# 基于Spark的点排序识别聚类结构算法

瞿 原 邓维斌 胡 峰 张其龙 王 鸿
(重庆邮电大学计算智能重庆市重点实验室 重庆 400065)

**摘 要** 点排序识别聚类结构(Ordering Points to Identify the Clustering Structure,OPTICS)的密度聚类算法能以可视化的方式导出数据集的内在聚类结构,并且可以通过簇排序提取基本的聚类信息。但是该算法由于时空复杂度较高,不能很好地适应当今社会出现的大型数据集。随着云计算和并行计算的发展,提供了一种解决OPTICS算法复杂度缺陷的方法和一种建立在基于Spark内存计算平台的点排序识别聚类结构并行算法。测试的实验结果表明,它能极大地降低OPTICS算法对时间和空间的需要。

**关键词** 大数据,Spark,OPTICS算法,密度聚类
**中图法分类号** TP181 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.01.015

## Algorithm for Ordering Points to Identify Clustering Structure Based on Spark

QU Yuan DENG Wei-bin HU Feng ZHNG Qi-long WANG Hong
(Chongqing Key Laboratory of Computational Intelligence,Chongqing University of Posts and Telecommunications,Chongqing 400065,China)

**Abstract** Ordering points to identify the clustering Structure (OPTICS) is a hierarchical density-based data clustering algorithm,which can derive the intrinsic clustering structure of the dataset in a visual way,and can extract the basic clustering information by cluster sorting. However,due to its high temporal and spatial complexity,it can not adapt well to the large datasets in modern society. With the development of cloud computing and parallel computing,a method to solve the complexity of OPTICS algorithm was provided. This paper proposed a parallel OPTICS algorithm based on the Spark memory computing platform. The experimental results show that it can greatly reduce the time and space consumption of OPTICS algorithm.

**Keywords** Bigdata,Spark,OPTICS algorithm,Density based clustering

## 1 引言

聚类(clustering)算法研究是机器学习和数据分析领域的热点问题。迄今为止,聚类算法主要可分为四大类:1)分裂方法(partitioning methods),分裂聚类算法通过优化评价函数把数据集分割为$K$个部分,但是需要将$K$作为输入参数;2)层次方法(hierarchical method),层次聚类算法由不同层次的分割聚类组成,层次之间的分割具有嵌套的关系;3)基于密度的方法(density-based method),基于密度聚类的方法通过簇的密度连通性可以快速发现任意形状的簇;4)基于网格的方法(grid-based method),基于网格的聚类方法使用了一种多分辨率网格数据结构,所有聚类操作在网格结构上进行。在实际应用场景中常出现的簇都是不规则形状的簇,此时分裂方法和层次方法只能发现球形簇,不能很好地发现任意形状的簇,而基于密度的方法正好弥补了这一缺陷。

基于密度的聚类方法的主要思想在于:只要"邻域"中的密度(对于象或者数据点的数目)超过某个阈值,就继续增长给定的簇。对于给定簇中的每个数据点,在给定半径的邻域中必须至少包括最少数目的点[1]。基于密度的聚类算法具有可过滤噪声或离群点以及发现任意形状的簇的优点,因此许多学者在基于密度的聚类算法上作出了许多研究。比如,Kriegel等[2]提出了一种基于分层密度并能够适用于不确定型数据集的聚类算法,该聚类算法将两个模糊对象之间的相似性距离概率函数的概率值分配给每个可能的距离值;Viswanath等[3]讨论了一种快速的混合密度聚类方法,从粗粒度和细粒度两方面提出了两种不同的原型进行聚类;Hou等[4]提出了一种不需要人工提供参数的DBSCAN算法,能够快速寻找出任意形状的聚类且不需要输入任何参数;Amini等[5]提出了一种针对数据流的多密度聚类算法并采用网格结构有效地处理了噪声;Birant等[6]提出了一种新的基于密度的聚类算法,能够根据非空间、空间和时间上的对象值得到聚类结果。

到稿日期:2017-03-03 返修日期:2017-06-13 本文受国家自然科学基金项目(61309014,61379114,61472056),教育部人文社科规划基金项目(15XJA630003),重庆市基础与前沿研究计划(cstc2013jcyjA40063,cstc2014jcyjA40049),重庆市教委科学技术研究项目(KJ1500416)资助。
瞿 原(1993-),男,硕士,主要研究方向为数据挖掘和并行计算;邓维斌(1978-),男,博士,副教授,主要研究方向为智能信息处理和不确定性决策,E-mail:dengwb@cqupt.edu.cn(通信作者);胡 峰(1978-),男,博士,教授,主要研究方向为数据挖掘和机器学习;张其龙(1989-),男,硕士,主要研究方向为数据挖掘和机器学习;王 鸿(1992-),男,硕士,主要研究方向为数据挖掘和机器学习。

基于点排序识别聚类结构(Ordering Points to Identify the Clustering Structure,OPTICS)算法[7]是一种改进的 DBSCAN 算法,其思想与 DBSCAN 非常类似,但是与 DBSCAN 不同的是,OPTICS 算法可以获得不同密度的聚类,即经过 OPTICS 算法的处理在理论上可以获得任意密度的聚类。它弥补了聚类分析中使用一组全局参数的缺点,能够很好地刻画数据集内在的聚类结构并提供聚类结果的可视化。近年来,众多学者都在思考如何改进 OPTICS 算法并将其应用于实际场景中,从而得到满足期望的最终聚类结果。关于 OPTICS 算法的研究改进也获得了一定的进展,如 Patwary 等[8]提出了传统的 OPTICS 算法,采用图论中最小生成树思想进行聚类并进行了实验结果的验证;Goyal 等[9]提出在多核系统实现 OPTICS 算法并取得了很好的速度提升;Deng 等[10]改进了传统的 OPTICS 算法并且能够快速地聚类轨迹数据;Špitalsky 等[11]在公共电子邮件语料库上运用了一种自适应动态 OPTICS 算法并取得了较高的性能提升;Kalita 等[12]提出了一种基于排序三角形边长的改进 OPTICS 算法。

本文结合 Spark 内存计算框架,提出了一种基于 Spark 的点排序识别聚类结构算法(S-OPTICS)。S-OPTICS 算法结合传统的串行 OPTICS 算法及 Spark 中的广播、累加变量和 RDD 等数据结构,能够很好地适用于大型的数据集,因运用内存计算的方式,并行算法具有很高的运算性能。在单机和集群式的环境中的实验证明,算法具有处理大型数据集的能力和良好的可扩展性,在同等大小的数据集的情况下,其运算速度比 MapReduce 快 10～30 倍,在集群资源充足的情况下,随着并行程序使用的计算核心数的增加,程序的加速比也显著地增长。

## 2 基于点排序识别聚类结构算法

OPTICS 算法是对 DBSCAN 进行改进的一种密度聚类算法,并不显示地产生数据集的聚类,而是输出簇排序。这个排序是所有分析对象的线性表,并且代表了数据的基于密度的聚类结构[1]。OPTICS 算法输出的是数据集的可达距离线性表,从该线性表中可以获得任意密度的聚类结果。相关定义如下:

**定义 1**(ε-邻域[13]) ε-邻域是以给定对象为中心、以 ε 为半径的空间。

**定义 2**(核心对象[13],core object) 若一个对象的 ε-邻域至少包含 *MinPts* 个对象,则该对象是核心对象。

**定义 3**(直接密度可达[13],directly density reachable) 若 $p$ 是核心对象并且 $q$ 在 $p$ 的 ε-邻域内,则 $p$ 直接密度可达 $q$。

**定义 4**(密度可达[13],density reachable) 对于样本集合 $D$,给定一串样本点 $p_1,p_2,\cdots,p_n,p=p_1,q=p_n$,如果对象 $p_i$ 从 $p_{i-1}$ 直接密度可达,那么对象 $q$ 从对象 $p$ 密度可达。

**定义 5**(密度相连[13],density connected) 存在样本集合 $D$ 中的一点 $o$,如果对象 $o$ 到对象 $p$ 和对象 $q$ 都是密度可达的,那么 $p$ 和 $q$ 密度相连。

**定义 6**(核心距离[13],core distance) 对象 $p$ 的核心距离是指 $p$ 成为核心对象的最小 $\varepsilon'$,那么 $p$ 的核心距离定义为:

$$core\text{-}dist_{\varepsilon,MinPts}(p)=\begin{cases}UNDEFINED, & \text{if } |N_\varepsilon(p)|<MinPts\\ \text{MinPts-th smallest distance to } N_\varepsilon(p), & \text{otherwise}\end{cases} \tag{1}$$

**定义 7**(可达距离[13],reachability distance) 对象 $q$ 到对象 $p$ 的可达距离是使 $p$ 从 $q$ 密度可达的最小半径值,其定义如下:

$$reachability\text{-}dist_{\varepsilon,MinPts}(o,p)=\begin{cases}UNDEFINED, & \text{if } |N_\varepsilon(p)|<MinPts\\ \text{Max}(core\text{-}dist_{\varepsilon,MinPts}(p),dist(p,o)), & \text{otherwise}\end{cases} \tag{2}$$

由可达距离按顺序输出可得到最终的簇排序,并且代表了整个数据集的内在聚类结构。较稠密簇中的对象在簇排序中相互靠近,噪声对象会在簇排序的末尾部分堆积,在图中尾部形成急速的上扬。这个簇排序等价于从广泛的参数设置中得到的基于密度的聚类[1]。这样,OPTICS 算法不需要提供特定密度阈值就能得到广泛的聚类信息结构,便于导出最后的聚类结果。算法的具体过程如下。

输入:数据集 D,邻域半径 ε,最小邻居对象数 MinPts

输出:结果序列(即基于密度的簇排序)

Step1 建立两个队列:有序队列(核心对象及其直接密度可达对象)和结果队列(存储样本输出及处理次序);

Step2 若 D 中的数据全部处理完毕,则算法结束,否则从 D 中选择一个未处理且为核心对象的对象,将该核心对象放入结果队列,并将该核心对象的直接密度可达对象放入有序队列,所有直接密度可达对象按照可达距离升序排列;

Step3 若有序序列为空,则转 Step2,否则从有序队列中取出第一个对象;

Step4 判断该对象是否为核心对象,若不是则转 Step3,若该对象不在结果队列则将该对象存入结果队列;

Step5 若该对象是核心对象,找到其所有直接密度可达对象,并将这些对象放入有序队列,且将有序队列中的对象按照可达距离重新排序,若该对象已经在有序队列中且新的可达距离较小,则更新该对象的可达距离;

Step6 重复 Step3,直至有序队列为空;

Step7 算法结束,输出结果序列。

由于 OPTICS 算法的结构与 DBSCAN 非常相似,因此两个算法具有相同的时间复杂度。若使用空间索引,则其复杂度为 $O(n\log n)$,否则为 $O(n^2)$,其中 $n$ 为对象数。以簇排序画图,能够很快地得到聚类的层次结构。用一张二维图(见图 1)表示可达序列结果,横坐标表示结果序列的对象序列,纵坐标表示可达距离。由于聚类中最近的两个对象的可达距离最小,因此这些聚类在可达图中表现为波谷,波谷越深,聚类越密。

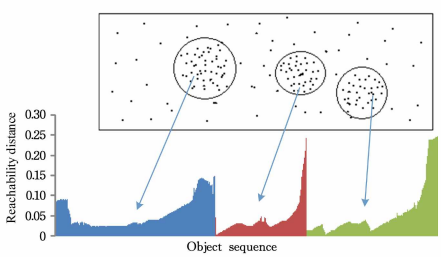


图 1 可达序列图

Fig. 1 Reachable distance sequence diagram

## 3　Spark 并行内存计算框架

Apache Spark 内存计算框架是加州大学伯克利分校所提出的开源类 Hadoop 的 MapReduce 并行计算框架,Spark 计算框架保留了 Hadoop MapReduce 的优点,并把计算过程中产生的中间结果缓存在内存中,减少了不必要的磁盘读写开销,大幅提高了计算速度[14]。Spark 框架因处理数据的方式不同,运算速度快于 MapReduce。Spark 的批处理速度是 MapReduce 的近 10 倍,内存中的数据分析速度则是 MapReduce 的近 100 倍。

Spark 的核心概念是分布式弹性数据集(Resilient Distributed Dataset,RDD),指的是只读的、可分区的分布式数据集,这个数据集的全部或部分可以缓存在内存中,可以在多次计算间重用。RDD 是 Spark 提供的最重要的抽象概念,是一种有容错机制的特殊集合,可以分布在集群的节点上,以函数式编程操作集合的方式进行各种并行操作。RDD 可以理解为具有容错机制的特殊集合,它提供了一种只读和只能由已存在的 RDD 变换而来的共享内存,将所有数据都加载到内存中以便进行多次重用。RDD 是分布式的,可以分布在多台机器上进行计算,其也是弹性的,当在计算过程中内存不够时它会与磁盘进行数据交换。图 2 给出了 Spark 处理数据的操作类型。

随着 Spark 内存计算框架的发展,许多的数据处理和机器学习算法被设计应用于 Spark 框架上,并获得了显著的提升。Bharill 等[15]提出了一种 Spark 环境下的并行模糊聚类算法,其性能与数据量成线性关系;Li 等[16]在 Spark 框架上利用近似的技术聚类来获得聚类效率和结果质量之间的权衡;Sinha 等[17]提出了一种基于 Spark 的 K-means 算法;Jin 等[18]实现了一种基于 Spark 的可扩展的层次聚类算法;Sarazin 等[19]提出了一种基于 Spark 和 MapReduce 的 SOM 算法,运算速度得到了大幅提高。

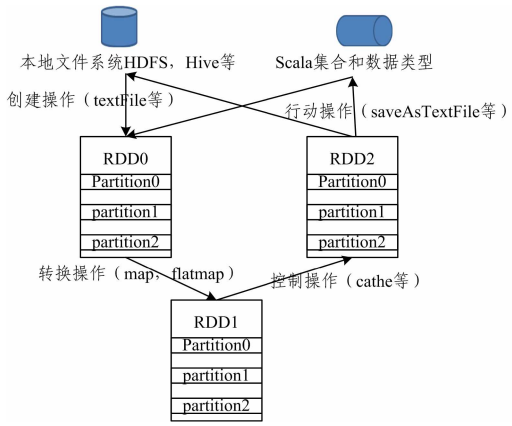


图 2　RDD 操作类型

Fig. 2　RDD operation type

## 4　基于 Spark 的 OPTICS 算法

本文提出了一种基于 Spark 内存计算框架的并行 OPTICS 算法——S-OPTICS 算法。S-OPTICS 算法的基本思想是:通过对数据集进行分割,在最大值与最小值差异较大的 5 个维度上进行划分,根据几个划分结果选取最优数据集划分结构并生成对应的 RDD。然后并行计算每个分区所有样本点的邻居样本点数及核心距离,对每个分区并行执行 OPTICS 算法,从而得到每个分区的簇排序并存储到分布式文件系统中。通过簇排序给每个分区的所有样本点赋予临时的分区簇,然后合并分区,最后每个样本点都能够得到全局的簇号。算法的流程图如图 3 所示。

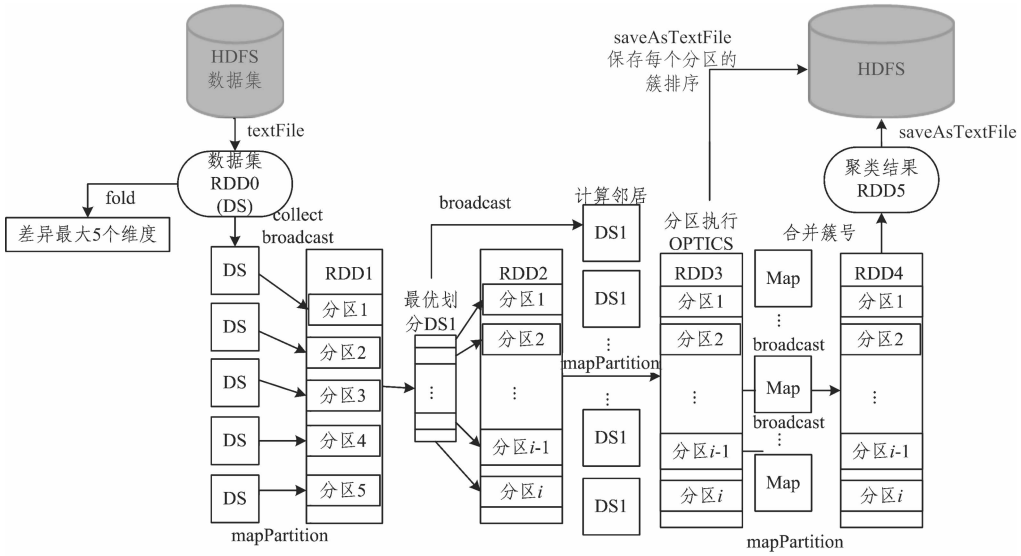


图 3　S-OPTICS 算法的流程图

Fig. 3　Flowchart of S-OPTICS algorithm

针对划分数据集,评价其划分是否平衡并找出最优划分,因此需寻找数据块序列划分长度最大及数据块样本数不倾斜的划分结构,即最优划分结构,故给出如下两个定义。

**定义 8**　平均样本数 $N$ 的计算如下:

$$N=G/L \tag{3}$$

其中,$G$ 为数据集的总样本数,$L$ 为划分数据块数量。

**定义 9**　倾斜值 $P$ 的计算如下:

$$P=N/M \tag{4}$$

其中，$N$ 为平均样本数，$M$ 为划分得到的数据块线性表中样本数最多的数据块样本数。

**定义 10**（最优划分值 $K$） 数据集的维度为 $n$，在每个维度进行划分可得到一组数据块长度序列 $L_1, L_2, \cdots, L_n$ 和一组倾斜值序列 $P_1, P_2, \cdots, P_n$，因此最优划分值 $K$ 为：

$$K=\max_{i\in\{0,\cdots,n\}}\left(\frac{L_i-L_{\min}}{L_{\max}-L_{\min}}+\frac{P_i-P_{\min}}{P_{\max}-P_{\min}}\right) \tag{5}$$

根据上述定义，可以在划分数据集时比较各个分区划分结构的具体信息，从而得到最优划分值 $K$ 的第 $i$ 个维度划分数据集最优，此时这个分区划分的数据集为最优划分。

S-OPTICS 算法的具体流程如下。

输入：数据集 DS，分区样本点最小个数 K，邻域半径 ε，最小邻居对象数 MinPts

输出：各分区簇排序，聚类最终结果

Step1 从 HDFS 上读取 DS 并将其转换为 RDD0，然后通过 cathe 算子保存 RDD0 于内存中。

Step2 RDD0 调用 reduce 算子寻找每一维度的最大最小值，并计算得到差异最大的前 5 个维度。若维度小于 5，则计算所有维度。

Step3 通过广播变量广播 DS 到每台机器上。新建 RDD1 并分配 5 个分区，每个分区分别按照差异最大的 5 个维度进行生成划分。

Step4 首先每个分区通过广播变量获取 DS，并按照分配的维度对其进行平分。将平分后得到的两个数据块再进行平分，直到数据块中样本点的个数小于 K 或者数据块的维度差小于 2 倍 ε，停止划分。每个分区最后可得到数据块形成的线性表。

Step5 通过累加变量得到 RDD1 的每个分区线性表长度 L 和数据块线性表中样本数最多的数据块样本数 M，根据式(3)－式(5)得到最优划分值 K，K 值最小分区就是最优划分分区，保存最优划分分区，去掉其余分区。最后调用 collect 算子得到 DS1。

Step6 DS1 按照自定义分区策略创建 RDD2，将第 i 块数据块分配到 RDD2 的第 i 个分区。

Step7 通过广播变量将 DS1 广播到每台机器上。RDD2 通过调用 mapPartition 算子，根据 RDD2 第 i 块分区中数据块的所有样本点和 DS1 中第 i 个数据块中样本点计算块内邻居样本点，根据 DS1 中第 i－1 个、第 i＋1 个数据块中的样本点计算块间邻居样本点。此时 RDD2 中的每个样本点都可以得到一组邻居样本点。

Step8 RDD2 调用 map 算子，通过邻居样本点及距离，根据式(1)得到每个样本点的核心距离，此时 RDD2 转换为 RDD3。

Step9 RDD3 调用 mapPartition 算子进行经典的 OPTICS 算法，并根据式(2)得到每个样本点的可达距离。每个分区根据结果序列得到每个分区数据块中样本点的簇排序，并将其保存到 HDFS 中。通过每个分区的簇排序来可视化观察和理解每个数据块的聚类结构，此时用户键入密度期望值 $\varepsilon_1$。

Step10 RDD3 调用 mapPartition 算子，根据 $\varepsilon_1$ 提取聚类簇号，每个样本点可得到在分区内的临时簇号，并将其转换为 RDD4。

Step11 RDD4 调用 filter 算子过滤非分界点，留下每个分区数据块的边界点，并调用 collect 算子得到数据块边界点数组(Boundary)。通过广播变量广播 Boundary 到每台机器上。

Step12 RDD4 调用 mapPartition 算子，每个分区各自创建独立的存储簇号，转换 Map，判断 RDD4 第 i 块分区中数据块的所有样本点与 Boundary 中第 i－1 个数据块中的样本点是否为邻居，即是否存在簇号转换关系，若存在，则把簇号转换关系加入 Map 中。

Step13 通过累加变量获得每个分区的簇号转换 Map，并将其整合到全局 Map 中，通过广播变量将全局 Map 广播到每台机器上。RDD4 调用 map 算子，通过全局 Map，每个样本点得到最终的簇号。此时 RDD4 转换为 RDD5，并存储到分布式文件系统 HDFS 上。

## 5 实验与结果

### 5.1 度量标准

1)纯度[20](*Purity*)

纯度是某个聚类结果 $S_r$ 中数量最多的类簇在聚类结果中的比例[20]。纯度定义公式如下：

$$P(S_r)=\frac{1}{n_r}\max_i(n_{ri}) \tag{6}$$

其中，$S_r$ 表示聚类的第 $r$ 类，$n_r$ 表示 $r$ 类的成员个数，$n_{ri}$ 表示第 $r$ 类成员分到第 $i$ 类的个数。因此，总体的聚类纯度通过单个聚类的纯度加权平均得到：

$$purity=\sum_r^k\frac{n_r}{n}P(S_r) \tag{7}$$

2)运行时间(*Runtime*)

实验中测试了 S-OPTICS 和 OPTICS 算法的运行时间。

3)加速比(*Speedup*)

为了验证并行算法和串行算法的效率提升，测试了算法的加速比：

$$Speedup=\frac{base\_line}{parallel\_time} \tag{8}$$

其中，*base_line* 为串行程序运行时间，*parallel_time* 为并行程序运行总时间。

### 5.2 数据集与运行环境

为了验证 S-OPTICS 算法的可行性和有效性，本文从 UCI[21] 机器学习数据库中选取了 14 个数据集进行实验，其中包含 9 个小型数据集，3 个中型数据集，2 个大型数据集。数据集概要信息如表 1 所列。

表 1 数据集信息

Table 1 Dataset information

| Dataset | Instances | Attributes |
|---|---|---|
| Iris | 150 | 4 |
| Haberman | 306 | 3 |
| Glass | 214 | 9 |
| LiverDisorders | 345 | 6 |
| Wine | 178 | 13 |
| Ionosphere | 351 | 34 |
| Landsat | 6435 | 36 |
| Phishing Websites | 11055 | 30 |
| Waveform(version1) | 5000 | 21 |
| Skin_NonSkin | 245057 | 3 |
| 3D Road Network | 434874 | 3 |
| Covertype | 581012 | 53 |
| SUSY | 5000000 | 17 |
| KDD Cup 1999 | 4000000 | 41 |

本实验使用的 Spark 平台是由 16 台服务器搭建的真实

物理集群,串行实验的运行环境和 Spark 的单服务器配置相同。表 2 列出了每个节点的硬件概要信息,表 3 列出了集群的软件概要信息。

表 2　单台结点的硬件概要信息

Table 2　Details of hardware in single node

| Processor type | CPU Cores | CPUfrequency /GHz | Cache capacity /MB | Memorycapacity /GB |
|---|---|---|---|---|
| Intel(R) Xeon(R) CPU E5-2620 | 24 | 2.0 | 15 | 64 |

表 3　集群的软件概要信息

Table 3　Details of software in cluster

| Operating System | Cent OS6.5 |
|---|---|
| Java Version | 1.7.0_71 |
| Scala Version | 2.10.4 |
| Hadoop Version | 2.5.2 |
| Spark Version | 1.4.0 |

### 5.3　实验方法

为了考察所提算法 S-OPTICS 的性能,本文将其与 Weka[22] 中的 OPTICS,BOPT[12],MOPTICS[8],MR-DBSCAN[23] 等算法分别进行了对比实验。然而因为 DBSCAN 算法的思想大体与 OPTICS 算法一样,时间复杂度也与 OPTICS 算法一致,所以使用 MR-DBSCAN 算法代替 OPTICS 算法在MapReduce上运行。为了提高实验的公平性和客观性,3 个串行的对比程序均均使用 Java 语言编程实现,它们与本文的 S-OPTICS 程序和对比的 MapReduce 版本的聚类程序均运行在同样的 JVM 环境中。实验先对并行 S-OPTICS 算法和 3 个串行算法在小型数据集和中型数据集上的运行时间和纯度进行比较,其次在大型数据集上进行并行度与算法时间的比较,最后进行 S-OPTICS 算法与 MR-DBSCAN 算法的加速比比较。

### 5.4　实验结果与分析

在小型数据集上运行串行算法和 S-OPTICS 算法的实验结果如表 4 和表 5 所列。其中,在小型数据集上 S-OPTICS 算法运行的并行度都为 8,即使用 CPU Cores 的数量为 8。而在实验的过程中,算法输入参数是在大致一样的情况下进行比较的。表 6 列出了串行算法和 S-OPTICS 算法在中型数据集上的实验结果。其中 SUSY,KDDCup 数据集前的 10w,20w 和 50w 表示的是两个数据集的子集,从中随机抽取 10 万,20 万和 50 万条数据。而 S-OPTICS 算法运行的并行度都为 144,即使用 CPU Cores 的数量为 144。

表 4　不同算法在小型数据集上的纯度对比

Table 4　Purity comparison of different algorithms on small datasets

| Dataset | OPTICS | BOPT | MOPTICS | S-OPTICS |
|---|---|---|---|---|
| Iris | 0.6667 | 0.6603 | 0.6667 | 0.6667 |
| Haberman | 0.7317 | 0.7325 | 0.7315 | 0.7319 |
| Glass | 0.3585 | 0.3604 | 0.3498 | 0.3536 |
| LiverDisorders | 0.5753 | 0.5768 | 0.5679 | 0.5686 |
| Wine | 0.6798 | 0.6830 | 0.6753 | 0.6793 |
| Ionosphere | 0.7036 | 0.7013 | 0.7094 | 0.7013 |
| Landsat | 0.6536 | 0.6415 | 0.6456 | 0.6475 |
| Phishing Websites | 0.5569 | 0.5503 | 0.5535 | 0.5569 |
| Waveform(version1) | 0.4115 | 0.4125 | 0.4236 | 0.4068 |

表 5　不同算法在中型数据集上的纯度对比

Table 5　Purity comparison of different algorithms on medium datasets

| Dataset | OPTICS | BOPT | MOPTICS | S-OPTICS |
|---|---|---|---|---|
| Skin_NonSkin | 0.7925 | 0.7915 | 0.7905 | 0.7925 |
| 3D Road Network | 0.3157 | 0.3147 | 0.3137 | 0.3099 |
| covertype | 0.5238 | 0.5268 | 0.5258 | 0.5178 |
| 10w-SUSY | 0.5794 | 0.5734 | 0.5744 | 0.5759 |
| 20w-SUSY | 0.5836 | 0.5846 | 0.5856 | 0.5839 |
| 50w-SUSY | 0.5415 | 0.5409 | 0.5419 | 0.5415 |
| 10w-KDDCup | 0.5803 | 0.5783 | 0.5793 | 0.5791 |
| 20w-KDDCup | 0.5843 | 0.5867 | 0.5847 | 0.5844 |
| 50w-KDDCup | 0.5655 | 0.5685 | 0.5695 | 0.5683 |

表 6　不同算法在中型数据集上的时间对比

Table 6　Time comparison of different algorithms on medium datasets

| Dataset | OPTICS | BOPT | MOPTICS | S-OPTICS |
|---|---|---|---|---|
| Skin_NonSkin | 2912 | 3025 | 3167 | 113 |
| 3D Road Network | 9878 | 9538 | 9668 | 454 |
| covertype | 78653 | 75154 | 75689 | 2027 |
| 10w-SUSY | 3687 | 3384 | 3445 | 251 |
| 20w-SUSY | 9802 | 9005 | 9454 | 362 |
| 50w-SUSY | 73771 | 70125 | 71236 | 2743 |
| 10w-KDDCup | 3554 | 3279 | 3312 | 75 |
| 20w-KDDCup | 8562 | 8045 | 8369 | 115 |
| 50w-KDDCup | 71042 | 70067 | 70536 | 575 |

表 4 中的实验结果可以表明在数据集一致的情况下,S-OPTICS算法因为需要分割计算数据集,所以在运算速度上比 3 个串行算法都慢,但是并行算法的纯度值基本与串行 OPTICS 程序大致保持一致。

从表 5 和表 6 的实验结果可知,S-OPTICS 算法在中型数据集上的运行速度明显快于 3 个串行算法,并且其纯度值也与串行算法大体一致。串行算法运行中型数据集的运算时间随着样本数的增加而大幅度增加。而 S-OPTICS 算法在并行度为 144 的情况下,仅为原来的 1/10 到 1/120 不等。因为有些数据集在几个维度上的数据分布都比较倾斜,所以其速度提升的效果不明显。但 S-OPTICS 在中型数据集上,其都取得了良好的效果,即使在数据倾斜的数据集上,其运行时间也能减少到可以承受的范围之内。

串行算法在 KDDCup 和 SUSY 数据集上的运行时间超过一周,因此表 6 中未列出串行算法的实验时间。图 4 给出了 S-OPTICS 在两个大型数据集上的实验结果。图 5 给出了在中型数据集 50w-KDDCup 上,MR_DBSCAN 算法和 S-OPTICS 算法的加速比比较结果,因为 MapReduce 的一个 Map 函数默认调用一个核心,所以实验比较都是在核心数相同下的情况下进行的。

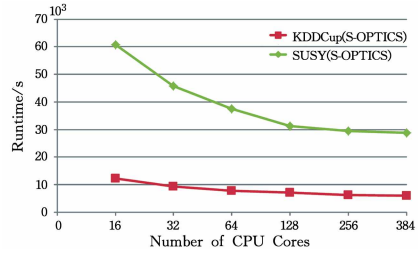


图 4　S-OPTICS 在大型数据集上的时间比较

Fig. 4　Time comparison of S-OPTICS on large datasets

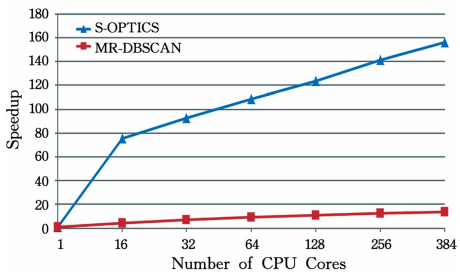


图 5 并行算法的加速比比较

Fig. 5 Speedup comparison of parallel algorithms

图 4 表明 S-OPTICS 程序的运行时间随着使用的核心数的增加而下降，但是下降的速度逐渐变慢。这是由于随着 CPU 核心数的增加，集群上各个机器之间的通信时间增加，其减少的算法计算时间逐渐变小，增加的数据通信时间逐渐变大，算法的性能趋于平缓甚至变弱。

图 5 表明在同等的数据集下，S-OPTICS 算法在实验数据集上的加速比随着 CPU 核心数的增加而增加，其加速比都是同等算法在 MapReduce 程序的 10～25 倍，也表明了随着核数的增加，S-OPTICS 计算聚类的时间会不断减少。相比于 MR_DBSCAN，S-OPTICS 算法在加速比上有较大优势，其原因在于：Spark 内存计算框架在运行计算数据时，把中间结果都放在内存中保存，而 MepReduce 框架会把中间结果不断地保存在磁盘上，进行 I/O 读写，从而导致运行时间增加。加速比也显示了 S-OPTICS 算法对于大规模数据集的良好适应能力和可扩展性。

**结束语** 本文基于 Spark 并行框架设计和实现了 OPTICS 算法的并行版本 S-OPTICS 算法，目的是使 OPTICS 这一聚类性能较好的密度聚类算法能适应于大规模数据集的聚类要求，从而满足工程和实际应用的需要。传统的机器学习技术从海量的数据中抽取知识，无法满足时间和空间的需求。Spark 提供了一种简单易用、透明、高效的并行接口，以便用户把有价值的机器学习算法应用到大数据上。本文结合 MapReduce 并行编程思想和 Spark 本身提供的内存迭代计算技术，提高了 S-OPTICS 算法适应大数据集的能力。在单机和集群上的实验表明，S-OPTICS 算法具有很好的加速比和扩展性，相对于串行程序，S-OPTICS 算法的聚类精确度并未受到影响；相对于 MapReduce 程序，S-OPTICS 算法的运行时间也大幅度减少。但 S-OPTICS 算法在不平衡的数据集中的运行时间会稍长。后续工作将进一步优化 S-OPTICS 算法在不平衡数据集上的性能，展开对其他聚类效果较好的聚类算法的并行化研究，使更多的现有聚类算法可以应用于海量数据挖掘的应用场景中。

## 参 考 文 献

[1] 韩家炜，坎伯，裴建. 数据挖掘：概念与技术(第 3 版)[M]. 北京：机械工业出版社，2012：288-291.

[2] KRIEGEL H P，PFEIFLE M. Hierarchical Density-Based Clustering of Uncertain Data[C]// Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Chicago，Illinois：ACM，2005：689-692.

[3] VISWANATH P，PINKESH R. I-DBSCAN：A Fast Hybrid Density Based Clustering Method[C]// International Conference on Pattern Recognition. HongKong：IEEE，2006：912-915.

[4] HOU J，GAO H，LI X. DSets-DBSCAN：A Parameter-Free Clustering Algorithm[J]. IEEE Transactions on Image Processing，2016，25(7)：3182-3193.

[5] AMINI A，SABOOHI H，WAH T Y. A Multi Density-Based Clustering Algorithm for Data Stream with Noise[C]// IEEE，International Conference on Data Mining Workshops. Dallas，Texas：IEEE，2013：1105-1112.

[6] BIRANT D，KUT A. ST-DBSCAN：An algorithm for clustering spatial-temporal data [J]. Data & Knowledge Engineering，2007，60(1)：208-221.

[7] 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.

[8] PATWARY M A，PALSETIA D，AGRAWAL A，et al. Scalable parallel OPTICS data clustering using graph algorithmic techniques[C]// International Conference for High Performance Computing，Networking，Storage and Analysis. New York：ACM，2013.

[9] GOYAL P，KUMARI S，KUMAR D，et al. Parallelizing OPTICS for multicore systems [C]// ACM COMPUTE. New York：ACM，2014：1-6.

[10] DENG Z，HU Y，ZHU M，et al. A scalable and fast OPTICS for clustering trajectory big data [J]. Cluster Computing，2015，18(2)：549-562.

[11] ŠPITALSKY V，GRENDÁR M. OPTICS-Based Clustering of Emails Represented by Quantitative Profiles[M]// Distributed Computing and Artificial Intelligence. Berlin：Springer，2013：53-60.

[12] KALITA H K，BHATTACHARYA D K，KAR A. A New Algorithm for Ordering of Points to Identify Clustering Structure Based on Perimeter of Triangle：OPTICS(BOPT)[C]// International Conference on Advanced Computing and Communications. Guwahati，Assam：IEEE Computer Society，2007：523-528.

[13] ANKERST M. OPTICS：ordering points to identify the clustering structure[J]. Acm Sigmod Record，1999，28(2)：49-60.

[14] ZAHARIA M，CHOWDHURY M，FRANKLIN M J，et al. Spark：cluster computing with working sets[C]// Usenix Conference on Hot Topics in Cloud Computing. Boston：USENIX Association，2010：1765-1773.

[15] BHARILL N，TIWARI A，MALVIYA A. Fuzzy Based Scalable Clustering Algorithms for Handling Big Data using Apache Spark[J]. IEEE Transactions on Big Datam，2016，4(2)：339-352.

(下转第 107 页)

表 7　不添加通配符的匹配次数

Table 7　The number of matches without wildcard

| 模式 | 匹配次数 | 运行次数 |
|---|---|---|
| $P_3=aOA$ | 3 | 761 |
| $P_4=aOAB$ | 1 | 867 |
| $P_5=baOAB$ | 0 | 930 |
| $P_6=baOABB$ | 0 | 1041 |

数据集 2 反映的规律与数据集 1 类似，并且对于股票交易数据而言，更小的数据波动意味着更少的精确匹配。

**结束语**　本文提出了一种适用于模式匹配的新方法。具体来说，先设计一个将时间序列转换为序列的编码表，然后定义弱通配符以及模式匹配的算法。对油井和股票数据的实验结果表明，这种新算法所匹配到的模式是符合要求的，匹配位置是准确的，匹配次数满足 $AP \leqslant GP \leqslant WP$，因此该方法对时序的相似性分析以及后续的数据预测是非常有意义的。

从所提算法以及进一步的研究来看，如何对匹配到的模式进行挖掘和分析是该算法的重要意义所在。由于大部分事物都具有多个属性，即含有多个时间序列，若能选取多个时间序列进行模式匹配，就可以匹配到更加精确的模式，更有利于开展后续工作。在这项工作中如何选择时间节点，以及如何提取专家模式来进行更合适的匹配是未来需要解决的问题。

## 参 考 文 献


[1] MONTGOMERY D C, JENNINGS C L, KULAHCI M. Introduction to time series analysis and forecasting[M]. John Wiley & Sons, 2015.

[2] SAKURAI Y, FALOUTSOS C, YAMAMURO M. Stream monitoring under the time warping distance[C]// 2007 IEEE 23rd International Conference on Data Engineering. IEEE, 2007: 1046-1055.

[3] HE Y, WU X, ZHU X, et al. Mining frequent patterns with wildcards from biological sequences[C]// 2007 IEEE International Conference on Information Reuse and Integration. IEEE, 2007: 329-334.

[4] LU C J, LEE T S, CHIU C C. Financial time series forecasting using independent component analysis and support vector regression[J]. Decision Support Systems, 2009, 47(2): 115-125.

[5] KEOGH E, KASETTY S. On the need for time series data mining benchmarks: a survey and empirical demonstration[J]. Data Mining and Knowledge Discovery, 2003, 7(4): 349-371.

[6] YANG Q, WANG X. 10 Challenging Problems in data mining research[J]. International Journal of Information Technology and Decision Making, 2006, 5(4): 597-604.

[7] FISCHER M J, PATERSON M S. String-matching and other products[C]// Proceeding of the 7th SIAM AMS Complexity of Com-putation. Cambridge, USA, 1974: 113-125.

[8] INDYK P. Faster algorithms for string matching problems: matching the convolution bound[C]// 39th Annual Symposium on Foundations of Computer Science. IEEE, 1998: 166-173.

[9] KALAI A. Efficient pattern-matching with don't cares[C]// Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms. Philadelphia, PA, USA: ACM, 2002: 655-656.

[10] MANBER U, BAEZA-YATES R. An algorithm for string matching with a sequence of don't cares[J]. Information Processing Letters, 1991, 37(3): 133-136.

[11] WU Y, WU X, MIN F, et al. A Nettree for pattern Matching with flexible wildcard Constraints[C]// IEEE International Conference on Information Reuse and Integration. IEEE, 2010: 109-114.

[12] MIN F, WU X, LU Z. Pattern Matching with Independent Wildcard Gaps[C]// Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing. 2009: 194-199.

[13] CHEN G, WU X, ZHU X, et al. Efficient string matching with wildcards and length constraints[J]. Knowledge & Information Systems, 2006, 10(4): 399-419.

[14] TAN C D, FAN M, WANG M, et al. Discovering patterns with weak-wildcard gaps[J]. IEEE Access, 2016, 4: 4922-4932.


（上接第 102 页）


[16] LI J, LI D, ZHANG Y. Efficient Distributed Data Clustering on Spark[C]// IEEE International Conference on CLUSTER Computing. Chicago: IEEE Computer Society, 2015: 504-505.

[17] SINHA A, JANA P K. A novel K-means based clustering algorithm for big data[C]// International Conference on Advances in Computing, Communications and Informatics. Jaipur: IEEE, 2016: 1875-1879.

[18] JIN C, LIU R, HENDRIX W, et al. A Scalable Hierarchical Clustering Algorithm Using Spark[C]// IEEE First International Conference on Big Data Computing Service and Applications. San Francisco Bay: IEEE, 2015: 418-426.

[19] SARAZIN T, AZZAG H, LEBBAH M. SOM Clustering Using Spark-MapReduce[C]// IEEE International Parallel & Distributed Processing Symposium Workshops. Phoenix: IEEE Computer Society, 2014: 1727-1734.

[20] CONRAD J G, AL-KOFAHI K, ZHAO Y, et al. Effective document clustering for large heterogeneous law firm collections[C]// Proceedings of the 10th international conference on Artificial intelligence and law. Bologna: ACM, 2005: 177-187.

[21] UCI Machine Learning Repository [DB/OL]. http://archive.ics.uci.edu/ml.

[22] Wikipedia Weka (machine learning) [CP/OL]. http://en.wikipedia.org/wiki/Weka, 2010.

[23] xj1986. MR-DBSCAN [EB/OL]. [2013-5-15]. https://github.com/xj1986/MR-DBSCAN.