_l$  合并到  $g_r$  所在的类中
7  对  $redges$  中的噪音网格做同样的操作
8  // 2. 检查切面邻居中的核心网格, 将符合条件的类进行合并
9  foreach  $g_r \in redges$  and  $g_r$  是核心网格 do
10      $lCoreN \leftarrow \{g_l \mid g_l \in ledges \wedge g_l \text{ 是核心网格} \wedge dist(g_l, g_r) \leq \epsilon\}$ 
11      $lNclusters \leftarrow$  在  $lclusters$  中提取包含  $lCoreN$  中网格的全部类
12      $newCluster \leftarrow$  将  $g_r$  所在的类  $cluster_{g_r}$  和  $lNclusters$  中的类合并为一个新的类
13      $rclusters \leftarrow rclusters \setminus cluster_{g_r}$ 
14      $lclusters \leftarrow lclusters \setminus lNclusters \cup newCluster$ 
15   $node.clusters \leftarrow lclusters$ 

```

并调用任意符合本文第 3 章所述网格模型限制的聚类算法对叶子节点中的网格进行聚类。该调用的网格聚类算法实现可以参考算法 3-3, 只需把其中通过数学计算求邻居网格的部分改为遍历查找 ϵ -邻居即可。非叶子节点的聚类合并如算法 4-4 中第 3 行到 5 行所示, 从树的 $levels$ 列表中由底至上取出每一层非叶子节点,

并对每个非叶子节点进行聚类合并。对非叶子节点的聚类合并的实现如算法 4-5 所示, 首先处理切面邻居网格中的噪音网格, 如果一个孩子节点中的噪音网格同时是切面邻居网格, 而另一个孩子节点中的核心网格同时也是切面邻居网格, 且这两个网格的距离小于或等于 ϵ , 则将这个噪音网格合并到该核心网格所在的类中, 如算法 4-5 中第 3 到 7 行所示。接下来对子节点的类进行合并。如果一个切面邻居网格 g_r 是右孩子节点的核心网格, 那么 g_r 必然属于右孩子的一个类 $cluster_{g_r}$ 。在切面邻居网格中找到所有属于左孩子且与 g_r 距离不大于 ϵ 的核心网格, 并从左孩子中提取出这些核心网格所属的所有类 $LNclusters$, 则可以将 $cluster_{g_r}$ 与 $LNclusters$ 中的类合并为一个新的类, 并用这个新的类来更新左孩子节点中的类。当属于右孩子的核心切面邻居网格都经历上述操作后, 则该节点最终的聚类结果被存放在了左孩子中, 如算法 4-5 中第 8 到 15 行所示。当根节点对其孩子节点的合并完成后, 整个树的聚类过程就完成了, 最后利用根节点的聚类结果对网格空间 G 打上标签, 如算法 4-4 中第 6 到 10 行所示。

4.3.2 算法复杂度分析

GTCLU 算法的复杂度受空间划分树的节点数量、每个节点中网格的数量以及每次切割时的切面邻居网格数量的影响, 这些数据与数据集的分布形态和网格划分结果密切相关。为了分析 GTCLU 的算法复杂度, 需要先对这些数据做出合理的假设。设数据集点的数量为 n , 整个网格空间的非空网格数量为 $m < n$, 子空间的网格数量为 m' , $m' < m$ 且在绝大部分子空间下满足 $m' \ll m$ 。设一个非叶子左右两个孩子节点的切面邻居网格分别为 e 和 e' , 且令 $e' \leq e$ 。设空间划分树的非叶子节点个数为 x , 叶子节点的个数为 x' 。同时, 由于维度的计算是不可避免的, 因此为简化分析, 忽略维度计算: 假设维度为 d , 令一个数据点的存储空间为 $O(1)$, 两个点的距离计算时间也为 $O(1)$ 。

时间复杂度: 首先对数据集进行了归一化处理, 并根据数据集生成了网格空间, 对每个数据点只需要 $O(1)$ 的时间进行处理, 因此这一步需要花费 $O(n)$ 的时间。然后对网格空间进行划分并构成空间划分树。对于每一个非叶子节点: 通过 $O(1)$ 的时间确定切面, 然后用 $O(m')$ 的时间将所有网格保存到左右子节点及左右切面邻居网格集中。对于左右两个切面邻居网格集, 需要计算其来自相互的外部密度, 这一步需要 $O(e^2)$ 的时间, 通过 **kd-tree** 等索引结构可以将其优化到 $O(e \log e)$ 。因此划分一个非叶子节点总共需要 $O(m' + e \log e)$ 的时间。完成整棵的构建需要处理 x 个非叶子节点, 则建树的总时间为 $O(xm' + xe \log e)$ 。接下来对空间划分树进行聚类。首先对各个叶子节点进行网格聚类, 根据所调用的聚类算法, 每个叶子节点最快需要的聚类时间为 $O(m' \log m')$, 一共有 x' 个叶子节

点, 因此处理叶子节点的总时间为 $O(x'm' \log m')$ 。然后由底至上对非叶子节点进行聚类合并, 一个非叶子节点的合并需要 $O(m' + e \log e)$ 时间, 其中包括 $O(e \log e)$ 的时间计算左右切面邻居网格的 ϵ -邻居, 以及 $O(m')$ 的时间对子类进行合并。一共有 x 个非叶子节点, 因此聚类合并过程总共需要 $O(xm' + xe \log e)$ 的时间。最后对网格打标签需要 $O(m)$ 时间, 对数据点打标签需要 $O(n)$ 时间。由于 $m < n$, 因此对以上过程合并后得到 $O(n + x(m' + e \log e) + x'm' \log m')$ 的总时间复杂度。

空间复杂度: 原始数据集存储在硬盘中, 内存中只需存储非空网格, 且一个非空网格只需要 $O(1)$ 的存储空间, 因此整个非空网格空间需要 $O(m)$ 的存储空间。同时, 在树中只需要保存网格实例所在的地址, 一个树需要保存 m 个网格地址, 且网格地址所占的内存小于网格所需内存, 因此空间划分树所占的内存空间应该小于 $O(m)$ 。综上所述, 得到 GTCLU 的总空间复杂度为 $O(m)$ 。

4.4 实验分析

为了评估 GTCLU 算法的性能, 本文从聚类质量和聚类效率两个维度进行了实验分析。选定的对比算法包括两个密度聚类算法 DBSCAN 和 OPTICS, 以及两个网格聚类算法 BANG 和 CLIQUE。这四个算法经常作为对比算法出现在大量研究文章中, 具有广泛的代表性。同时, 这四个算法被大部分聚类相关的机器学习库所实现, 表明了它们在实践领域中得到了广泛的应用。本文在第 2 章中已对这四个对比算法进行了详细的介绍。

4.4.1 实验环境

本章的所有实验都是在同一台云服务器上进行的: 操作系统为 64 位的 Ubuntu 20.04.6 LTS 服务端版本; 处理器型号为 Intel Xeon Gold 6230@3.0GHz, 64 核; 内存大小为 160GB; 算法运行环境为 Python 3.10 解释器。

本实验的所有算法均采用 Python 实现, 其中 DBSCAN、OPTICS、CLIQUE、BANG 四个对比算法由 Python 的聚类算法库 Pyclustering^[86] 实现, 并关闭了其 C 语言加速的功能。本文的 GTCLU 代码采用 Python 3.10 版本实现。本实验的所有代码都在 Python 3.10 解释器下运行。

4.4.2 实验数据集

为了评估不同算法在聚类质量上的表现, 本文选择了 13 个广泛应用于聚类质量测试的基准数据集 (如表 4-1 所示, 编号 1 至 13)。数据集 1 至 6 包含不同分布形态的数据, 其中的类呈现多样形状; 而数据集 7 至 13 中的类符合高斯分布。具体如下: Compound 数据集 (来源于文献[87]) 包含 399 个二维数据点,

分为 6 个类；D31（来源于文献[88]）包含 3100 个二维数据点，分为 31 个类；R15（来源于文献[88]）包含 600 个二维数据点，分为 15 个类；Flame 数据集（来源于文献[89]）包含 240 个二维数据点，分为 2 个类；Iris 数据集（来源于文献[90]）包含 150 个四维数据点，分为 3 个类；Pathbased 数据集（来源于文献[91]）包含 312 个二维数据点，分为 3 个类。数据集 7 至 9（来源于文献[92]）分别包含二维、四维和八维的 2048 个数据点，每个数据集分为 2 个类。数据集 11 至 13（来源于文献[93]）均由 5000 个二维数据点组成，每个数据集包含 15 个类，其主要区别在于不同数据集中高斯分布的标准差设置不同。

表 4-1 测试数据集

编号	数据集	维度	类的数量	点的数量
1	Compound	2	6	399
2	D31	2	31	3100
3	Flame	2	2	240
4	Iris	4	3	150
5	Pathbased	2	3	312
6	R15	2	15	600
7	G2-2-30	2	2	2048
8	G2-4-30	4	2	2048
9	G2-8-30	8	2	2048
10	S1	2	15	5000
11	S2	2	15	5000
12	S3	2	15	5000
13	S4	2	15	5000
14	G10-10d-10k	10	10	10000
15	G20-20d-20k	20	20	20000
16	G10-30d-1m	30	10	1000000
17	G10-30d-10m	30	10	10000000

为了评估各算法在高维大数据集上的聚类效率，本文选取了编号为 14 到 17 的四个数据集来测试各算法的运行时间。由于 DBSCAN、OPTICS、CLIQUE、BANG 等四个算法在高维数据集上的运行效率较低，为了能得到它们的运行结果，本文在编号 14 和 15 两个较小规模的数据集上对它们的运行时间进行了测试。同时，为了证明 GTCLU 可以被应用于高维大数据的聚类分析，本文在 16 和 17 两个大规模数据上对其运行时间进行了测试。这四个数据集均采用 Scikit-learn 库生成，其中的类都符合标准差为 0.6 的高斯分布。各数据集具体如下：G10-10d-10k 由 1 万个 10 维的数据点组成，分为 10 个类；G20-20d-20k 由 2 万个 20 维的

数据点组成，分为 20 个类；G10-30d-1m 由百万个 30 维的数据点组成，分为 10 个类；G10-30d-10m 由千万个 30 维的数据点组成，分为 10 个类。

4.4.3 聚类质量分析

本文在编号 1 至 13 的十三个基准数据集上对各种聚类算法进行了质量评估。本实验选用了调兰德指数 (*ARI*) 和调互信息 (*AMI*) 作为评价指标，这两个指标分别可以衡量真实标签集与聚类标签集之间的相似性和一致性，并被广泛应用于各种聚类模型的质量比较。*ARI* 和 *AMI* 的取值范围在 $[-1,1]$ 之间，数值越大表示聚类质量越高，取值为 1 时意味着聚类结果与真实结果完全一致。本文在第 2 章已对 *ARI* 和 *AMI* 的两个指标进行了详细的介绍。

为了获得各个算法在不同数据集上的最佳聚类结果，我们对每个算法的参数在每个数据集上进行了大量的迭代。在 *ARI* 达到最大值时，选取相应的参数，并根据该参数下的聚类结果计算相应的 *AMI* 值。最终，各个算法在各数据集下的最佳聚类结果如表 4-2 所示。

表 4-2 聚类质量对比

评价指标	<i>ARI</i> <i>AMI</i>				
测试数据集	DBSCAN	OPTICS	BANG	CLIQUE	GTCLU
Compound	0.93 0.89	0.93 0.89	0.91 0.74	0.92 0.79	0.98 0.96
D31	0.84 0.91	0.80 0.90	0.66 0.80	0.64 0.86	0.84 0.91
Flame	0.97 0.93	0.96 0.92	0.92 0.84	0.87 0.73	0.97 0.93
Iris	0.94 0.89	0.94 0.89	0.99 0.97	1.00 1.00	1.00 1.00
Pathbased	0.92 0.89	0.95 0.92	0.50 0.59	0.67 0.67	0.95 0.92
R15	0.99 0.99	0.98 0.98	0.90 0.95	0.82 0.89	0.98 0.99
G2-2-30	0.91 0.84	0.91 0.84	0.68 0.51	0.86 0.70	0.94 0.88
G2-4-30	0.99 0.98	0.99 0.98	0.85 0.69	0.97 0.94	1.00 1.00
G2-8-30	0.85 0.79	0.85 0.79	1.00 1.00	0.98 0.95	1.00 1.00
S1	0.97 0.97	0.97 0.97	0.88 0.88	0.89 0.92	0.98 0.97
S2	0.75 0.88	0.75 0.88	0.67 0.77	0.63 0.80	0.75 0.85
S3	0.42 0.61	0.42 0.61	0.37 0.50	0.29 0.57	0.47 0.58
S4	0.43 0.54	0.43 0.54	0.41 0.50	0.19 0.47	0.45 0.53
平均值	0.84 0.85	0.84 0.85	0.75 0.75	0.75 0.79	0.87 0.89

根据结果分析，CLIQUE 算法仅在 Iris 一个数据集上取得了领先，其 *ARI* 和

AMI 两个指标在 13 个数据集下的平均值为 0.75 和 0.79。*BANG* 算法也仅在 G2-8-30 一个数据集上取得了领先，其 *ARI* 和 *AMI* 的平均值分别为 0.75 和 0.79。*CLIQUE* 和 *BANG* 两个网格算法在聚类质量方面表现相近。*DBSCAN* 的 *ARI* 指标在 4 个数据集上取得了领先，*AMI* 指标在 7 个数据集上取得了领先，其 *ARI* 和 *AMI* 的平均值分别为 0.84 和 0.85。*OPTICS* 的 *ARI* 在 2 个数据集上取得了领先，*AMI* 在 5 个数据集上取得了领先。虽然 *OPTICS* 在单项数据集上的最优测试结果略差于 *DBSCAN*，但在 *ARI* 和 *AMI* 的平均指标上取得了和 *DBSCAN* 同样的结果，分别为 0.84 和 0.85。这表明 *DBSCAN* 和 *OPTICS* 两个密度算法的聚类质量相当，但 *DBSCAN* 在部分数据集上优势更加突出，而 *OPTICS* 在各数据集上的测试结果更加均衡。本文提出的 *GTCLU* 算法的 *ARI* 指标在 13 个数据集上取得了领先，*AMI* 指标在 10 个数据集上取得领先，几乎在所有数据集上取得了聚类指标的优势。此外，*GTCLU* 在 *ARI* 和 *AMI* 的平均值上分别达到了 0.87 和 0.89，显著优于其他四个对比算法。从平均结果来看，*GTCLU* 在 *ARI* 上比 *DBSCAN* 和 *OPTICS* 两个密度聚类算法提高了 3.6%，在 *AMI* 上提高了 4.7%，证明 *GTCLU* 的聚类质量相较于密度聚类有一定程度的提升。与 *BANG* 和 *CLIQUE* 两个网格算法相比，*GTCLU* 在 *ARI* 指标上提高了 16.0%，在 *AMI* 上分别提高了 18.7% 和 12.7%，证明了 *GTCLU* 相较于如 *BANG* 和 *CLIQUE* 等传统的网格聚类算法在聚类质量方面带来了显著的提升。

4.4.4 聚类效率分析

本文从两个维度来评估各算法在高维数据下的聚类效率：首先，以聚类质量为准，测试各算法在达到给定的 *ARI* 指标时所需的最短时间；其次，通过设定多组参数，测试各算法在不同参数设定下的聚类时间。

(1) 以 *ARI* 为质量基准进行各算法的聚类效率对比分析

本实验首先以 *ARI* 分别为 0.6、0.7、0.8 和 0.9 为聚类质量标准，在样本量为 3 万、维度为 20 的数据集 G10-20d-30k 下对各个算法进行了测试。通过迭代大量的参数，选择算法达到指定 *ARI* 指标时的最快时间所对应的参数，并记录下了各算法在该参数设定下的平均聚类时间，其结果如图 4-3 所示。

从结果可以看出，本文提出的 *GTCLU* 算法在 *ARI* 取 0.6 到 0.9 中的任何值作为质量基准时，都能非常迅速地完成任务。当 $ARI \geq 0.9$ 时，*GTCLU* 仅需要 3.3 秒即可完成聚类，相较其它算法展现出了巨大的效率优势。*DBSCAN* 和 *OPTICS* 两个密度算法在这项对比实验中表现出了非常相近的性能。当 $ARI \geq 0.6$ 时，*DBSCAN* 最快在 642.1 秒完成聚类，*OPTICS* 最快需要 631.8 秒。当 $ARI \geq 0.9$ ，*DBSCAN* 最快需要 651.0 秒完成聚类，*OPTICS* 最快需要 665.1 秒。在这项

测试中, GTCLU 算法比 DBSCAN 和 OPTICS 两个密度聚类快了 197 倍以上。CLIQUE 算法在 ARI 的四项测试上都没能在限定时间内完成聚类, 证明 CLIQUE 网格算法不适用于高维数据的聚类分析。而 BANG 算法在四个对比算法中表现较好, 当 $ARI \geq 0.6$ 时, BANG 最快能在 144.9 秒内完成聚类。当 $ARI \geq 0.9$ 时, BANG 最快需要 226.9 秒, 本文的 GTCLU 算法在此项测试上比 BANG 快了 68.7 倍以上。

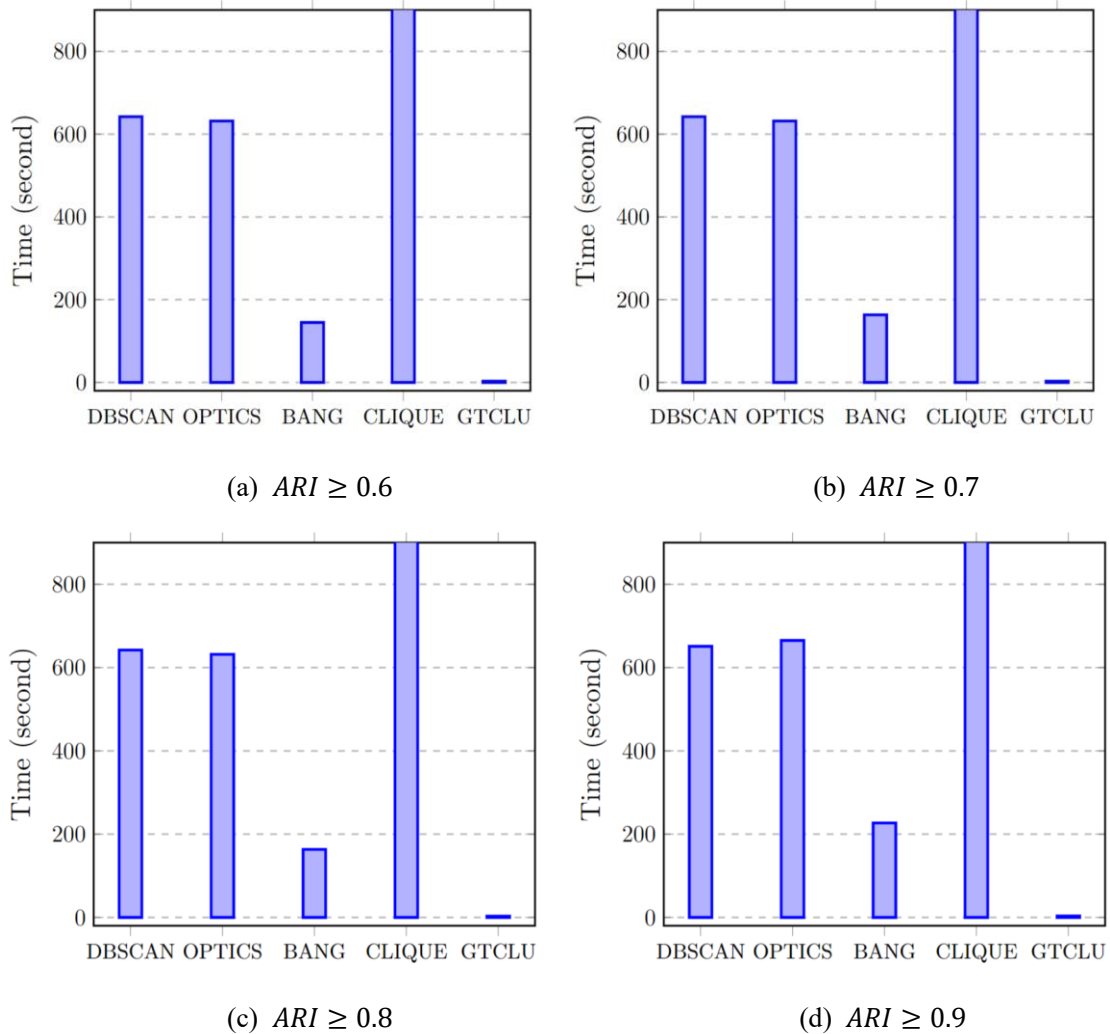


图 4-3 当 ARI 分别达到 0.6、0.7、0.8、0.9 时, 各算法在 G10-20d-30k 数据集上的最快聚类时间。

以 ARI 作为基础对算法进行聚类效率测试只能作为评价标准之一, 因为聚类质量受到数据集分布的影响较大, 而数据的分布会影响各算法达到相应 ARI 值的参数设定, 从而影响算法的运行时间。为了更全面地评估算法的聚类效率, 还需在多组不同参数下对各算法进行对比实验。

(2) 在多组参数下进行 GTCLU 与密度聚类的聚类效率对比分析

为了更好的展示 GTCLU 于 DBSCAN 等密度聚类在高维大数据集下的效率差异,本实验选择了样本量为 1 万的 10 维数据集来与 OPTICS 与 DBSCAN 进行对比。不选取更大数据集的原因是如果样本量过大或维度过高会导致 DBSCAN 和 OPTICS 超时,无法得到有效的对比结果。 ϵ 参数取值范围 0.1 到 1,步长为 0.1; $minPts$ 分别设置为 5、20 和 50。最终的实验结果如图 4-4 所示。

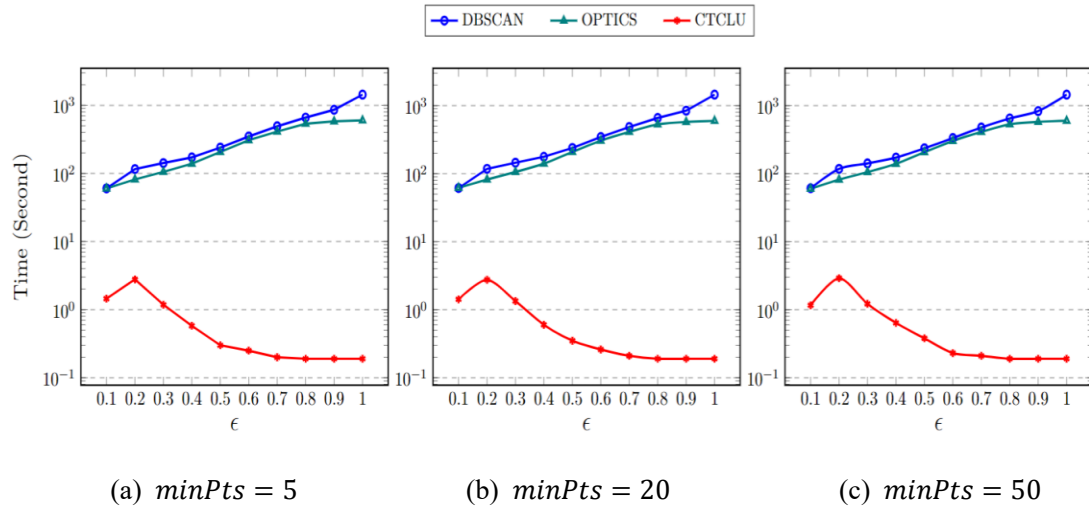


图 4-4 DBSCAN、OPTICS、GTCLU 在 G10-10d-10k 数据集上的聚类时间对比

从实验结果可以看到,在 G10-10d-10k 数据集上,GTCLU、DBSCAN、OPTICS 三个算法的聚类时间受 $minPts$ 参数的影响不大,主要受到 ϵ 参数的影响。GTCLU 算法的聚类时间随着 ϵ 的增大而下降,其中在 ϵ 为 0.2 时产生了波动。DBSCAN 和 OPTICS 算法的聚类时间随着 ϵ 的增大而增大。当 $minPts$ 为 5 时, DBSCAN 和 OPTICS 在 ϵ 取 0.1 时的聚类时间分别为 60.3 秒和 60.2 秒,在 ϵ 取 1 时的聚类时间分别为 1435.8 秒和 601.0 秒。当 $minPts$ 为 20 时, DBSCAN 和 OPTICS 在 ϵ 取 0.1 时的聚类时间分别为 61.4 秒和 61.9 秒,在 ϵ 取 1 时的聚类时间分别为 1451.1 秒和 597.6 秒。当 $minPts$ 为 50 时, DBSCAN 和 OPTICS 在 ϵ 取 0.1 时的聚类时间分别为 61.0 秒和 60.0 秒,在 ϵ 取 1 时的聚类时间分别为 1447.8 秒和 598.0 秒。同时参照图中曲线,可以看到 DBSCAN 和 OPTICS 在 ϵ 小于 0.8 时的聚类效率几乎相当,但随着 ϵ 的增大,OPTICS 开始显现出时间的优势,其聚类效率在 ϵ 为 1 时可以达到 DBSCAN 的 2.4 倍左右。本文提出的 GTCLU 算法在 ϵ 取 0.2 时聚类时间消耗最多,但均不超过 3 秒,在 ϵ 取 1 时的聚类时间不超过 0.2 秒。在 ϵ 分别为 0.1 和 0.2 时,GTCLU 分别对 DBSCAN 和 OPTICS 的聚类效率优势最小,是 DBSCAN 的 43 倍左右,是 OPTICS 的 29 倍左右。当 ϵ 为 1 时,GTCLU 的效率可以达到 DBSCAN 的 7600 倍以上,达到 OPTICS 的 3100 倍以上。GTCLU 聚类效率优势还会随着数据集和数据维度的增

长而进一步放大。本实验的参数设置覆盖了实践中不可能用到的极端范围,如 ϵ 取 0.1 时,整个网格空间的大小来到了 100 亿,远远超过了数据样本量的大小。

(3) 在多组参数下进行 GTCLU 与网格聚类的聚类效率对比分析

本实验同样在 G10-10d-10k 数据集上对 CLIQUE 和 GTCLU 算法在多组参数下的聚类时间进行了对比测试。基于与密度聚类对比的同样原因,没有选择更大数据集与 CLIQUE 相比较。CLIQUE 需要输入一个参数 l 对每个维度进行等量切分,进而将原始空间切分成由 l^d 个超立方体网格组成的网格空间。本文为了将 GTCLU 的参数与 CLIQUE 保持一致,同样选择 l 作为参数,并通过 $\epsilon = \sqrt{d} / l$ 来计算半径阈值。实验设置 l 为 2 到 10,选取 $minPts$ 分别为 5、20 和 50,最终得到的试验结果如图 4-5 所示。从结果中可以看到,CLIQUE 算法的聚类时间随着 l 的增大而急剧增长,在 l 为 4 时就超过了 4500 秒,在 l 为 5 时甚至超过了 1 万秒以上直至终止程序。而 CLIQUE 算法的聚类时间随着 l 的增大而缓慢增长,在 l 为 2 时仅需 0.15 秒即可完成聚类,在 l 为 10 时最大聚类时间也仅需 0.93 秒。

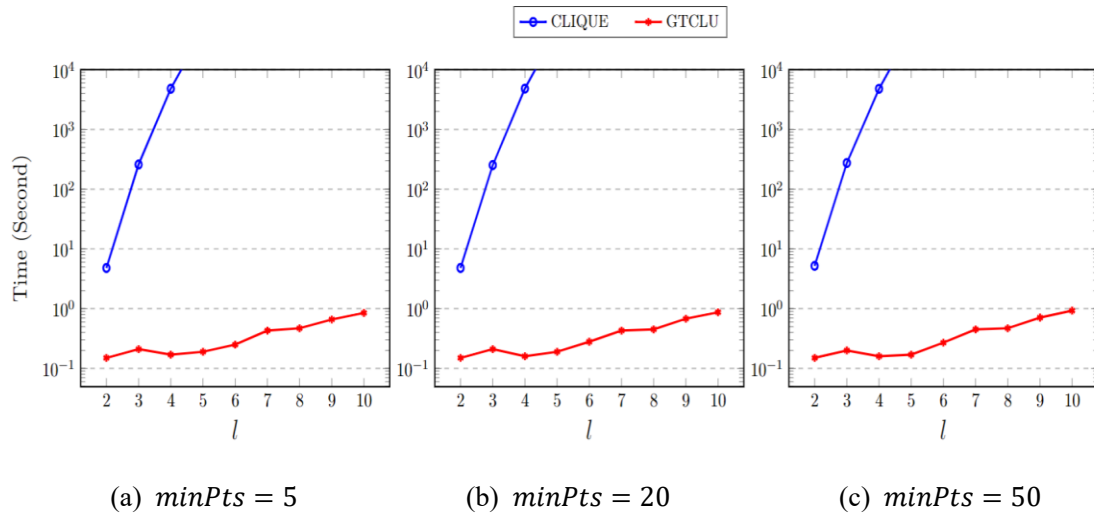


图 4-5 CLIQUE、GTCLU 在 G10-10d-10k 数据集上的聚类时间对比

BANG 算法在网格生成上与 CLIQUE 和 GTCLU 有显著差异,因此不能直接对比。BANG 采用树形结构将空间分割成网格,每次将原始空间分成两个子空间,最后在树的 $level$ 层停止划分,将原始数据空间划分成与叶子节点数相等的网格空间。为了比较 GTCLU 和 BANG 在相近参数值下的聚类时间,本文计算了 BANG 和 GTCLU 在参数 $level$ 和 l 下的最大网格数量,并据此将 $level$ 与 l 参数匹配。最终确定,当 GMCLU 的 l 参数在 2 到 9 范围内取值时,BANG 的 $level$ 参数应在 15 到 47 范围内取值。此外,BANG 中的 $threshold$ 参数表示当网格内点的数量小于等于该值时,将该网格视为噪声网格并停止划分,因此本实验将

$threshold$ 设为 $minPts - 1$ 。基于这些参数设置, 本实验在拥有 2 万个样本的 20 维数据集 G20-20d-20k 上测试了 GTCLU 和 BANG 的聚类时间, 其结果如表 4-3 所示。

从结果中可以看到, BANG 的聚类效率受到 $level$ 参数和 $threshold$ 参数的共同影响。因为 $level$ 参数增大, BANG 划分的网格数量随之增大, 而 $threshold$ 参数增大, BANG 会将大量网格视为噪音网格进而不再对其划分, 使得网格数量变小。而 GTCLU 在此数据集上的聚类效率相对稳定, 主要受参数 l 的影响, 其聚类时间随着 l 的增大而缓慢增长。BANG 在 $level$ 为 15 时, 在 $threshold$ 为 0、4、19 时的聚类时间分别为 28.3 秒、23.7 秒、18.2 秒。与之对应的 GTCLU 在 l 为 2 时的聚类时间均为 0.3 秒, 其效率分别比 BANG 快了 94.3 倍、79 倍、60.7 倍。当 $level$ 设置为 47, 与之对应的 l 设置为 10 时, BANG 在三个 $threshold$ 下的聚类时间分别为 953.9 秒、340.8 秒、158.2 秒, 而 GTCLU 在对应 $minPts$ 下的聚类时间为 6.0 秒、5.8 秒、5.9 秒, 其效率分别比 BANG 快了 159.0 倍、58.7 倍、26.8 倍。结果表明 BANG 在高维大数据集上相较于其它三个对比算法更加高效, 但本文的 GTCLU 算法还是比 BANG 有着更加显著的优势, 更适合应用于高维大数据的聚类分析。

表 4-3 GTCLU 与 BANG 在 G20-20d-20k 数据集上的聚类时间对比 (时间: 秒)

参数: l	2	3	4	5	6	7	8	9	10
参数: $level$	15	23	29	33	37	40	43	45	47
GTCLU: $minPts=1$	0.3	0.4	0.5	1.2	1.9	3.3	4.1	5.5	6.0
BANG: $threshold=0$	28.3	73.0	135.8	204.5	322.7	429.6	592.8	734.7	953.9
GTCLU: $minPts=5$	0.3	0.4	0.4	1.0	1.5	2.8	3.7	5.0	5.8
BANG: $threshold=4$	23.7	56.5	91.1	122.7	171.0	212.7	257.3	281.0	340.8
GTCLU: $minPts=20$	0.3	0.4	0.4	1.0	1.5	2.8	3.4	4.9	5.9
BANG: $threshold=19$	18.2	41.3	61.6	82.4	94.1	121.2	131.5	137.7	158.2

(4) 在样本量为千万的大数据集下测试 GTCLU 的聚类效率

由于其它四个对比算法在样本量过高时会导致算法超时, 因此无法得到在大样本数据集下的各聚类算法的对比结果。本文选择了一个百万样本量的数据集

G10-30d-1m 和一个千万样本量的数据集 G10-30d-10m 对 GTCLU 进行大数据聚类测试, 两个数据集的维度都为 30。 l 参数设置为从 2 到 5, 并通过 $\epsilon = \sqrt{d}/l$ 计算出半径阈值。 $minPts$ 参数设置分别设置为 1、5、10 和 20。需要说明的是, 由于网格空间的网格数量为 l^d , 与维度呈指数上升关系, 因此在 30 维的数据下, 当 l 设置为 2 时, 整个网格空间的容量已经超过了 10 亿, 远远超过了点的数量。这也是为何在此项测试中本实验仅将 l 的最大值设为 5。

此项测试的实验结果如图 4-6 所示。可以看到 GTCLU 在这两个高维大数据集下对 $minPts$ 参数表现出较低敏感性, 其聚类时间主要受到网格划分参数 l 的影响。且在 l 取值从 2 到 4 时聚类时间差异不大, 当到 5 时才出现时间剧增的情况。在样本量为百万的 30 维数据下, l 取值为 2 和 3 时, GTCLU 的聚类时间不超过 22 秒, 取值为 4 的聚类时间不超过 58 秒。 l 取值为 5 时虽然聚类时间突然上升, 但也不超过 168 秒。在样本量为千万的 30 维数据集下, GTCLU 在 l 取值为 2 和 3 时的聚类时间不超过 219 秒, 取值为 4 时的聚类时间不超过 452 秒。虽然当 l 取值为 5 时的最大聚类时间达到了 3400 秒, 但如前文所述, 该参数设置下的网格空间过于庞大。

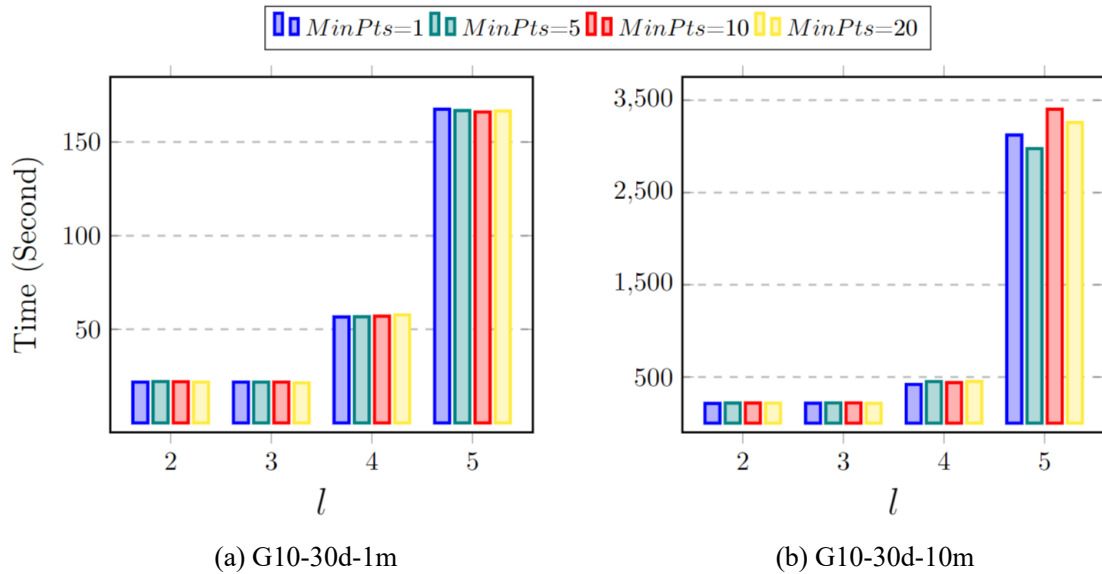


图 4-6 GTCLU 在 G10-30d-1m 和 G10-30d-10m 数据集下的聚类时间

综上所述, 本实验在高维数据下对 DBSCAN、OPTICS、CLIQUE、BANG、GTCLU 五个算法进行了对比测试。在聚类质量的测试环节中, 本文证明了 GTCLU 相较于 CLIQUE 和 BANG 两个网格算法带来了巨大的质量改善, 在 13 个基准数据集下的 ARI 指标的平均值提高了 16.0%, AMI 提高了 12.7% 以上。同时相较于 DBSCAN 和 OPTICS 两个密度算法也带来了一定程度的改善, 其 ARI 提高了 3.6%, AMI 提高了 4.7%。在聚类效率的测试环节中, 本文从以聚类质量

为基准的聚类时间测试和在不同参数设置下的聚类时间测试两个维度对各算法进行了对比,证明了 GTCLU 相较于其它四个对比算法具有显著的效率优势,且该优势随着数据样本量和维度的增长而进一步扩大。同时,本实验在样本量为百万和千万的 30 维数据集下对 GTCLU 进行了聚类时间测试,验证了 GTCLU 可有效地应用于高维度大数据的聚类分析。

最后,在图 4-7 中,本文直观地展示了 GTCLU 算法在四个常用测试数据集上的聚类效果图。

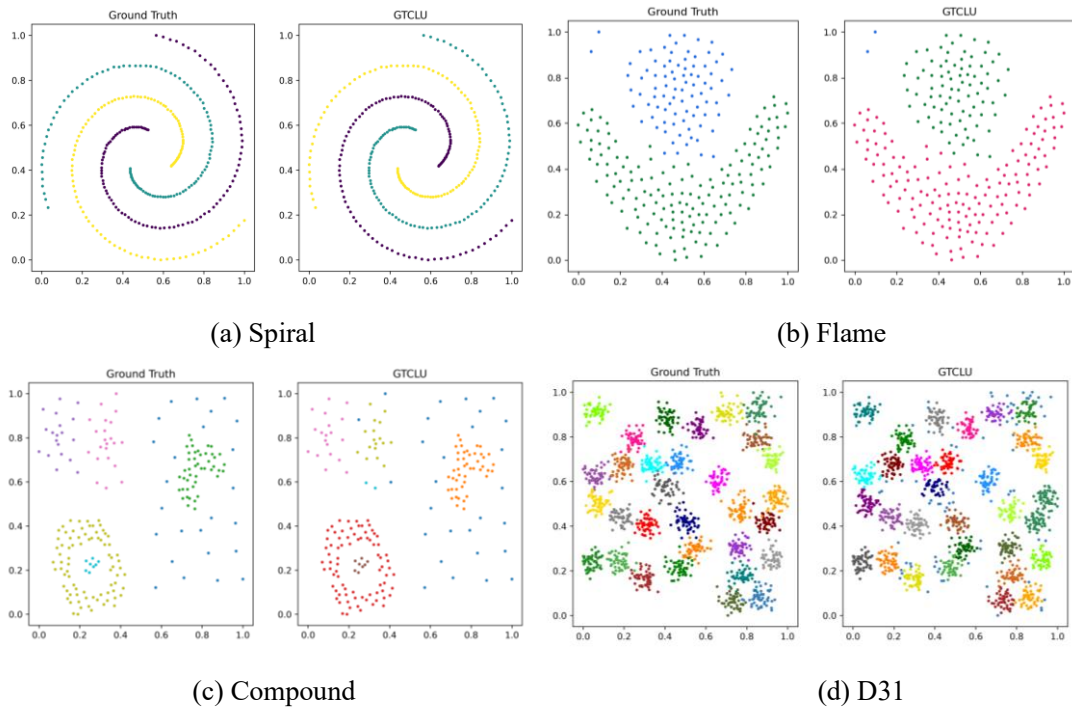


图 4-7 GTCLU 在 Spiral、Flame、Compound、D31 数据集上的聚类效果图

附注: 图中左图为真实标签结果, 右图为 GTCLU 的聚类结果。

4.5 本章小结

为了解决密度聚类 and 网格聚类在高维度大数据集下的低效问题,本章提出了一种基于树的子空间划分和子聚类合并的聚类框架,并根据该框架实现了网格聚类算法 GTCLU。本章的实验部分对 GTCLU 与 DBSCAN、OPTICS 两个密度聚类算法及 CLIQUE、BANG 两个网格聚类算法进行了对比测试,证明了 GTCLU 在提高聚类质量的同时,其聚类效率显著优于其它四个对比算法。实验最后使用样本量为一千万、维度为 30 的大数据集,验证了 GTCLU 能够有效应用于高维大数据的聚类分析。

第5章 总结与展望

5.1 研究工作总结

本文研究了在数据挖掘、机器学习等领域中的被广泛应用的聚类分析问题，主要聚焦于聚类分析中热门的密度聚类和网格聚类方向，研究密度聚类和网格聚类在大数据中的应用问题。网格聚类是密度聚类的一个重要的衍生方向，旨在改善密度聚类算法的聚类效率，但同时会带来聚类质量的下降。此外，网格聚类的聚类效率随着数据维度的增长而快速下降，无法处理高维大数据集。针对网格聚类存在聚类质量下降和在高维大数据集中聚类效率低下问题，本文展开了研究并取得了以下成果：

(1) 本文提出了一种新的网格聚类模型，该模型能够有效地提高网格的聚类质量。针对网格聚类中的网格划分问题，本模型将网格聚类中的网格划分参数 l 与密度聚类中的半径阈值参数 ϵ 相联系起来，通过半径阈值 ϵ 计算网格宽度，并保证了网格中最远两点的距离在 ϵ 范围之内，从而确保了同一网格中任意两点互为 ϵ -邻居。针对网格密度的计算问题，传统的网格聚类算法在计算网格密度时仅考虑当前网格内的点，会带来较大的聚类误差。本模型重新定义了网格密度的计算方法，考虑了网格周围的 ϵ -邻居网格对其产生的影响，并设计了有效的计算公式来计算结合了邻居网格的综合密度。这个新的密度计算方法可以有效地防止核心网格被误判为边缘网格，进而提高网格聚类的准确性。针对网格距离的计算问题，传统的网格聚类算法忽略了网格距离与点距离的差异，直接用网格距离来聚类网格。而本模型考虑了网格聚类的本质是对网格内点的聚类，对网格的表示方法进行了改进，在网格中存储了相关点的位置摘要信息，可以通过计算两个网格中点的距离来替代网格距离，使得网格聚类的结果更加准确。通过以上设计，本模型能够显著地提高网格的聚类质量，通过实验证明，本模型在 ARI 指标上比 BANG、CLIQUE 等网格算法提高了至少 16% 以上，在 AMI 指标上提高了至少 12.7% 以上。同时，跟 DBSCSN、OPTICS 等密度聚类算法相比，本模型也能在 ARI 和 AMI 指标上分别提高 3.6% 和 4.7%。

(2) 根据提出的网格模型，本文设计了一种适用于低维度大数据集的高效聚类算法 GMCLU，该算法在低维空间下能够达到 $O(n)$ 的时间复杂度。本文对 GMCLU 算法进行了系统的实验分析。通过与两个密度聚类算法和两个网格聚类算法的对比测试，验证了 GMCLU 在聚类效率上的巨大提升，并通过样本量为千万的数据集测试证明了 GMCLU 算法能够被有效地应用于低维大数据的聚类分析。

(3) 本文提出了一种基于树的空间划分和子聚类合并的算法框架, 该框架旨在解决网格聚类算法对高维大数据集聚类的低效问题, 并基于该框架实现了 GTCLU 聚类算法。网格聚类算法的效率瓶颈主要源于对邻居网格的查找及合并过程, 通常这个过程需要 $O(m^2)$ 的时间 (m 表示非空网格的数量), 虽然通过 kd-tree 等索引模型可以将其平均时间降低到 $O(m \log m)$, 但仍不能满足高维大数据的效率要求。且带 kd-tree 等索引模型聚类算法的最坏时间依然是 $O(m^2)$, 同时 kd-tree 等索引结构本身的建立和维护也会消耗大量时间。为了解决这个问题, 本文的核心思想是采取分治策略, 通过将原本的网格空间划分成更小的子空间来降低聚类网格的规模。本文的 GTCLU 算法通过一棵树来对网格空间进行子空间划分和维护, 能够在叶子节点上对最终的子空间快速完成聚类, 并根据非叶子节点上维护的少量网格信息快速完成子空间的聚类合并, 最终在根节点得到整个网格空间的聚类结果。本文对 GTCLU 算法进行了系统的实验测试, 在对比实验中证明了 GTCLU 算法在聚类质量和聚类效率上均能得到显著的提升。此外, 本实验通过在样本量为千万的高维数据集上证明了 GTCLU 可以被有效地应用于高维大数据的聚类分析。

5.2 展望

本文提出了一种新的网格聚类模型设计, 可以有效地提高网格的聚类质量。在该模型下, 本文提出了应用于低维大数据分析的聚类算法 GMCLU。针对高维大数据的聚类, 本文提出了一种基于树的空间划分和子聚类合并的算法框架, 并基于该框架实现了 GTCLU 算法。针对密度聚类和网格聚类研究, 尤其是在高维大数据中的聚类应用, 本文认为还有其它一些值得继续研究的工作。

(1) 尽管 GTCLU 通过子空间划分降低了高维大数据聚类中的网格规模, 有效地提高了网格的聚类效率, 但在超高维的数据中, 网格的划分和管理会面临更大的困难, 有可能会降低其聚类质量和效率。降维技术已被广泛地应用于大数据挖掘之中, 可以研究能否有效地将降维技术应用于 GTCLU 框架中, 在维持聚类质量的同时进一步提高其在超高维数据下的聚类效率。

(2) 数据流的聚类问题是当前的一个热门研究方向, 其对聚类的时效要求性更加严格, 一个高效的基础聚类算法能够有效地改善数据流的聚类问题。能否对 GTCLU 和 GMCLU 进行改进, 将其应用到数据流的聚类中, 是一个值得研究的方向。

(3) 可以探索如何自适应地调整 GTCLU 和 GMCLU 的算法参数, 以适应不同的数据场景, 从而提高算法的通用性和鲁棒性。

参考文献

- [1] Naeem M, Jamal T, Diaz-Martinez J, et al. Trends and Future Perspective Challenges in Big Data[C]. Proceedings of the 6th Euro-China Conference on Intelligent Data Analysis and Applications, 2022: 309-325.
- [2] 韩家炜, 裴健, 范明, 等. 数据挖掘: 概念与技术[M]. 北京: 机械工业出版社, 2012.
- [3] Shi Y. Advances in Big Data Analytics - Theory, Algorithms and Practices[M]. Springer, 2022.
- [4] Ghosal A, Nandy A, Das A K, et al. A Short Review on Different Clustering Techniques and Their Applications[J]. Emerging Technology in Modelling and Graphics, 2020: 69-83.
- [5] 张冬梅, 李敏, 徐大川, 等. K-均值问题的理论与算法综述[J]. 中国科学: 数学, 2020, 50(9): 1387-1404.
- [6] Lloyd S. Least Squares Quantization in PCM[J]. IEEE Transactions on Information Theory, 1982, 28(2): 129-137.
- [7] Sibson R. SLINK: An Optimally Efficient Algorithm for The Single-link Cluster Method[J]. The Computer Journal, 1973, 16(1): 30-34.
- [8] Ester M, Kriegel H P, Sander J, et al. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise[C]. Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, 1996, 96(34): 226-231.
- [9] 章永来, 周耀鉴. 聚类算法综述[J]. 计算机应用, 2019, 39(7): 1869-1882.
- [10] Gan G, Ma C, Wu J. Data Clustering: Theory, Algorithms, and Applications[M]. Society for Industrial and Applied Mathematics, 2020.
- [11] Anand S K, Kumar S. Experimental Comparisons of Clustering Approaches for Data Representation[J]. ACM Computing Surveys, 2022, 55(3): 1-33.
- [12] Géron A. Hands-on Machine Learning with Scikit-learn, Keras, and TensorFlow[M]. O'Reilly Media, 2022.
- [13] Schubert E, Zimek A. ELKI: A Large Open-source Library for Data Analysis. ArXiv Preprint ArXiv:1902.03616, 2019.
- [14] Ratra R, Gulia P. Experimental Evaluation of Open Source Data Mining Tools

- (WEKA and Orange)[J]. International Journal of Engineering Trends and Technology, 2020, 68(8): 30-35.
- [15] Meng X, Bradley J, Yavuz B, et al. Mllib: Machine Learning in Apache Spark[J]. The Journal of Machine Learning Research, 2016, 17(1): 1235-1241.
- [16] Giorgi F M, Ceraolo C, Mercatelli D. The R language: An Engine for Bioinformatics and Data Science[J]. Life, 2022, 12(5): 648.
- [17] Bouveyron C, Celeux G, Murphy T B, et al. Model-based Clustering and Classification for Data Science: with Applications in R[M]. Cambridge University Press, 2019.
- [18] Badillo S, Banfai B, Birzele F, et al. An Introduction to Machine Learning[J]. Clinical Pharmacology & Therapeutics, 2020, 107(4): 871-885.
- [19] Campello R J G B, Kröger P, Sander J, et al. Density-based Clustering[J]. WIREs Data Mining and Knowledge Discovery, 2020, 10(2): e1343.
- [20] Ezugwu A E, Ikotun A M, Oyelade O O, et al. A Comprehensive Survey of Clustering Algorithms: State-of-the-art Machine Learning Applications, Taxonomy, Challenges, and Future Research Prospects[J]. Engineering Applications of Artificial Intelligence, 2022, 110: 104743.
- [21] Bhattacharjee P, Mitra P. A Survey of Density Based Clustering Algorithms[J]. Frontiers of Computer Science, 2021, 15: 1-27.
- [22] Cheng W, Wang W, Batista S. Data Clustering[M]. Chapman and Hall/CRC, 2018: 128-148.
- [23] Benabdellah A C, Benghabrit A, Bouhaddou I. A Survey of Clustering Algorithms for An Industrial Context[J]. Procedia Computer Science, 2019, 148: 291-302.
- [24] Oyelade J, Isewon I, Oladipupo O, et al. Data Clustering: Algorithms and Its Applications[C]. Proceedings of the 19th International Conference on Computational Science and Its Applications, 2019: 71-81.
- [25] Starczewski A, Cader A. Grid-based Approach to Determining Parameters of the DBSCAN Algorithm[C]. Proceedings of the 19th International Conference on Artificial Intelligence and Soft Computing, 2020: 555-565.
- [26] Tareq M, Sundararajan E A, Harwood A, et al. A Systematic Review of Density Grid-based Clustering for Data Streams[J]. IEEE Access, 2021, 10: 579-596.
- [27] Rani P. A Survey on STING and CLIQUE Grid Based Clustering Methods[J]. International Journal of Advanced Research in Computer Science, 2017, 8(5).
- [28] Patel K M A, Thakral P. The Best Clustering Algorithms in Data Mining[C].

- Proceedings of the 6th International Conference on Communication and Signal Processing, 2016: 2042-2046.
- [29] Chen Y, Zhou L, Bouguila N, et al. BLOCK-DBSCAN: Fast Clustering for Large Scale Data[J]. Pattern Recognition, 2021, 109: 107624.
- [30] Sajana T, Rani C M S, Narayana K V. A Survey on Clustering Techniques for Big Data Mining[J]. Indian Journal of Science and Technology, 2016, 9(3): 1-12.
- [31] Edla D R, Tripathi D, Kuppili V, et al. Survey on Clustering Techniques[C]. Proceedings of the 2nd International Conference on Inventive Communication and Computational Technologies, 2018: 696-703.
- [32] Hinneburg A, Keim D A. An Efficient Approach to Clustering in Large Multimedia Databases with Noise[M]. Bibliothek Der Universität Konstanz, 1998.
- [33] Ankerst M, Breunig M M, Kriegel H P, et al. OPTICS: Ordering Points to Identify the Clustering Structure[J]. ACM Sigmod Record, 1999, 28(2): 49-60.
- [34] Campello R J G B, Moulavi D, Sander J. Density-based Clustering Based on Hierarchical Density Estimates[C]. Proceedings of the 17th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, 2013: 160-172.
- [35] Zhou S, Zhou A, Cao J, et al. Combining Sampling Technique with DBSCAN Algorithm for Clustering Large Spatial Databases[C]. Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2000: 169-172.
- [36] Borah B, Bhattacharyya D K. An Improved Sampling-based DBSCAN for Large Spatial Databases[C]. Proceedings of the 1st International Conference on Intelligent Sensing and Information Processing, 2004: 92-96.
- [37] Tsai C F, Liu C W. KIDBSCAN: A New Efficient Data Clustering Algorithm[C]. Proceedings of the 8th International Conference on Artificial Intelligence and Soft Computing, 2006: 702-711.
- [38] Gholizadeh N, Saadatfar H, Hanafi N. K-DBSCAN: An Improved DBSCAN Algorithm for Big Data[J]. The Journal of Supercomputing, 2021, 77: 6214-6235.
- [39] Ertöz L, Steinbach M, Kumar V. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data[C]. Proceedings of the 3rd SIAM International Conference on Data Mining, 2003: 47-58.
- [40] Laloux J F, Le-Khac N A, Kechadi M T. Efficient Distributed Approach for Density-based Clustering[C]. Proceedings of the 20th IEEE International

- Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2011: 145-150.
- [41] Yu Y, Zhao J, Wang X, et al. Cludoop: An Efficient Distributed Density-based Clustering for Big Data Using Hadoop[J]. International Journal of Distributed Sensor Networks, 2015, 11(6): 579391.
- [42] He Y, Tan H, Luo W, et al. Mr-dbscan: An Efficient Parallel Density-based Clustering Algorithm Using Mapreduce[C]. Proceedings of the 17th IEEE International Conference on Parallel and Distributed Systems, 2011: 473-480.
- [43] Corizzo R, Pio G, Ceci M, et al. DENCAST: Distributed Density-based Clustering for Multi-target Regression[J]. Journal of Big Data, 2019, 6: 1-27.
- [44] Yang K, Gao Y, Ma R, et al. DBSCAN-MS: Distributed Density-based Clustering in Metric Spaces[C]. Proceedings of the 35th IEEE International Conference on Data Engineering, 2019: 1346-1357.
- [45] Hosseini B, Kiani K. A Big Data Driven Distributed Density Based Hesitant Fuzzy Clustering Using Apache Spark with Application to Gene Expression Microarray[J]. Engineering Applications of Artificial Intelligence, 2019, 79: 100-113.
- [46] Yin C, Xia L, Zhang S, et al. Improved Clustering Algorithm Based on High-speed Network Data Stream[J]. Soft Computing, 2018, 22: 4185-4195.
- [47] Putri G H, Read M N, Koprinska I, et al. ChronoClust: Density-based Clustering and Cluster Tracking in High-dimensional Time-series Data[J]. Knowledge-Based Systems, 2019, 174: 9-26.
- [48] Oliveira A S, de Abreu R S, Guedes L A. Macro SOSStream: An Evolving Algorithm to Self Organizing Density-Based Clustering with Micro and Macroclusters[J]. Applied Sciences, 2022, 12(14): 7161.
- [49] Isaksson C, Dunham M H, Hahsler M. SOSStream: Self Organizing Density-based Clustering over Data Stream[C]. Proceedings of the 8th International Conference on Machine Learning and Data Mining in Pattern Recognition, 2012: 264-278.
- [50] Ouyang T, Shen X. Online Structural Clustering Based on DBSCAN Extension with Granular Descriptors[J]. Information Sciences, 2022, 607: 688-704.
- [51] Kim B, Koo K, Enkhbat U, et al. DenForest: Enabling Fast Deletion in Incremental Density-Based Clustering over Sliding Windows[C]. Proceedings of the 48th International Conference on Management of Data, 2022: 296-309.
- [52] Zubaroğlu A, Atalay V. Data Stream Clustering: A Review[J]. Artificial

- Intelligence Review, 2021, 54(2): 1201-1236.
- [53] Schikuta E. Grid-clustering: An Efficient Hierarchical Clustering Method for Very Large Data Sets[C]. Proceedings of 13th International Conference on Pattern Recognition, 1996, 2: 101-105.
- [54] Schikuta E, Erhart M. The BANG-clustering System: Grid-based Data Analysis[C]. Proceedings of the 2nd International Conference on Advances in Intelligent Data Analysis Reasoning about Data, 1997: 513-524.
- [55] Agrawal R, Gehrke J, Gunopulos D, et al. Automatic Subspace Clustering of High Dimensional Data[J]. Data Mining and Knowledge Discovery, 2005, 11: 5-33.
- [56] Wang W, Yang J, Muntz R. STING: A Statistical Information Grid Approach to Spatial Data Mining[C]. Proceedings of the 23rd International Conference on Very Large Data Bases, 1997: 186-195.
- [57] Wang W, Yang J, Muntz R. STING+: An Approach to Active Spatial Data Mining[C]. Proceedings of the 15th International Conference on Data Engineering, 1999: 116-125.
- [58] Hinneburg A, Keim D A. Optimal Grid-clustering: Towards Breaking the Curse of Dimensionality in High-dimensional Clustering[C]. Proceedings of the 25th International Conference on Very Large Data Bases, 1999.
- [59] Sheikholeslami G, Chatterjee S, Zhang A. WaveCluster: A Wavelet-based Clustering Approach for Spatial Data in Very Large Databases[J]. The International Journal on Very Large Data Bases, 2000, 8: 289-304.
- [60] Aggarwal C C. A Human-computer Interactive Method for Projected Clustering[J]. IEEE transactions on knowledge and data engineering, 2004, 16(4): 448-460.
- [61] Loh W K, Moon Y S, Park Y H. Fast Density-based Clustering using Graphics Processing Units[J]. IEICE Transactions on Information and Systems, 2014, 97(5): 1349-1352.
- [62] Chen Y, Tu L. Density-based Clustering for Real-time Stream data[C]. Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data mining, 2007: 133-142.
- [63] Reddy K S S, Bindu C S. StreamSW: A Density-based Approach for Clustering Data Streams over Sliding Windows[J]. Measurement, 2019, 144: 14-19.
- [64] Chen J, Lin X, Xuan Q, et al. FGCH: A Fast and Grid Based Clustering Algorithm for Hybrid Data Stream[J]. Applied Intelligence, 2019, 49: 1228-1244.

- [65] Bhatnagar V, Kaur S, Chakravarthy S. Clustering Data Streams Using Grid-based Synopsis[J]. Knowledge and Information Systems, 2014, 41: 127-152.
- [66] Amini A, Sabooi H, Herawan T, et al. MuDi-Stream: A Multi Density Clustering Algorithm for Evolving Data Stream[J]. Journal of Network and Computer Applications, 2016, 59: 370-385.
- [67] Deng C, Song J, Sun R, et al. GRIDEN: An Effective Grid-based and Density-based Spatial Clustering Algorithm to Support Parallel Computing[J]. Pattern Recognition Letters, 2018, 109: 81-88.
- [68] Ahmed R, Dalkılıç G, Erten Y. DGStream: High Quality and Efficiency Stream Clustering Algorithm[J]. Expert Systems with Applications, 2020, 141: 112947.
- [69] Li Y, Li H, Wang Z, et al. Esa-stream: Efficient Self-adaptive Online Data Stream Clustering[J]. IEEE Transactions on Knowledge and Data Engineering, 2020, 34(2): 617-630.
- [70] Taha K. Semi-supervised and Un-supervised Clustering: A Review and Experimental Evaluation[J]. Information Systems, 2023: 102178.
- [71] Fan J, Li R, Zhang C H, et al. Statistical Foundations of Data Science[M]. CRC Press, 2020.
- [72] Banerjee P, Chattopadhyay T, Chattopadhyay A K. Comparison among Different Clustering and Classification Techniques: Astronomical Data-dependent Study[J]. New Astronomy, 2023, 100: 101973.
- [73] Awasthi P, Balcan M F. Center Based Clustering: A Foundational Perspective[J]. Carnegie Mellon University, 2014.
- [74] Sinaga K P, Yang M S. Unsupervised k -means Clustering Algorithm[J]. IEEE Access, 2020, 8: 80716-80727.
- [75] Nielsen F. Hierarchical Clustering[J]. Introduction to HPC with MPI for Data Science, 2016: 195-211.
- [76] Cohen-Addad V, Kanade V, Mallmann-Trenn F, et al. Hierarchical Clustering: Objective Functions and Algorithms[J]. Journal of the ACM, 2019, 66(4): 1-42.
- [77] Schubert E, Sander J, Ester M, et al. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN[J]. ACM Transactions on Database Systems, 2017, 42(3): 1-21.
- [78] Rand W M. Objective Criteria for The Evaluation of Clustering Methods[J]. Journal of The American Statistical Association, 1971, 66(336): 846-850.
- [79] Cover T M. Elements of Information Theory[M]. John Wiley & Sons, 1999.

- [80] Rosenberg A, Hirschberg J. V-measure: A Conditional Entropy-based External Cluster Evaluation Measure[C]. Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, 2007: 410-420.
- [81] Rousseeuw P J. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis[J]. Journal of Computational and Applied Mathematics, 1987, 20: 53-65.
- [82] Davies D L, Bouldin D W. A Cluster Separation Measure[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1979 (2): 224-227.
- [83] Caliński T, Harabasz J. A Dendrite Method for Cluster Analysis[J]. Communications in Statistics-theory and Methods, 1974, 3(1): 1-27.
- [84] Chacón J E. A Close-up Comparison of the Misclassification Error Distance and the Adjusted Rand Index for External Clustering Evaluation[J]. British Journal of Mathematical and Statistical Psychology, 2021, 74(2): 203-231.
- [85] Newman M E J, Cantwell G T, Young J G. Improved Mutual Information Measure for Clustering, Classification, and Community Detection[J]. Physical Review E, 2020, 101(4): 042304.
- [86] Novikov A V. PyClustering: Data Mining Library[J]. Journal of Open Source Software, 2019, 4(36): 1230.
- [87] Zahn C T. Graph-theoretical Methods for Detecting and Describing Gestalt Clusters[J]. IEEE Transactions on Computers, 1971, 100(1): 68-86.
- [88] Veenman C J, Reinders M J T, Backer E. A Maximum Variance Cluster Algorithm[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24(9): 1273-1280.
- [89] Fu L, Medico E. FLAME, A Novel Fuzzy Clustering Method for the Analysis of DNA Microarray Data[J]. BMC Bioinformatics, 2007, 8(1): 1-15.
- [90] Tung A K H, Xu X, Ooi B C. Curler: Finding and Visualizing Nonlinear Correlation Clusters[C]. Proceedings of the 31st ACM SIGMOD International Conference on Management of Data, 2005: 467-478.
- [91] Chang H, Yeung D Y. Robust Path-based Spectral Clustering[J]. Pattern Recognition, 2008, 41(1): 191-203.
- [92] Fränti P, Mariescu-Istodor R, Zhong C. XNN Graph[C]. Proceedings of the Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition, 2016: 207-217.

- [93] Fränti P, Virmajoki O. Iterative Shrinking Method for Clustering Problems[J]. Pattern Recognition, 2006, 39(5): 761-775.
- [94] Bentley J L. Multidimensional Binary Search Trees Used for Associative Searching[J]. Communications of the ACM, 1975, 18(9): 509-517.
- [95] Omohundro S M. Five Balltree Construction Algorithms[M]. Berkeley: International Computer Science Institute, 1989.

攻读学位期间主要的研究成果

一、发表的学术论文

[1] 樊超,冯启龙. GBSCAN: 一种应用于大数据分析的密度聚类算法. 中南大学计算机学院学术年会优秀论文, 2022. (本人一作)

[2] 樊超,冯启龙. GTCLU: A Fast Tree-based Clustering Framework for Big Data. (拟投, 本人一作)

致 谢

时光荏苒，转眼间我即将度过在中南极为充实和宝贵的三年。在这段短暂而美好的时光里，我收获了太多的快乐和知识，也经历了很多困难和波折。在此，我想依依不舍的为研究生的求学经历画上一个句号，并感谢那些陪伴我度过这段时光的人们。

首先，我衷心感谢我的导师冯启龙教授，他在学术和研究上对我倾囊相授，给予了我巨大的支持与鼓励。冯老师严谨治学的态度和卓越的专业素养深深地感染了我，让我的学术能力得到了迅速的提升，并为我未来的职业生涯奠定了坚实的基础。在面对研究中的种种困惑和难题时，冯老师总是耐心细致地为我答疑解惑，带领我一步步战胜困难，渡过难关。在此，我要向冯老师表达我最真挚的感激之情。

其次，我要感谢石峰教授，他在科研过程中给予了我宝贵的建议和指导，对我的研究进展产生了积极的推动作用。同时，在我投稿新享期刊的文章时，石老师悉心审阅并提出了许多中肯的修改意见，使我的论文更加严谨和完善。对于石老师在我研究生生涯中的无私帮助，我表示由衷的敬意和感激。

同时，我也要感谢实验室的师兄、师姐、师弟、师妹们。他们在学习和科研中给予了我很多帮助。我们一起度过了无数个难忘的日夜，共同学习、探讨问题、取得进步。在我遇到困难时，他们总是毫不犹豫地伸出援手，让我倍感温暖。实验室充满的和谐氛围和团队精神让我不断成长，也让我们结下了深厚的友谊。我们一起学习、一起运动、一起娱乐的美好时光将成为了我人生中难以忘怀的珍贵回忆。

最后，我要感谢我的家人，尤其是我的父母和女朋友，他们一直是我坚实的后盾。父母在我的求学道路上给予了无尽的支持和鼓励，他们的无私奉献让我充满信心地迈向未来。女朋友的陪伴和关心让我的生活更加美好和充实。无论我遇到什么困难，他们总是陪伴在我身边，给我温暖和力量。我爱你们。

在回顾这三年的研究生生涯时，我由衷地感激和庆幸能够与你们相遇。正是有了你们，才使我的校园生活变得如此充实和难忘。展望未来，我将铭记导师的教诲和学校的培养，勇敢地迈向新的挑战。我会珍视这段时光的点滴，让这些宝贵的经历成为前行的动力。再次感谢我的导师、同门、朋友和家人，期待我们共同迈向更加美好的未来。