IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 24, 2019, accepted January 10, 2020, date of publication January 27, 2020, date of current version February 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2969440

Adaptive Density-Based Spatial Clustering for Massive Data Analysis

ZIHAO CAI, JIAN WANG, AND KEJING HE, (Member, IEEE)
School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China

Corresponding author: Kejing He (kejinghe@ieee.org)

This work was supported in part by the Science and Technology Planning Project of Guangdong Province, China, under Grant 2017B030306016 and Grant 2019A050510024, in part by the Pearl River Science and Technology Star Project under Grant 201610010046, in part by the Special Support Program of Guangdong Province under Grant 201528004, and in part by the National Natural Science Foundation of China (NSFC) under Grant 61272200.

ABSTRACT Clustering is a classical research field due to its broad applications in data mining such as emotion detection, event extraction and topic discovery. It aims to discover intrinsic patterns which can be formed as clusters from a collection of data. Significant progress have been made by the Density-based Spatial Clustering of Applications with Noise (DBSCAN) and its variants. However, there is a major limitation that current density-based algorithms suffer from linear connection problem, where they perform poorly to discriminate objective clusters which are ''connected'' by a few data points. Moreover, the parameter setting and the time cost make it hard to be well-adapted in massive data analysis. To address these problems, we propose a novel adaptive density-based spatial clustering algorithm called Ada-DBSCAN, which consists of a data block splitter and a data block merger, coordinated by local clustering and global clustering. We conduct extensive experiments on both artificial and real-world datasets to evaluate the effectiveness of Ada-DBSCAN. Experimental results show that our algorithm evidently outperforms several strong baselines in both clustering accuracy and human evaluation. Besides, Ada-DBSCAN shows significant improvement of efficiency compared with DBSCAN.

INDEX TERMS Clustering, density-based algorithms, linear connection, data block splitter, data block merger.

I. INTRODUCTION

With the growing of large collection of data in various domains like business management and cloud services [1], much attention has been given to data mining algorithms, which can be applied to many tasks such as event detection [2], personalized recommendation [3] and the Internet of Things (IoT) [4]. As a typical data mining algorithm, clustering shows broad applications in data analysis, where density-based clustering algorithms play a crucial role [5]. Conventional density-based clustering algorithms such as Density-based Spatial Clustering of Applications with Noise (DBSCAN) [6] and its variant OPTICS [7], can discover cluster structures of data points as well as filter out noise. It is the advantage of DBSCAN that makes it require no prior knowledge of the number of clusters in contrast to distance-based clustering algorithms like K-Means [8]. Previous studies revealed that DBSCAN can be widely applied in numerous fields [9]–[11].

Several studies have been explored to improve the shortcomings of DBSCAN, such as parameter setting improvement [7] and efficiency optimization [12]–[14]. Despite the progress of these algorithms, there are still several challenges for discovering clusters effectively and efficiently, which are not addressed well in previous works. **First**, DBSCAN and its extended density-based clustering algorithms suffer from linear connection problem, which refers to that they perform poorly when different objective clusters are ''connected'' by a few data points with strong inner association shaped like a line. Current density-based clustering algorithms are difficult to distinguish such patterns and tend to recognize the two objective clusters as the same cluster. As an example occurs in event detection shown in Fig. 1, the left-side words are covered by event of quantum mechanics, while the right-side words are covered by event of computer science. There are several words with strong relevance between two objective

The associate editor coordinating the review of this manuscript and approving it for publication was Ahmet M. Elbir.

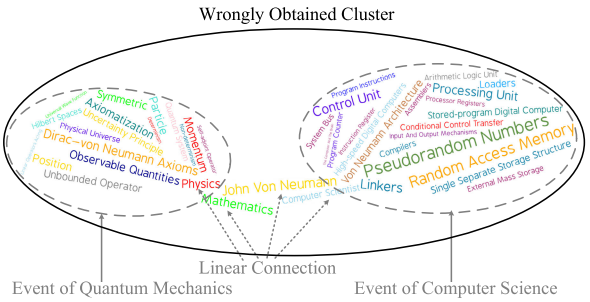


FIGURE 1. An example of linear connection in clustering.

events, i.e., ''Physics'', ''John Von Neumann'', ''Mathematics'' and ''Computer Scientist'', which prevent DBSCAN from discriminating the two cluster structures, resulting in an incorrect cluster obtained finally. It is because the data points located in the line connecting two individual clusters are actually directly reachable to data points located in both clusters, causing DBSCAN widen the radius range of clustering. Although some methods are proposed to set parameters more stringently to obtain cluster centers and then expand them cautiously [15], they still suffer from another problem because adjusting parameters may break the structures of either originally discovered clusters or other clusters out of linear connection. More seriously, some clusters may no longer be discovered with strictly constrained parameters, since the data points causing linear connection may have strong correlation with each other which should be regarded as centers of clusters. Furthermore, other density-based clustering algorithms almost fail to take the linear connection problem into consideration.

Second, DBSCAN requires two key parameters keeping fixed to perform clustering, including the maximum radius range (*eps*) and the minimum number of neighbors (*minPts*). The fixed parameters deteriorate DBSCAN to adapt the data with heterogeneous density as stated in [6], since the performance of clustering is sensitive to initial values of the two parameters. Although an optimized version called OPTICS [7] was proposed to improve the parameter setting by ordering data points to identify the clustering structure, it is still sensitive to the density of data.

Third, the computational efficiency of DBSCAN is low, making it limited in massive data analysis. The calculation of Cartesian products is required in DBSCAN to determine the degrees of relevance between data points, costing $O(N^2)$ time complexity, which will spend high time cost on massive data if being applied in practical applications.

To deal with the aforementioned limitations, we propose a novel Adaptive Density-based Spatial Clustering of Applications with Noise (Ada-DBSCAN), which consists of a data block splitter and a data block merger, coordinated by local clustering and global clustering. The data block splitter first allocates data points to a set of data blocks hierarchically based on the core idea of uniform data distribution which is an essential premise for alleviating previous drawbacks. The data block splitter guarantees that the data blocks containing the region of linear connection and the data blocks inside any cluster have their own independent data density, while the density of data points in each data block is uniform, making our model discriminate the structures of linear connection and objective clusters effectively. Then local clustering with parameter adaptation is performed to discover a collection of local clusters inside each data block, where the key parameters *eps* and *minPts* are dynamically updated according to the number of data points in each data block, making it less sensitive to initial values of parameters. To obtain final clustering results, global clustering is performed to merge local clusters. The merging process may involve different data blocks, where the data block merger is used to dynamically merge two blocks step by step. Overwhelmingly, by leveraging the idea of splitting and merging, Ada-DBSCAN reduces the computation of neighboring points and thus has higher efficiency. In addition, Ada-DBSCAN is greatly compatible to be integrated in distributed frameworks such as Hadoop and Spark.

Our main contributions can be summarized as follows:

- We propose Ada-DBSCAN, a novel adaptive density-based clustering algorithm which leverages the idea of data splitting and data merging to dynamically discover clusters from local to global, making it well-suited for addressing the issue of linear connection.
- We derive a simple but effective parameter adaptation mechanism for clustering, making Ada-DBSCAN applicable for data with heterogeneous density and less sensitive to initial values of parameters.
- Extensive experiments on both artificial and real-world datasets show that Ada-DBSCAN outperforms existing density-based clustering algorithms significantly. Moreover, Ada-DBSCAN shows significant lower computational cost compared with DBSCAN.

The remainder of this paper is organized as follows. Section II will provide a brief review of related works. Section III will elaborate details of the proposed Ada-DBSCAN algorithm. Section IV will describe our experimental setup and Section V will present experimental results and discussions about Ada-DBSCAN and other baselines. Section VI will conclude our work.

II. RELATED WORK

In recent years, the development of clustering algorithms has attracted great attention in both academic and industrial communities. In particular, density-based clustering algorithms have an advantage of requiring no prior knowledge about the number of clusters which needs to be set manually. Furthermore, density-based clustering algorithms are highly applicable to data with noise. As one of the most typical density-based clustering algorithms, Density-based Spatial Clustering of Applications with Noise (DBSCAN) was proposed in 1996 [6], in which the word ''density'' is defined as the number of data points within a specified radius (*eps*) range

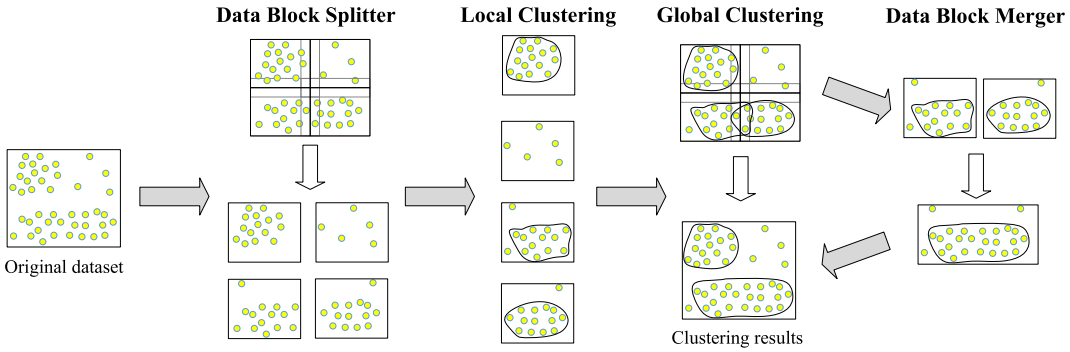


FIGURE 2. Overview of the proposed Ada-DBSCAN framework.

in multi-dimensional space. In DBSCAN, all data points are divided into three categories including core points, reachable points and outliers. The core points refer to the data points which contain no less than specific number of neighbors (*minPts*) within a radius range of *eps*. Moreover, the points within the radius range of *eps* are defined as the directly reachable points, while the points that are connected to the directly reachable points by at least one path are defined as the reachable points. The data points outside the radius range of *eps* centered at any core points are called the outliers.

Briefly, DBSCAN contains several crucial steps. First, DBSCAN randomly selects an unvisited core point associated with its directly reachable points, forming an initial cluster. Then DBSCAN iteratively selects a core point in this cluster until all core points are visited, which aims to aggregate all its directly reachable points that appear for the first time. DBSCAN repeats the above steps until all clusters are determined.

However, DBSCAN shows inferior performance when data points are heterogeneous, which is a drawback that it is hard for DBSCAN to be widely-used in many data analysis scenarios. To address the problem, the Enhanced DBSCAN algorithm [16] was proposed to calculate the density variance for all core points, with the help of the parameter *eps* to keep track of local density variation. Then the algorithm expands a core point if its density variance is smaller than a threshold, which is determined by homogeneity test. Authors of DSets-DBSCAN [17] introduced the concept of ''DSets'' by applying histogram equalization to the pairwise similarity matrix of input data, which further improves DBSCAN since the parameters of DBSCAN can be determined by DSets. In addition, RNN-DBSCAN [18] proposed to use statistical results of reverse nearest neighbors to solve the issue of heterogeneous density. RECORD [19], IS-DBSCAN [20], and ISB-DBSCAN [21] are other similar studies based on the reverse nearest neighbors to improve the performance of DBSCAN.

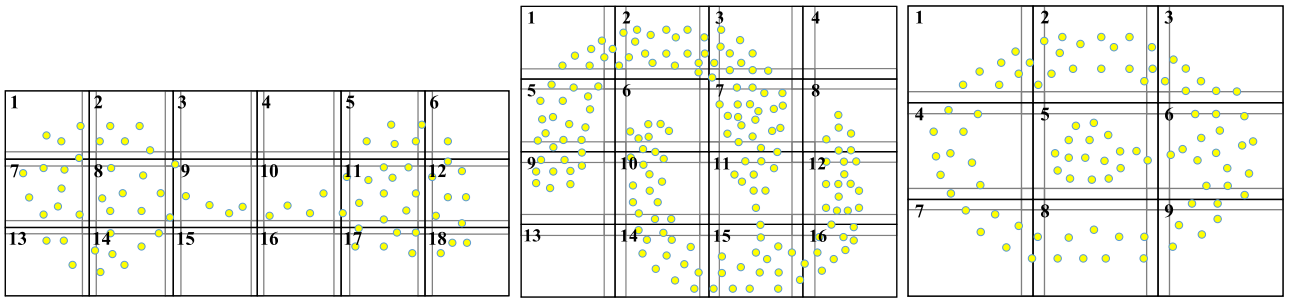
With DBSCAN employed in practical applications, the low efficiency has been a bottleneck. Thus, various DBSCAN-extended algorithms have been proposed to enhance the performance of DBSCAN. GF-DBSCAN [22] was proposed to project data points into a grid, limiting the search space of neighboring points to its adjacent cells. To reduce the time cost, G-DBSCAN [23] was proposed to speed up the procedure of searching neighboring points, which is scalable and applicable for high-dimensional datasets. Besides, in order to implement fast query of nearest neighbors, ISB-DBSCAN [21] was proposed to improve the traditional locality sensitive hashing approach. By utilizing graphics processing units (GPUs), CudaSCAN [24] is another method to boost efficiency of clustering from the hardware perspective [25]. In addition, some extended versions of DBSCAN with supporting of distributed and parallel computing [26]–[30] were implemented to apply for massive data analysis.

III. ADA-DBSCAN

In this section, we propose Adaptive Density-based Spatial Clustering of Applications with Noise (Ada-DBSCAN) which focuses on solving the issue of linear connection existed in DBSCAN and other density-based clustering algorithms, meanwhile improving the parameter setting and efficiency of DBSCAN when applied into massive data. As shown in Fig. 2, Ada-DBSCAN consists of four major modules, including: (1) data block splitter, which allocates data points to a set of data blocks hierarchically based on the core idea of uniform data distribution. (2) local clustering, which is a local density-based clustering process in each data block. (3) global clustering, which is a merging process for local clusters to obtain final clustering results. (4) data block merger, which is used for merging two data blocks during global clustering. Ada-DBSCAN first performs a top-down process, which adopts the data block splitter to split data points into a set of data blocks, and then local clustering is performed inside each data block. To obtain final clustering results, Ada-DBSCAN performs a bottom-up global clustering process, where the data block merger serves as a key sub-component. Details of each proposed module in Ada-DBSCAN are elaborated as below.

A. DATA BLOCK SPLITTER

Different from previous algorithms, we propose a novel data block splitter by taking different circumstances of linear connection into consideration. As shown in Fig. 3, we divide



(a) External connection (b) Concave connection (c) Internal connection

FIGURE 3. Demonstration of the data block splitter for different circumstances of linear connection problem.

linear connection into three circumstances: external connection, concave connection and internal connection. For example, the data points in the 9-th block and the 10-th block in Fig. 3(a) are easily to be viewed as reachable points in conventional density-based clustering algorithms since they seem to be "connected", causing inferior cluster discovery. Another case of linear connection is shown in Fig. 3 (b), two sickle-shaped clusters are "connected" in the 11-th block and the 15-th block, which is even harder to distinguish by both density-based or distance-based clustering methods. For the internal connection in Fig. 3 (c), data points in the 5-th block and the 6-th block also seem to be "connected", where data points in the 5-th block are completely surrounded by external data points. Generally, the difficulty to distinguish these individual clusters comes from analysis for high-dimensional datasets.

The data block splitter aims to automatically identify appropriate data blocks, with each data block under uniform data distribution. Here, the uniform data distribution refers that the data blocks containing the region of linear connection and the data blocks inside any cluster have their own independent data density, while the density of data points in each data block is uniform. Thus, the associated data blocks can be separated by borderlines. When a sub-block is splitted from a particular block, skew distribution of data could be avoided. Concretely, we elaborate the mechanism of the data block splitter as follows.

Inspired by the theory of information entropy [31], the data block splitter ensures the splitting with uniform distribution by measuring the average rate of information on each data block. Formally, the information entropy is defined as:

$$H(X) = \sum P(x)I(x) = -\sum P(x)\log_b P(x) \tag{1}$$

where x is a random variable belonging to dataset X, $P(x)$ is the occurrence probability of x, and b is the base of information entropy and we set $b = 2$ in this paper. Information entropy can be considered as a method to measure the uniformity of data distribution. High information entropy means that the data points are disordered, while low information entropy means that the data points are more concentrated or more ordered. For multi-dimensional data, the data block splitter first determines that data points should be splitted into blocks in which dimension. Concretely, given a collection of data $\{x_m^{(1)}, x_m^{(2)}, \cdots, x_m^{(N)}\}$, where N is the total number of data points and m is the dimensionality of given data. We first divide the data into N sub-ranges equally in the i-th ($i \in [1, m]$) dimension, where each sub-range covers a certain number of data points. Empirically, we can obtain a probability sequence $\{p_i^{(1)}, p_i^{(2)}, \cdots, p_i^{(N)}\}$ represented by the frequency of each sub-range, which is the proportion of data points that each sub-range covers. Thus, the information entropy of the i-th dimension can be calculated by Equation (1). Note that the data block splitter selects to split data points into blocks in a specific number of dimension, with the criteria that the information entropy in these dimension are the largest ones. It is because data with lower information entropy is generally in ordered distribution, which makes it more effective for clustering.

Suppose the data block splitter splits data in the i-th dimension, the initial number of data blocks ρ is given by:

$$\rho = \left\lceil \sqrt[\lambda]{\frac{N}{e}} \right\rceil \tag{2}$$

where e denotes the expected average number of data points in a data block, λ is a hyper-parameter representing the number of dimensions to be splitted. Note that λ is restricted to be fixed as 3 when the dimensionality of the dataset is larger than 3, since a higher value of λ makes ρ tend to be 1. Therefore, $\lambda \leq 3$. Furthermore, for the i-th dimension, the bias of the data block splitter (denoted as ω_i) is given by:

$$\omega_i = \frac{\mu_i}{\rho_i} = \frac{\max_i - \min_i}{\rho_i} \tag{3}$$

where $\max_i$, $\min_i$ denotes the maximum and the minimum value of data points in the i-th dimension, μ_i is the difference between $\max_i$ and $\min_i$, ρ_i is the number of data blocks in the i-th dimension. Note that in Fig. 3, the gray dotted lines located near the boundaries between two blocks are called "extended boundary", which is because a data point is possible to be located at the boundary between two or more blocks and may have directly reachable points within a radium range of *eps*. For a specific data block, to satisfy that its left adjacent block and right adjacent block can simultaneously expand

their boundary towards the middle with a distance of eps, the bias ω_i is required to have a minimum threshold of $2eps$. In particular, if $\omega_i < 2eps$, ρ_i will be formed as $\left\lceil \frac{\mu_i}{2eps} \right\rceil$. When splitting the last block, the bias ω' should be $\mu_i - 2(\rho_i - 1)eps$, if it is smaller than eps, the splitting should be early stopped and ρ_i is formed as $\left\lfloor \frac{\mu_i}{2eps} \right\rfloor$. Therefore, the final number of data blocks in the i-th dimension ρ_i' is defined as:

$$\rho_i' = \begin{cases} \left\lceil \sqrt[\lambda]{\frac{N}{e}} \right\rceil & \text{if } \omega_i \geq 2eps \\ \left\lceil \frac{\mu_i}{2eps} \right\rceil & \text{if } \omega_i < 2eps \text{ and } \omega_i' \geq eps \\ \left\lfloor \frac{\mu_i}{2eps} \right\rfloor & \text{if } \omega_i < 2eps \text{ and } \omega_i' < eps \end{cases} \tag{4}$$

Similarly, the final expected average number of data points in a data block is denoted as e', given by:

$$e' = \frac{\omega_i e}{\mu_i/\rho} = \begin{cases} e & \text{if } \omega_i \geq 2eps \\ \frac{\omega_i e \rho}{\mu_i} & \text{if } \omega_i < 2eps \end{cases} \tag{5}$$

Therefore, each data block can be determined by the lower bound and the upper bound in standard coordinate system, formed as:

$$g_l(k_i) = min_i + \omega_i k_i \tag{6}$$

$$g_u(k_i) = min_i + (\omega_i + 1)k_i \tag{7}$$

where k_i is the index of the data block, $g_l(k_i)$, $g_u(k_i)$ denotes the lower bound and the upper bound of the k-th block respectively. In addition, for data points in the "extended boundary", the data block splitter performs a "boundary expansion" operation by keeping a backup of data points inside their adjacent blocks and recording their previous block indexes. Then the scope of the k-th data block in the i-th dimension is represented as $\delta(k_i)$, which is given by:

$$\delta(k_i) = \begin{cases} [g_l(k_i), g_u(k_i) + eps] & \text{if } k_i = 0 \\ [g_l(k_i) - eps, g_u(k_i) + eps] & \text{if } 0 < k_i < \rho_i - 1 \\ [g_l(k_i) - eps, g_u(k_i)] & \text{if } k_i = \rho_i - 1 \end{cases} \tag{8}$$

Subsequently, sub-blocks are splitted from their parent blocks. However, too few data points in a specific block may cause block splitting unnecessary with higher time cost. To make our data block splitter more effective and more efficient, we propose a criterion that block splitting from a parent block to a certain number of sub-blocks should be stopped when the number of data points in the parent block is less than $0.5e$, otherwise the block splitting should be continued. Hence, given the initial index of data block as 0, the index of the k-th sub-block is given by:

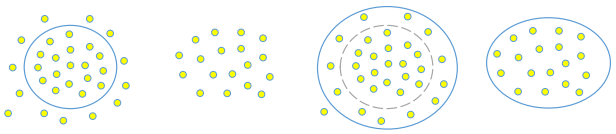
$$\varepsilon_k = \rho \cdot \varepsilon_{father} + k + 1 \tag{9}$$

where ε_{father} is the index of the parent data block. Finally, all splitted blocks are recorded by a queue, which is highly suitable for the block splitting process with an obvious feature of first-in-first-out (FIFO). In summary, the main procedure of the data block splitter is described in Algorithm 1.

Algorithm 1 Data Block Splitter

Input: a dataset D, a radius hyper-parameter eps
Output: a set of data blocks in a queue Q

```
MAX, MIN ← select maximum and minimum values in each dimension of D
data block b ← D.toBlock()
b.setFirstBolock()
Q ← initialize an empty queue
while b.isNotLastPartitionedDimension() do
    if b.isFirstBlock() then
        b.changeNewDimension()
        calculate ω_i, ρ_i and using Eq. (3) and Eq. (4)
    end if
    if b.size > 0.5e then
        BLOCKS ← partitionData(b) using Eq. (8)
        BLOCKS.calculateIndexs() using Eq. (9)
        if b.isFirstBlock() then
            BLOCKS[0].setFirstBlock()
        end if
        Q.enqueue(BLOCKS)
    else
        Q.enqueue(b)
    end if
    b ← Q.dequeue()
end while
return Q
```



(a) DBSCAN with small eps (b) DBSCAN with large eps

FIGURE 4. Influence of the radius range *eps* in DBSCAN.

B. LOCAL CLUSTERING

Local clustering is a density-based clustering process inside each individual data clock. Compared with DBSCAN, our local clustering strengthens the ability of parameter adaptation. As shown in Fig. 4, the radius range eps in DBSCAN may affect the performance of clustering when data points are heterogeneous. Fig. 4 (a) depicts that DBSCAN with small eps may only discover dense clusters (left side) while ignore spare clusters (right side). However, Fig. 4 (b) shows that DBSCAN with large eps may obtain clusters but with noise (left side) though both dense and sparse clusters can be identified. Ada-DBSCAN performs adaptive local clustering first, since the influence of heterogeneous density can be ignored when each data block is splitted appropriately enough, making clusters with different densities discriminative.

Concretely, given the query index of the b-th data block, the dimension index in the i-th dimension is denoted as $q(b_i)$, and the last data block in the i-th dimension is denoted as $p(i)$. The scope is determined by the lower bound and the upper bound in standard coordinate system, formed as:

$$f_l(b_i) = min_i + \omega_i q(b_i) \tag{10}$$

$$f_u(b_i) = min_i + \omega_i (q(b_i) + 1) \tag{11}$$

where ω_i is the bias of the data block splitter, $f_l(b_i)$ and $f_u(b_i)$ denote the lower bound and the upper bound respectively. Then the scope of the data block $h(b_i)$ is given by:

$$h(b_i) = \begin{cases} [f_l(b_i), f_u(b_i)) & \text{if } q(b_i) < p(i) \\ [f_l(b_i), f_u(b_i)] & \text{if } q(b_i) = p(i) \end{cases} \tag{12}$$

Thus, the number of the data points within the block scope is represented as:

$$\varphi = \left| \left\{ x | \forall x_i \in x, x_i \in h(b_i) \right\} \right| \tag{13}$$

where x is the vector of a data point in the b-th data block with different dimensions, x_i is the i-th dimension of the x.

In local clustering, given the initial parameters *eps* and *minPts*, where *eps* denotes the threshold of radius range, *minPts* denotes the threshold of minimum number of points in the neighborhood of radius *eps*. We dynamically update the two parameters as:

$$eps_{new} = \frac{e}{\varphi} \times eps \tag{14}$$

$$minPts_{new} = \left\lceil \frac{\varphi}{e} \times minPts \right\rceil \tag{15}$$

where e denotes the average number of data points after splitting in a data block. The updated parameters guarantee that small *eps* along with large *minPts* are utilized for dense clusters, and large *eps* along with small *minPts* are applied in opposite situations. In summary, the procedure of local clustering is described in Algorithm 2.

Algorithm 2 Local Clustering

Input: a data block B, a radius parameter eps_{new}, a threshold $minPts_{new}$

Output: a set of local clusters S in the block B

```
1:  B.setUnvisited()
2:  S ← initialize an empty set
3:  Q ← initialize an empty queue
4:  for each unvisited point p in B do
5:    Neighbors ← p.getNeighbors(eps_new)
6:    if Neighbors.size() > minPts_new then
7:      a new cluster C contains p
8:      Q.enqueue(Neighbors)
9:      while Q.isNotEmpty() do
10:       p' ← Q.dequeue()
11:       if p'.isNotVisited() then
12:         if p'.isNotMemberOfAnyCluster() then
13:           C.add(p')
14:         end if
15:         Neighbors' ← p'.getNeighbors(eps_new)
16:         if Neighbors'.size() > minPts_new then
17:           Q.addNewAppearance(Neighbors')
18:         end if
19:         p'.setVisited()
20:       end if
21:     end while
22:     S.add(C)
23:   end if
24:   p.setVisited()
25: end for
```

C. GLOBAL CLUSTERING

Given data blocks splitted by data block splitter, where each data block contains a set of clusters by local clustering, Ada-DBSCAN further performs global clustering to obtain final clustering results. Since all local clusters are contained in a certain number of data blocks, the global clustering is actually a merging process for relevant local clusters as well as relevant data blocks.

Consider that all data blocks are splitted from their ''father'' blocks with an unique index each. The relation between the index of a ''father'' block and its ''child'' block is given by:

$$\varepsilon_{father} = \left\lfloor \frac{\varepsilon_{child}}{\rho} \right\rfloor \tag{16}$$

where ρ is the number of data blocks, ε_{child} and ε_{father} denote indexes of the child block and the father block, respectively. According to Equation (16), a data block should be merged to its adjacent data blocks if they have same ε_{father}, and the merged block should be reindexed as ε_{father} until it has no correlation with its adjacent data blocks simultaneously.

Briefly, for global clustering, we first put all data blocks containing local clusters into a queue Q. Then, data blocks are dequeued from Q one by one. If the current data block is related to the previous block, a merging operation between the two blocks will be performed, which is covered by our data block merger. Otherwise, we may reindex the previous dequeued block according to Equation (16), and then enqueue the reindexed block. The dequeue and enqueue operations above are repeated until all data blocks merged into one data block, where it covers global clusters. In summary, the procedure of global clustering is described in Algorithm 3. Note that the merging operation covered in line 8 and line 9 will be demonstrated in the following section.

D. DATA BLOCK MERGER

When merging two data blocks, it is not a simple pattern to collect all clusters scattered in different blocks because some clusters may cover several data blocks, as shown in Fig. 3. Thus, the most crucial step for data block merger is to judge the correlation of local clusters contained in different blocks. As an example illustrated in Fig. 5 (a), both A3 and A4 are

Algorithm 3 Global Clustering

Input: data blocks containing local clusters in queue Q
Output: a set of global clusters C

```
data block b1 ← Q.dequeue()
Level ← b1.Level
while Q.isNotEmpty() do
    data block b2 ← Q.dequeue()
    if b2.isFirstBlock() then
        Level ← b2.Level        // Level is the splitted depth
    end if
    if b1.isRelated(b2) then
        b1 ← DataBlockMerger(b1, b2)
    else
        if b1.Level >= Level then
            b1.Level ← b1.Level − 1
            b1.calcluateNewIndex() using Eq. (16)
        end if
        Q.enqueue(b1)
        b1 ← b2
    end if
end while
C ← all clusters in b1
return C
```

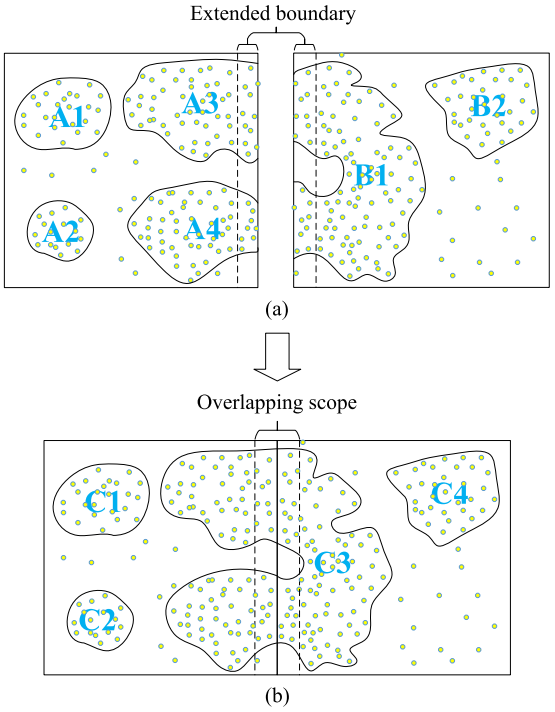


FIGURE 5. An example of two data blocks which need merging.

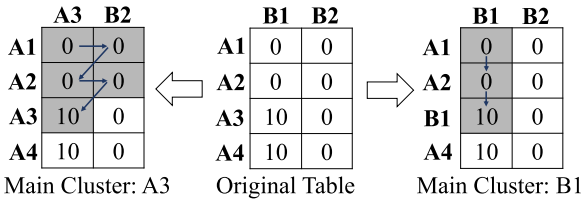


FIGURE 6. An example about the updating of table T.

correlated to B1, which is measured by the overlapping scope between dotted lines and boundaries. The data block merger then merges the two related data blocks, forming a new cluster C3 shown in Fig. 5 (b). In addition, merging local clusters located on both sides of blocks may compose an U-shape "parent" cluster or its extended types. As shown in Fig. 5, C3 is an U-shape "parent" cluster, which is because the correlation between local clusters A3 and A4 can be confirmed by their adjacent data block since the local cluster B1 has the same pattern in "extended boundary". Thus, the data block merger should record the connections of all local clusters.

Concretely, the data block merger employs a table T to record the connections of all local clusters contained in the two data blocks, where each value entry denotes connection degree between different local clusters which is counted by the number of interconnections between core points inside the local clusters. The number of connections between two merged local clusters should be larger than a minimum threshold, denoted as β, which is fixed to *minPts*. It is because the data points between these merged local clusters can be guaranteed with strong correlation based on core points to avoid occurring linear connection. If a local cluster has larger than β connections with other local clusters according to T, then it should be selected as the main cluster, and data points in other relevant local clusters will be aggregated to the main cluster. Moreover, the table header corresponding to the merged clusters in the table T is updated as the identifier of the main cluster.

For the merging of data blocks shown in Fig. 5, the corresponding updating of the table T is depicted in Fig. 6. Suppose $\beta = 3$, and connections between different local clusters is recorded in the original table T shown in Fig. 6, where each value entry denotes connection degree. For example, A3 and B1 is "connected" with a connection degree of 10. If we take a strategy of horizontal traverse for table T, data points in B1 are aggregated to A3 and the header of B1 in corresponding column of T will be updated into A3, which means A3 is elected as the main cluster. Otherwise, if we traverse the table T vertically, data points in A3 are aggregated to B1 and the header of A3 in corresponding row of T will be updated into B1, which indicates B1 is elected as the main cluster. In summary, the main procedure of the data block merger is described in Algorithm 4.

IV. EXPERIMENTAL SETUP

A. DATASETS

We evaluate Ada-DBSCAN and other clustering methods on two types of datasets: artificial datasets and real-world datasets. Artificial datasets are generated by the Utility of Scikit-Learn [32] which is a widely-used sample generator. We generate artificial datasets by considering the dimensionality (i.e., the number of features) to be 2 and 3.

Algorithm 4 Data Block Merger

Input: data block $b1$, data block $b2$, a hyper-parameter β
Output: a merged data block b

```
map ← toMap(b1.copiedData)
list ← toList(b2.copiedData)
T ← initialize an empty table
for each point in list do
    for each neighbor in point.neighborlist do
        if map.isExist(neighbor) then
            T.addConnection(point, neighbor)
        end if
    end for
end for
for each connection c in T do
    if c.connectCount > β then
        c.cluster1 ← mergeClusters(c.cluster1, c.cluster2)
        T.modifyHeader(c.cluster1, c.cluster2)
    end if
end for
if b2.isNotEmpty() then
    b1.appendCluster(b2)
end if
b ← b1
return b
```

TABLE 1. Summary of artificial datasets.

Dataset	Instances	Features	Clusters
Normal-1	5,000	2	6
Normal-2	5,000	3	6
Heterogeneous	2,000	2	4
External	2,500	2	3
Concave	2,500	2	2
Internal	2,000	2	2

Finally, the generated artificial datasets are divided into four categories: (1) Normal datasets, composed of **Normal-1** and **Normal-2** datasets, where data points are both simply distributed but with different number of features. (2) **Heterogeneous** dataset, where the density of data points is heterogeneous with noise. (3) **External** dataset, where data points are distributed with external connection which is shown in Fig. 3 (a). (4) **Concave** dataset, where data points are distributed with concave connection which is shown in Fig. 3 (b). (5) **Internal** dataset, where data points are distributed with internal connection which is shown in Fig. 3 (c). The statistics of artificial datasets are summarized in Table 1.

We also evaluate different clustering algorithms using real-world datasets with predefined labels from UCI Machine Learning Repository [33]. We select several released datasets[1] for clustering, including: (1) Iris dataset (denoted as **IR**), which contains 3 classes of 50 instances each, where each class refers to a type of Iris plant. (2) Ecoli dataset (denoted as **EC**), which contains a total of 336 instances

[1]https://archive.ics.uci.edu/ml/index.php

TABLE 2. Summary of real-world datasets.

Dataset	Instances	Features	Clusters
IR	150	4	3
EC	336	7	8
ZO	101	16	7
BA	1,372	4	2
WIL	2,000	7	4
TAE	151	5	3
SS-1	50,000	3	2
SS-2	100,000	3	2
SS-3	150,000	3	2
SS-4	200,000	3	2

with 8 attributes. (3) Zoo dataset (denoted as **ZO**), which is a simple database containing information about 7 types of animals, where each type of animal has 16 attributes. (4) Banknote Authentication dataset (denoted as **BA**), which contains varieties of wavelet transformed images with 1,372 instances in total. (5) Wireless Indoor Localization dataset (denoted as **WIL**), which contains 2,000 instances about signal strengths of 7 WiFi signals visible on a smartphone. (6) Teaching Assistant Evaluation dataset (denoted as **TAE**), which consists of 3 roughly equal-sized categories about evaluation of teaching performance of 151 teaching assistant assignments. (7) Skin Segmentation dataset, which contains over 245K images with red (R), green (G) and blue (B) values from various groups of people. For better comparison, we randomly extract 50,000 instances (denoted as **SS-1**), 100,000 instances (denoted as **SS-2**), 150,000 instances (denoted as **SS-3**) and 200,000 instances (denoted as **SS-4**) from Skin Segmentation dataset, respectively. In summary, the statistics of real-world datasets are described in Table 2.

B. BASELINE METHODS

We compare our model with several strong baseline algorithms for clustering, which are all implemented using C++ programming language:

- *DBSCAN:* the most typical density-based spatial clustering algorithm with fundamental concepts about density [6].
- *OPTICS:* an optimized density-based clustering algorithm which improves the parameter setting of DBSCAN by ordering data points to identify the clustering structure [7].
- *HDBSCAN*[2]*:* a DBSCAN-based algorithm which automatically selects the parameters of DBSCAN in hierarchical manner [34].
- *AnyDBC*[3]*:* an efficient density-based clustering algorithm which reduces both the range query and the label propagation time of DBSCAN [35].
- *SDC:* a scalable clustering algorithm which partly investigates the linear connection problem in density-based clustering algorithms [15].

[2]https://github.com/ojmakhura/hdbscan.
[3]https://github.com/VigneshN1997/AnyDBC_C_Code.

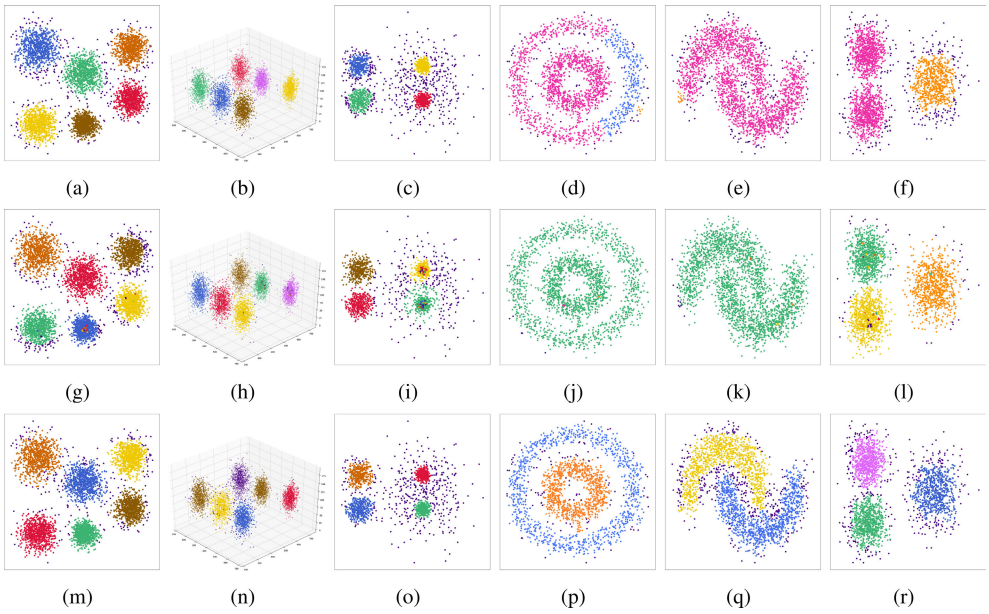


FIGURE 7. Qualitative results on artificial datasets, where sub-figures (a) - (f) are results of DBSCAN, (g) - (l) are results of SDC, and (m) - (r) are results of Ada-DBSCAN. Note that columns 1 - 6 correspond to Normal-1, Normal-2, Heterogeneous, Internal, Concave and External datasets, respectively.

C. IMPLEMENTATION DETAILS

We implement the proposed Ada-DBSCAN using C++ programming language. For fair comparison, we conduct all experiments in the same machine with 2 Intel Xeon 2.13GHz L5630 processors (totally 8 cores and 16 threads) and 32 GB PC3-10600 ECC Reg RAM, where the operating system is Ubuntu 16.04 LTS. For Ada-DBSCAN and all other baselines, the parameters *minPts* and *eps* are dynamically tuned to achieve best performance on artificial datasets. We vary the parameter *minPts* with initial values of $\{3, 4, 5\}$ to evaluate Ada-DBSCAN and all other baselines on real-world datasets, which is a common parameter in density-based clustering algorithms. Besides, other model-specific parameters of baseline methods are initialized with original default values and tuned to achieve best performance.

To evaluate the performance of each clustering algorithm in real-world datasets, we adopt average accuracy as the evaluation metric, which is given by:

$$accuracy = \frac{\sum_{j=1}^{|C|} |\alpha_j|}{N} \tag{17}$$

where $|C|$ is the total number of the clusters, j is the index of a particular cluster, $|\alpha_j|$ is the number of correctly matched data points in cluster C_j, and N is the total number of data points.

V. EXPERIMENTAL RESULTS

A. ANALYSIS OF LINEAR CONNECTION

As a key issue discussed in this paper, linear connection problem is widely existed in DBSCAN and other density-based clustering algorithms. In all baseline methods, SDC is the particular one that explores the linear connection problem by taking the advantage of distance-based clustering. Thus, we compare the performances of SDC and our model from both qualitative and quantitative perspectives. Fig. 7 depicts qualitative results of Ada-DBSCAN compared with SDC and DBSCAN on artificial datasets. We first analyse qualitative results point by point as follows.

I. The results of the first two columns in Fig. 7 report the performances of DBSCAN, SDC and Ada-DBSCAN on Normal-1 and Normal-2 datasets. We can observe that Ada-DBSCAN and DBSCAN achieve approximate results when data points are distributed with uniform density. However, the clusters obtained by SDC are inferior, as some points belonging to objective clusters are treated as noise. SDC tends to obtain sparse clusters, which is because SDC first searches all centers of the objective clusters, causing a certain number of centers being obtained in a dense cluster sometimes, but the final cluster is formed by only one center.

II. The performances of different models on Heterogeneous dataset are depicted in the third column in Fig. 7. As shown in Fig. 7 (c), DBSCAN with fixed parameters *eps* and *minPts* is challenging to obtain satisfactory clusters, since it ignores necessary data points which should be contained in some sparse clusters. However, if the parameters *eps* and *minPts* are adjusted to satisfy requirements of sparse clusters, we have conducted experiments and verified that it causes dense clusters containing noise data points. For SDC, the results shown in Fig. 7 (i) demonstrates that SDC still suffers the drawback that a few internal points belonging to objective clusters are treated as noise. In contrast, our Ada-DBSCAN ultimately discovers 4 clearly distinguished clusters as shown in Fig. 7 (o), which is because Ada-DBSCAN discriminates the density of

different data blocks to adjust the parameters *eps* and *minPts* dynamically.

III. The results of the fourth column in Fig. 7 report the performances of DBSCAN, SDC and Ada-DBSCAN on Internal dataset. When internal connection occurs, DBSCAN breaks the structure of one cluster shown in Fig. 7 (d) instead of getting all clusters correctly. SDC confuses two individual clusters as illustrated in Fig. 7 (j), indicating that SDC cannot discover such clusters with internal connection. It is because that SDC runs clustering by searching a center first and carrying out a concentric expansion. Different from DBSCAN and SDC, the result of Fig. 7 (p) shows that Ada-DBSCAN performs well when suffering the situation of linear connection and obtains two objective clusters, indicating that our proposed data block splitter and merger along with two-level clustering is effective.

IV. As shown in the fifth column in Fig. 7, the performances of DBSCAN and SDC are inferior to Ada-DBSCAN on Concave dataset. When concave connection occurs, DBSCAN discovers a noisy cluster with noise data points contained at the left-side tail, which can be observed from Fig. 7 (e). We have further verified that applying stricter restriction of parameters accelerates the breakdown of objective clusters. Fig. 7 (k) depicts that SDC still cannot handle this variety of linear connection due to its limitation of concentric expansion. However, Ada-DBSCAN is able to distinguish the difference between two objective clusters and thus obtain much better clustering results, as illustrated in Fig. 7 (q).

V. The last column in Fig. 7 shows the performances of DBSCAN, SDC and Ada-DBSCAN on External dataset. As can be seen from Fig. 7 (f), DBSCAN cannot distinguish two different clusters located at left-side, whose drawback is greatly conquered by SDC as illustrated in Fig. 7 (l). However, Fig. 7 (r) demonstrates Ada-DBSCAN further improves the clustering performance due to the fact that it eliminates noise compared with SDC.

To further verify the superiority of Ada-DBSCAN compared with SDC, we conduct extensive experiments on three real-world datasets (BA, ZO, IR). As shown in Fig. 8, Ada-DBSCAN outperforms SDC significantly on BA dataset, with an increase of about 29% accuracy. On behalf of ZO and IR datasets, Ada-DBSCAN achieves higher accuracy results than SDC with an improvement from 7% to 10%. In conclusion, both qualitative results and quantitative results demonstrate Ada-DBSCAN has better superiority over SDC.

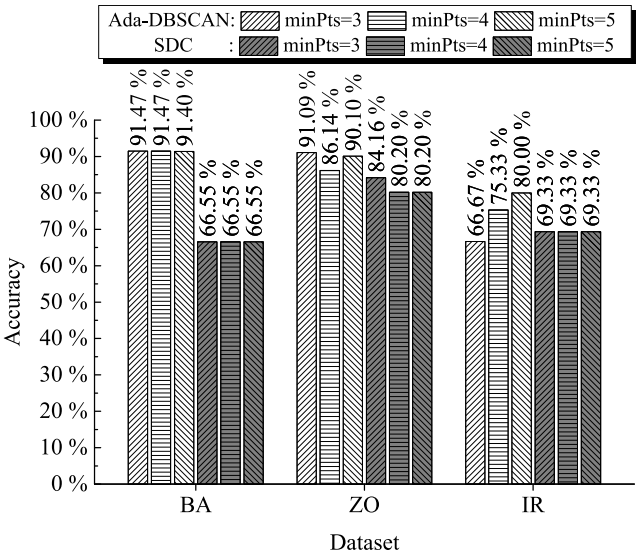


FIGURE 8. Performance comparison between Ada-DBSCAN and SDC.

TABLE 3. Quantitative results on real-world datasets.

Dataset	minPts	DBSCAN	OPTICS	HDBSCAN	AnyDBC	Ada-DBSCAN
IR	3	66.67%	58.67%	50.00%	65.33%	**66.67%**
	4	74.00%	72.67%	66.67%	64.67%	**75.33%**
	5	69.33%	76.67%	66.67%	66.67%	**80.00%**
EC	3	52.80%	**58.33%**	44.64%	48.21%	52.98%
	4	57.14%	57.14%	44.35%	54.17%	**58.04%**
	5	49.11%	57.44%	44.35%	50.60%	**59.52%**
WIL	3	54.55%	47.90%	37.65%	44.75%	**54.55%**
	4	58.10%	**59.55%**	27.90%	38.10%	58.10%
	5	54.50%	52.55%	27.80%	30.85%	**61.50%**
ZO	3	91.09%	49.50%	40.59%	85.15%	**91.09%**
	4	86.14%	67.33%	69.31%	81.19%	**86.14%**
	5	83.17%	71.29%	65.35%	79.21%	**90.10%**
BA	3	91.47%	72.52%	14.80%	60.93%	**91.47%**
	4	91.47%	73.25%	20.19%	75.73%	**91.47%**
	5	**91.47%**	77.55%	17.49%	81.56%	91.40%
TAE	3	47.02%	23.18%	12.58%	43.05%	**47.02%**
	4	47.02%	27.15%	17.22%	47.02%	**47.02%**
	5	47.02%	33.77%	21.85%	47.02%	**47.02%**

B. QUANTITATIVE RESULTS

We further compare Ada-DBSCAN and other density-based clustering algorithms on several real-world datasets. The quantitative results of Ada-DBSCAN and other baselines are illustrated in Table 3, where the best results are highlighted in boldface.

As shown in Table 3, Ada-DBSCAN achieves significant improvement over baseline methods on IR dataset. Meanwhile, the performance of Ada-DBSCAN rises with the initial value of the parameter *minPts* increasing, which may be because higher threshold of minimum number of points in the neighborhood of radius *eps* benefits the model with higher adaptivity for local clustering. In comparison, the clustering accuracy results of HDBSCAN and OPTICS are much lower when *minPts* is 3. For the results on EC and WIL datasets, HDBSCAN performs pretty inferior to other models, indicating that the hierarchical manner of clustering in HDBSCAN is limited in such scenarios. As a strong baseline, OPTICS achieves the highest clustering accuracy of 58.33% when $minPts = 3$ on EC dataset and 59.55% when $minPts = 4$ on WIL dataset. However, Ada-DBSCAN still achieves

TABLE 4. *T*-test between Ada-DBSCAN and other baseline algorithms.

	Ada-DBSCAN			
	DBSCAN	OPTICS	HDBSCAN	AnyDBC
t-value	1.96	2.81	3.22	3.47
corresponding p	0.0536	0.0188	0.0117	0.0089

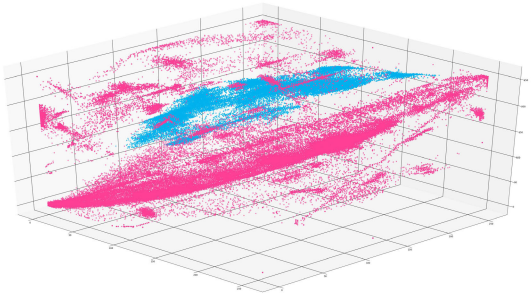


FIGURE 9. Data distribution of Skin Segmentation dataset.

an improvement of about 2% when *minPts* is selected in a larger value. Besides, compared with HDBSCAN and AnyDBC, Ada-DBSCAN shows significant superiority with an increase of about 10% on EC dataset and 20% on WIL dataset, respectively. Regarding to ZO, BA and TAE datasets, the performance of OPTICS extremely deteriorates while Ada-DBSCAN still retains good performance. As presented in Table 3, ZO, BA and TAE are three typical examples showing that several extended versions of DBSCAN may not be able to perform better than DBSCAN. However, Ada-DBSCAN achieves equal or higher clustering accuracy compared with DBSCAN, which also shows better stability and robustness on all the datasets with *minPts* varying.

We also conduct T-test between Ada-DBSCAN and other baseline algorithms to evaluate the significance of our quantitative results. According to T-test results presented in Table 4, Ada-DBSCAN significantly improves the performance compared to DBSCAN with corresponding $p < 0.1$, and significantly improves the performance compared to OPTICS and HDBSCAN with corresponding $p < 0.05$. Moreover, Ada-DBSCAN outperforms AnyDBC with corresponding $p < 0.01$. Overall, other advanced density-based algorithms may focus on certain aspects but is not applicable to general scenarios. Ada-DBSCAN retains the advantages of DBSCAN, making it well-suited for all scenarios that DBSCAN can handle. Therefore, Ada-DBSCAN has significant superiority over DBSCAN and its extended algorithms with high performance and good stability.

C. EFFICIENCY ANALYSIS

To evaluate the efficiency of our Ada-DBSCAN, we compare the time complexity between Ada-DBSCAN and DBSCAN on large-scale datasets. We conduct extensive experiments on four large-scale datasets SS-1, SS-2, SS-3 and SS-4, which are subsets sampled independently from Skin Segmentation dataset. The data distribution of original Skin Segmentation dataset is shown in Fig. 9, which is a typical instance that a cluster surrounded by another one. Note that other baselines including OPTICS, HDBSCAN and AnyDBC are not well-suited in such variety of dataset since their clustering performances are inferior as discussed above. It is the reason that we exclude them into efficiency comparison.

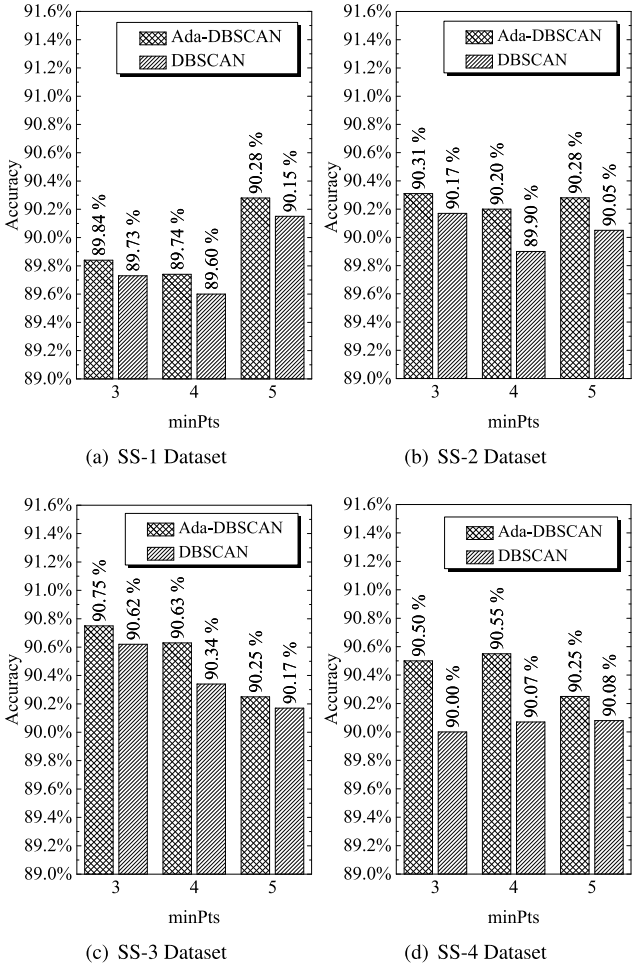


FIGURE 10. Performance comparison between Ada-DBSCAN and DBSCAN on large-scale datasets.

Preliminarily, we verify the effectiveness of Ada-DBSCAN on large-scale datasets. The quantitative results of Ada-DBSCAN and DBSCAN are illustrated in Fig. 10. Both Ada-DBSCAN and DBSCAN achieve over 89% clustering accuracy on the four datasets with different size. By comparing the optimal result with different *minPts* on a particular dataset and the optimal result on different datasets, it is verified that Ada-DBSCAN outperforms DBSCAN with a slight improvement of about 0.5% accuracy.

However, the time cost of Ada-DBSCAN is significantly lower than that of DBSCAN. Theoretically, the time complexity of DBSCAN is $O(N^2)$, while the time complexity of Ada-DBSCAN is $O(\frac{N}{e} \cdot e^2) = O(eN)$, where N denotes the size of the dataset, and e denotes the expected average number of data points in a data block.

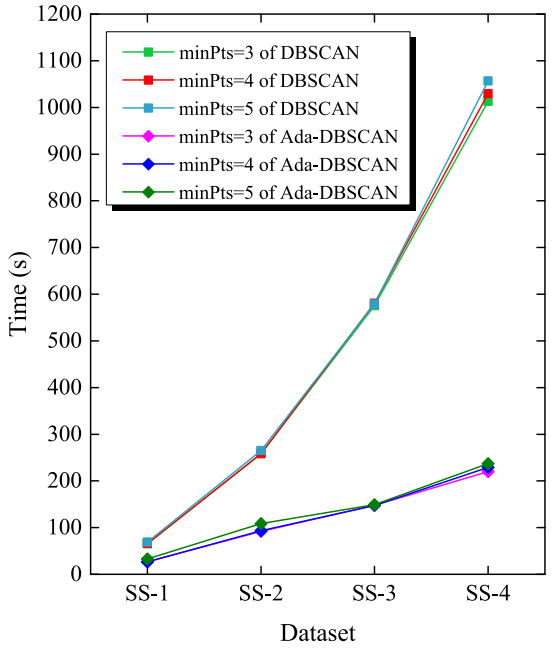


FIGURE 11. Comparison of time cost between Ada-DBSCAN and DBSCAN.

Note that e is specified according to the specific dataset and can not be omitted due to the calculation in "extended boundary", but usually $e \ll N$ when N is large. For fair comparison, we calculate the average running time for both algorithms using one CPU only. Fig. 11 depicts the results of time cost between Ada-DBSCAN and DBSCAN with the size of datasets increasing. The time cost of Ada-DBSCAN is close to DBSCAN when running on a small dataset SS-1, but obviously Ada-DBSCAN needs less time when enlarging the dataset. It is because that our data block splitter following with local clustering and global clustering is beneficial to speed up the clustering process, and the adaptive parameters also accelerate the convergence of our algorithm. In particular, DBSCAN needs over 1000s on SS-4 dataset while Ada-DBSCAN only needs about 200s to finish clustering, reducing about 80% time cost. Overall, Ada-DBSCAN shows significant superiority in efficiency compared with DBSCAN. In addition, by leveraging data block splitting and merging, Ada-DBSCAN can be well-adapted to distributed frameworks like MapReduce [36] for massive data analysis.

VI. CONCLUSION

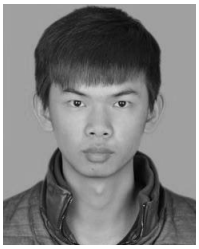
In this paper, we propose a novel adaptive density-based spatial clustering algorithm called Ada-DBSCAN, with the aim to address a key issue called linear connection that existing density-based clustering algorithms struggle to tackle. Ada-DBSCAN consists of a data block splitter and a data block merger, which are coordinated by local clustering and global clustering. In contrast to baseline algorithms, Ada-DBSCAN is able to conquer the linear connection effectively and set parameters adaptively. Meanwhile, Ada-DBSCAN achieves better clustering performance with lower computational cost. The extensive experiments on artificial datasets and real-world datasets demonstrate the effectiveness of our algorithm.

For future work, we plan to extend our algorithm to support distributed computing to fully explore the idea of data splitting and merging based on uniform data distribution, making it better-adapted to massive distributional data analysis. Another research direction is to exploit the ensemble clustering strategy to further improve our Ada-DBSCAN. Finally, we intend to apply our algorithm into more data mining tasks, such as event detection and trajectory prediction.

REFERENCES

[1] R. Sowmya and K. Suneetha, "Data mining with big data," in *Proc. 11th Int. Conf. Intell. Syst. Control (ISCO)*, 2017, pp. 246–250.
[2] C. Zhang, L. Liu, D. Lei, Q. Yuan, H. Zhuang, T. Hanratty, and J. Han, "TrioVecEvent: Embedding-based online local event detection in geo-tagged tweet streams," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 595–604.
[3] H. Li, Y. Ge, H. Zhu, and H. Zhu, "Point-of-interest recommendations: Learning potential check-ins from friends," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 975–984.
[4] J. Xiong, J. Ren, L. Chen, Z. Yao, M. Lin, D. Wu, and B. Niu, "Enhancing privacy and availability for data clustering in intelligent electrical service of IoT," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1530–1540, Apr. 2019.
[5] T. Tian, "Detecting particle clusters in particle-fluid systems by a density based method," *Commun. Comput. Phys.*, vol. 26, no. 5, pp. 1617–1630, Jun. 2019.
[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. Int. Conf. Knowl. Discovery Data Mining*, 1996, vol. 96, no. 34, pp. 226–231.
[7] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," *ACM SIGMOD Rec.*, vol. 28, no. 2, pp. 49–60, 1999.
[8] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probab.*, Oakland, CA, USA, 1967, vol. 1, no. 14, pp. 281–297.
[9] X. Li, V. Ceikute, C. S. Jensen, and K.-L. Tan, "Effective online group discovery in trajectory databases," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2752–2766, Dec. 2013.
[10] Z. Xiao, L. Ponnambalam, X. Fu, and W. Zhang, "Maritime traffic probabilistic forecasting based on vessels' waterway patterns and motion behaviors," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 11, pp. 3122–3134, Apr. 2017.
[11] S. Zhou, Y. Deng, R. Wang, N. Li, and Q. Si, "Effective mapping of urban areas using ENVISAT ASAR, sentinel-1A, and HJ-1-C data," *IEEE Geosci. Remote Sens. Lett.*, vol. 14, no. 6, pp. 891–895, Jun. 2017.
[12] A. Hinneburg and D. A. Keim, "An efficient approach to clustering in large multimedia databases with noise," in *Proc. Int. Conf. Knowl. Discovery Data Mining*, vol. 98, 1998, pp. 58–65.
[13] E. Biçici and D. Yuret, "Locally scaled density based clustering," in *Proc. Int. Conf. Adapt. Natural Comput. Algorithms*. Berlin, Germany: Springer, 2007, pp. 739–748.
[14] J. Gan and Y. Tao, "Fast Euclidean optics with bounded precision in low dimensional space," in *Proc. Int. Conf. Manage. Data*, 2018, pp. 1067–1082.
[15] C. C. Yang and T. D. Ng, "Analyzing and visualizing Web opinion development and social interactions with density-based clustering," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 6, pp. 1144–1155, Nov. 2011.
[16] A. Ram, A. Sharma, A. S. Jalal, A. Agrawal, and R. Singh, "An enhanced density based spatial clustering of applications with noise," in *Proc. Adv. Comput. Conf. (IACC)*, 2009, pp. 1475–1478.
[17] J. Hou, H. Gao, and X. Li, "DSets-DBSCAN: A parameter-free clustering algorithm," *IEEE Trans. Image Process.*, vol. 25, no. 7, pp. 3182–3193, Jul. 2016.
[18] A. Bryant and K. Cios, "RNN-DBSCAN: A density-based clustering algorithm using reverse nearest neighbor density estimates," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 6, pp. 1109–1121, Jun. 2018.

[19] S. Vadapalli, S. R. Valluri, and K. Karlapalem, "A simple yet effective data clustering algorithm," in *Proc. 6th Int. Conf. Data Mining (ICDM)*, 2006, pp. 1108–1112.
[20] C. Cassisi, A. Ferro, R. Giugno, G. Pigola, and A. Pulvirenti, "Enhancing density-based clustering: Parameter reduction and outlier detection," *Inf. Syst.*, vol. 38, no. 3, pp. 317–330, May 2013.
[21] Y. Lv, T. Ma, M. Tang, J. Cao, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan, "An efficient and scalable density-based clustering algorithm for datasets with complex structures," *Neurocomputing*, vol. 171, pp. 9–22, Jan. 2016.
[22] C.-F. Tsai, C.-T. Wu, and S. Chen, "GF-DBSCAN: A new efficient and effective data clustering technique for large databases," in *Proc. WSEAS Int. Conf. Math. Comput. Sci. Eng.*, no. 9, 2009, pp. 1–6.
[23] K. M. Kumar and A. R. M. Reddy, "A fast DBSCAN clustering algorithm by accelerating neighbor searching using groups method," *Pattern Recognit.*, vol. 58, pp. 39–48, Oct. 2016.
[24] W.-K. Loh and H. Yu, "Fast density-based clustering through dataset partition using graphics processing units," *Inf. Sci.*, vol. 308, pp. 94–112, Jul. 2015.
[25] N. Scicluna and C.-S. Bouganis, "ARC 2014: A multidimensional FPGA-based parallel DBSCAN architecture," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 9, no. 1, 2015, Art. no. 2.
[26] A. Lulli, M. Dell'Amico, P. Michiardi, and L. Ricci, "NG-DBSCAN: Scalable density-based clustering for arbitrary data," *Proc. VLDB Endowment*, vol. 10, no. 3, pp. 157–168, 2016.
[27] M. A. Patwary, D. Palsetia, A. Agrawal, W.-K. Liao, F. Manne, and A. Choudhary, "A new scalable parallel DBSCAN algorithm using the disjoint-set data structure," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2012, p. 62.
[28] M. Gowanlock, D. M. Blair, and V. Pankratius, "Optimizing parallel clustering throughput in shared memory," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2595–2607, Sep. 2017.
[29] B.-R. Dai and I.-C. Lin, "Efficient map/reduce-based DBSCAN algorithm with optimized data partition," in *Proc. IEEE 5th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2012, pp. 59–66.
[30] Q. Tong, X. Li, and B. Yuan, "Efficient distributed clustering using boundary information," *Neurocomputing*, vol. 275, pp. 2355–2366, Jan. 2018.
[31] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul./Oct. 1948.
[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, É. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[33] D. Dheeru and E. K. Taniskidou. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml
[34] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining*. Berlin, Germany: Springer, 2013, pp. 160–172.
[35] S. T. Mai, I. Assent, and M. Storgaard, "AnyDBC: An efficient anytime density-based clustering algorithm for very large complex datasets," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1025–1034.
[36] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.



ZIHAO CAI received the B.S. degree from South China Normal University, in 2017. He is currently pursuing the M.S. degree with the School of Computer Science and Engineering, South China University of Technology. His current research interests include machine learning and data mining.

JIAN WANG received the B.S. degree in network engineering from the South China University of Technology, in 2017, where he is currently pursuing the M.S. degree with the School of Computer Science and Engineering. His current research interests include natural language processing and machine learning.

KEJING HE (Member, IEEE) received the Ph.D. degree in electronic engineering from the South China University of Technology, in 2007. He is currently a Professor with the South China University of Technology. He is the author of more than 40 journal articles and conference papers. His current research interests include big data technology and applications, high performance computing, and cloud computing. He is a member of IEEE-CS and ACM.