

Received January 22, 2020, accepted February 1, 2020, date of publication February 6, 2020, date of current version March 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2972034

# An Improved DBSCAN Algorithm Based on the Neighbor Similarity and Fast Nearest Neighbor Query

**SHAN-SHAN LI** 

Department of IT Development and Management, Huaqiao University, Xiamen 361021, China  
 Information Construction and Management Office, Huaqiao University, Xiamen 361021, China  
 Provincial Key Laboratory for Computer Information Processing Technology, Soochow University, Soochow 215006, China  
 e-mail: lishany03@hqu.edu.cn

This work was supported in part by the National Natural Science Foundation of China under Grant 51305142, Grant 51305143, and Grant 71571056, in part by the Project of Quanzhou Science and Technology Plan under Grant 2018C114R, and in part by the Open Project of Provincial Key Laboratory for Computer Information Processing Technology, Soochow University, China, under Grant KJS1839.

**ABSTRACT** DBSCAN is the most famous density based clustering algorithm which is one of the main clustering paradigms. However, there are many redundant distance computations among the process of DBSCAN clustering, due to brute force Range-Query used to retrieve neighbors for each point in DBSCAN, which yields high complexity ( $O(n^2)$ ) and low efficiency. Thus, it is unsuitable and not applicable for large scale data. In this paper, an improved DBSCAN based on neighbor similarity is proposed, which utilizes Cover Tree to retrieve neighbors for each point in parallel, and the triangle inequality to filter many unnecessary distance computations. From the experiments conducted on large scale data sets, it is demonstrated that the proposed algorithm greatly speedup the original DBSCAN, and outperform the main improvements of DBSCAN. Comparing with  $\rho$ -approximate DBSCAN, which is the current fastest but approximate version of DBSCAN, the proposed algorithm has two advantages: one is faster and the other is that the result is accurate.

**INDEX TERMS** Clustering, DBSCAN, neighbor similarity, cover tree.

## I. INTRODUCTION

Clustering analysis refers to the analysis process of grouping a set of physical or abstract objects into multiple classes composed of similar objects. Its goal is to classify the data according to the similarity of the internal characteristics of the data, and reveal the internal natural structure of the data. In short, clustering refers to grouping abstract objects or physical object sets, so that the similarity of objects in a group is large, while the difference between different groups is large. It is unnecessary to training data set, so that it is called a kind of unsupervised machine learning method. Hence, it has a more prominent advantage in the face of some unknown distribution data sets, due to that it is capable of classifying a large number of unknown data to reveal the connection and difference between different categories.

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei .

As an important research direction in data mining, many clustering algorithms have been widely used in various fields of data processing and application, and plays an indispensable role that is more and more important, such as Vehicle re-identification [1], Road Extraction [2], Time Series processing [3], image fusion [4] and image denoising [5] etc.

Currently, there are numerous of clustering algorithms, among which density based clustering is one of the most popular algorithms, such as DPeak [6]–[9] algorithm and DBSCAN [10] algorithm. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm is a classical and one of the most important density based clustering algorithm. It can not only find noise points and outliers, but also identify any shape distribution categories in the data, which has strong adaptability to different data. At present, DBSCAN algorithm has been applied to many fields such as image processing [11], consumer market analysis, crime detection, medicine and agriculture. For example, in the field of online advertising, by clustering users with DBSCAN

algorithm, businesses can customize targeted advertising sales strategies for different types of users, which can not only improve the user experience to a certain extent, but also greatly improve the conversion rate of corresponding advertising. Another example, is that DBSCAN can also be used in credit card application review, network crime, tax evasion and other fields by comparing ordinary user behavior data for abnormal user identification, because outliers can be figured out easily by DBSCAN.

Although DBSCAN algorithm has been widely used, there are still some shortcomings in the algorithm itself. The input parameters of DBSCAN algorithm include scanning radius  $\varepsilon$  and density threshold  $MinPts$ . In the process of clustering, it needs to calculate the density of each data point. According to the definition of density in DBSCAN algorithm, the density of a data point is related to the distance from this point to all other points. Therefore, DBSCAN algorithm has a large amount of calculations, and runs in  $O(n^2)$  expected time. In addition, due to the “curse of dimension” DBSCAN is not suitable for high dimensional data.

In this paper, we deeply analyze and study the clustering principle of DBSCAN, and find that the core problem in DBSCAN is to find nearest neighbors for each point, then we propose a fast density algorithm based on nearest neighbor similarity (by triangle inequality) and fast nearest neighbor search (by cover tree), which can effectively reduce the number of distance calculations in the clustering process, and improve the clustering efficiency.

## II. RELATED WORK

DBSCAN algorithm needs two parameters, one is scan radius  $\varepsilon$ , the other is  $MinPts$ , which is used to determine whether a point is the density threshold of the core point. As mentioned above, the algorithm has high complexity and time consumption of corresponding clustering. For low-dimensional data, DBSCAN algorithm can effectively cluster the data; however, with the increasing of data processing dimensions, the performance of DBSCAN algorithm in clustering drops sharply. Therefore, DBSCAN algorithm cannot show its advantages in dealing with large scale data.

In order to solve or alleviate the shortcomings of DBSCAN algorithm and improve the efficiency of dealing with large scale data, many researches have put forward strategies to improve the algorithm. At present, there are mainly two ways to improve.

The first method is to build index structure for data, such as k-d tree [12] (and improved k-d tree [13]), Cover Tree [14], Semi-Convex Hull Tree [15] and SS-tree [16]. This kind of method is more efficient when the data dimension is less than 20. The research of this method shows that the time complexity of DBSCAN algorithm with these index structures is  $O(n \log(n))$ . However, it turns out to be a false statement. Gunawan [17] points out that no matter how  $\varepsilon$  and  $MinPts$  are taken, the worst complexity of DBSCAN is actually  $O(n^2)$ . However, this erroneous claim is widely accepted in many research papers and textbooks, such as

literature [18]–[20], etc. In addition, because of the so-called “curse of dimension”, that is, DBSCAN has little effect on the high-dimensional data, when processing the high-dimensional data (for example, more than 20 dimensions), these index structures are difficult to achieve the purpose of acceleration, or even degenerate into linear search.

Wu *et al.* [21] proposed a linear DBSCAN algorithm based on LSH (Locality-Sensitivity Hashing). The principle of LSH is to map the original data points to the hash bucket by establishing multiple hash functions. The mapping process needs to meet that the original data with similar distance is still similar after mapping. LSH can effectively reduce the data dimension and perform the nearest neighbor query in linear time complexity. From the algorithm proposed by Wu *et al.*, we can analyze two parts: the first part is resume LSH index; the second part is to execute DBSCAN algorithm based on LSH index. The advantage of this algorithm is to reduce the time complexity to  $O(n \log(n))$ ; the disadvantage is to increase the parameters, and it is difficult to set the value of the parameters.

The second improvement is to partition the data set. There are two ways to improve the method of data set partition. One method is to divide large data sets into many smaller data sets, and then use DBSCAN algorithm to cluster small data sets. In the preprocessing stage, k-means algorithm is often used to divide the initial data set into some independent small parts. This method reduces the data size of each processing, controls the scale of the problem, occupies a small cache, but sacrifices a certain algorithm accuracy.

There are also methods to divide data space into small grids. This kind of method is to process the data in the small grid first, and then merge the processed results in the large grid, so as to improve the performance of the algorithm by reducing the query time. Mahrand and Mahar [22] introduced an algorithm called GriDBSCAN, which improves the performance of DBSCAN by using grid division and merging, thus producing high performance with high parallelism. However, because the influence of redundancy in this algorithm increases exponentially with the dimension, this technique is not suitable for high-dimensional data. Similarly, Gunawan [17] proposed a Fast-DBSCAN fast algorithm to improve DBSCAN algorithm in two-dimensional data space. The algorithm creates a virtual grid  $T$  in two-dimensional space, in which each cell of  $T$  has a side length of  $\varepsilon/\sqrt{2}$ . If a non-empty cell contains at least  $MinPts$  points, the cell is called a core cell; moreover, because the maximum distance in the cell is  $\varepsilon$ , all points in the cell are core points, so it is not necessary to calculate the density of each point in the core cell. Based on fast DBSCAN algorithm, Gan and Tao proposed  $\rho$ -approximate DBSCAN [23] algorithm. The algorithm achieves an excellent complexity of  $O(n)$  in low dimension. However, if the dimension  $D$  is very large, such that  $d > \log(n)$ , then the algorithm degenerates to be an  $O(n^2)$  algorithm. Chen *et al.* proposed KNN-BLOCK DBSCAN [24] and NQDBSCAN [25], which are both simple but fast, based on two points, the first one is that the key problem

DBSCAN is a kNN problem, and the second is that a point has a similar type its neighbors.

In addition to the above two improved methods, Lulli *et al.* [26] proposed an extensible DBSCAN algorithm; Han *et al.* [27] accelerated DBSCAN by using spark parallel computing technology; Kumar and Reddy [28] proposed an improved DBSCAN algorithm based on group. These works use various techniques, which are different from the proposed algorithm, to pursue high performance of DBSCAN.

### III. DBSCAN AND COVER TREE REVISITED

#### A. DBSCAN

In DBSCAN, there are some important concepts and terms, where some of terms are used in the previous paragraphs, such as core point, directly density-reachable, density reachable, density-connected and border point, and then cluster and noise are defined according to these terms as below.

**Definition 1** [10]: The  $\varepsilon$ -neighborhood of a point  $p$ , denoted by  $N_\varepsilon(p)$ , is defined by  $N_\varepsilon(p) = \{q | q \in P, d_{p,q} \leq \varepsilon\}$ , where  $P$  is a set of points and  $d_{p,q}$  is a distance function e.g. Euclidian distance, between  $p$  and  $q$ .

**Definition 2** [10]: The density of a point  $p$ , denoted by  $|N_\varepsilon(p)|$ , is the total number of points within  $N_\varepsilon(p)$ .

**Definition 3** [10]: A point  $p$  is a core point if  $|N_\varepsilon(p)| \geq \text{MinPts}$ , where  $\text{MinPts}$  is the density threshold.

**Definition 4** [10]: A point  $p$  is directly density-reachable from a point  $q$  with respect to  $\varepsilon$  and  $\text{MinPts}$  if  $p \in N_\varepsilon(q)$  and  $q$  is a core point.

**Definition 5** [10]: A point  $p$  is a border point if  $p$  is directly density-reachable from a core point  $q$  and  $|N(p)| < \text{MinPts}$ .

**Definition 6** [10]: A point  $p$  is density-reachable from a point  $q$  with respect to  $\varepsilon$  and  $\text{MinPts}$  if there is a chain of points  $p_1, p_2, \dots, p_n$ , with  $p_1 = q$  and  $p_n = p$  such that  $p_{i+1}$  is directly density-reachable from  $p_i$ .

**Definition 7** [10]: A point  $p$  is density-connected to a point  $q$  with respect to  $\varepsilon$  and  $\text{MinPts}$  if there is a point  $o$  such that both  $p$  and  $q$  are density-reachable from  $o$ .

**Definition 8** [10]: A point  $p$  is a noise if it is neither a core point nor a border point. This implies that noise does not belong to any clusters.

**Definition 9** [10]: A cluster  $C$  with respect to  $\varepsilon$  and  $\text{MinPts}$  is a non-empty subset of  $P$  satisfying the following conditions:

(1)  $\forall p, q : \text{if } p \in C \text{ and } q \text{ is density-reachable from } p \text{ with respect to } \varepsilon \text{ and } \text{MinPts}, \text{ then } q \in C$ . (*Maximality*)

(2)  $\forall p, q \in C : p \text{ is density-connected to } q \text{ with respect to } \varepsilon \text{ and } \text{MinPts}$ . (*Connectivity*)

#### B. COVER TREE

Cover Tree [14] is an efficient nearest neighbor search algorithm, which is based on a special hierarchical tree. The data structure requires  $O(n)$  space regardless of the metric's structure yet maintains all performance properties of a navigating net, and can be constructed in  $O(c^6 n \log n)$  time. Furthermore, nearest neighbor queries require time about  $O(c^{12} \log n)$  time.

There are three important properties in the cover tree data structure as below:

(1) Nesting: if  $c_i$  is the  $i^{\text{th}}$  layer, then  $c_i \in c_{i-1}$ ;

(2) Coverage:  $\forall p \in c_{i-1}$  has  $\exists q \in c_i$  to  $d_{p,q} \leq 2^i$ , and at least one parent node with  $q$  as  $p$ ;

(3) Separation:  $\forall p, q \in c_{i-1}$  we have  $d_{p,q} > 2^i$ .

The brief process of cover tree is: Firstly, the algorithm build a hierarchical covering tree for the source data set in the offline state. Secondly, for a batch of queries, the algorithm also establishes a query hierarchy tree for them. Thirdly, based on the tree properties, cover tree launches parallel searching algorithm through the two trees build above.

### IV. THE PROPOSED ALGORITHM

According to the advantages and disadvantages of various improved DBSCAN algorithms mentioned above, in this paper, we propose a new version of DBSCAN, it is based on nearest neighbor similarity and fast parallel nearest neighbor search, where the former reduces larger amounts of redundant distance computation, and the later speedup the process of searching neighbors for each point by parallel range query technique, i.e., by cover tree.

First of all,  $P$  is the set containing  $N$  d-dimensional data points, i.e.  $P \subset \mathbb{R}^D$ ;  $p_i \in P$  is the  $i^{\text{th}}$  point;  $d_{i,j}$  represents a certain distance from  $p_i$  to  $p_j$  (e.g. Euclidean distance, Hamming distance, for the convenience of explaining that the distance mentioned below refers to Euclidean distance);  $r$  means scanning radius, and  $N_r(p_i)$  is the  $r$ -neighborhood of data point  $p_i$ .

**Definition 10:** The density of  $p_i$  in  $r$ -neighborhood is defined as:

$$\rho_{i,r} = \sum_{k=1}^N \chi(d_{i,k} \leq r) \quad (1)$$

where  $\chi(x) = \begin{cases} 1, & x \text{ is true} \\ 0, & \text{else.} \end{cases}$

This definition means that the density of a point is the total number of neighbors within its  $r$ -neighborhood.

#### A. PROPERTIES OF TRIANGULAR INEQUALITY

Similar to KNN-BLOCK DBSCAN [24], we have an opinion that a point should have similar density with its neighbors, where triangular inequality may determine the type of that point in some case. Hence, in the proposed algorithm, we randomly select a data point  $p_k$  which is used as a reference point to filter some unnecessary distance computations for other points. The detail is shown as below.

Firstly, we calculate the distances from  $p_k$  to all other points and sort the distances, then establish two vectors of  $p_k$ :

(1) Indexing vector:  $p_{k,(1)}, p_{k,(2)}, \dots, p_{k,(n-1)}$ , where  $p_{k,(i)}$  is the index of the  $i^{\text{th}}$  nearest neighbor of  $p_k$ .

(2) Distance vector:  $d_{p_{k,(1)}}, d_{p_{k,(2)}}, \dots, d_{p_{k,(n-1)}}$ , where  $d_{k,(i)}$  is the distance from  $p_k$  to its  $i^{\text{th}}$  nearest neighbor.

Taking Euclidean distance into consideration, we have the following theorems and properties according to triangular inequality.

**Theorem 1:** Let  $p_i, p_j, p_k$  be three points in the data space, then we have  $|d_{i,k} - d_{j,k}| \leq d_{i,j} \leq d_{i,k} + d_{j,k}$ .

**Proof:** (1) Right side of the inequality:  $d_{i,j} \leq d_{i,k} + d_{j,k}$ , obviously holds.

(2) Left side of inequality: according to the principle that the sum of two sides of a triangle is greater than the third side, we have  $d_{i,k} \leq d_{i,j} + d_{j,k}$  and  $d_{j,k} \leq d_{i,j} + d_{i,k}$ , then  $d_{i,k} - d_{j,k} \leq d_{i,j}$  and  $d_{j,k} - d_{i,k} \leq d_{i,j}$ , hence it is obtained that  $|d_{i,k} - d_{j,k}| \leq d_{i,j}$ .

The inequality  $|d_{i,k} - d_{j,k}| \leq d_{i,j}$  in Theorem 1 reveals a very simple phenomenon, that is: Let points  $p_i$  be close to point  $p_j$ , then (1) the distance from  $p_j$  to  $p_k$  should be small if  $p_i$  is close to  $p_k$ ; (2) on the contrary, if  $p_i$  is far away from  $p_k$ ,  $p_j$  should be far away from  $p_k$ , too. The following simple properties can be derived from this theorem:

**Property 1:** If  $|d_{i,k} - d_{j,k}| > r$ , then  $d_{i,j} > r$ , that is,  $p_i$  and  $p_j$  are outside the  $r$ -neighborhood of  $p_j$  and  $p_i$ , respectively.

**Property 2:** If  $d_{j,k} - r \leq d_{i,k} \leq d_{j,k} + r$ , then  $p_i$  may be in the  $r$ -neighborhood of  $p_j$ .

**Property 3:** If  $d_{i,k} + d_{j,k} \leq r$ , then  $d_{i,j} \leq r$ , that is,  $p_i$  and  $p_j$  are within the  $r$ -neighborhood of  $p_j$  and  $p_i$ , respectively.

Let  $[\varepsilon, MinPts]$  be the two parameters of DBSCAN algorithm, then according to Theorem 1 and properties mentioned above, there are some theorems as below.

**Theorem 2:** (1) if  $d_{p,(MinPts)} \leq \varepsilon$ , then  $p$  is the core point; (2) if  $d_{p,(i)} > \varepsilon$ , then  $p$  is the non-core point, where  $i \leq MinPts$ .

**Proof:** (1) Because  $d_{p,(MinPts)} \leq \varepsilon$ , we have  $d_{p,(1)} \leq d_{p,(2)} \leq \dots \leq d_{p,(MinPts)} \leq \varepsilon$ ,  $|N_\varepsilon(p)| \geq MinPts$ , then according to the Definition 3,  $p$  is a core point.

(2) Because  $i \leq MinPts$  and  $d_{p,(i)} > \varepsilon$ ,  $\varepsilon < d_{p,(i)} \leq d_{p,(MinPts)}$ , hence  $|N_\varepsilon(p)| < MinPts$ , according to the Definition 3,  $p$  is a non-core point.

**Theorem 3:** Given a point  $p \in P$ , if  $|N_{2\varepsilon}(p)| < MinPts$ , then  $\forall q \in N_\varepsilon(p)$ ,  $q$  is a non-core point.

**Proof:** Because  $|N_{2\varepsilon}(p)| < MinPts$  and  $N_\varepsilon(p) \subseteq N_{2\varepsilon}(p)$ ,  $|N_\varepsilon(p)| < |N_{2\varepsilon}(p)| < MinPts$ , then according to the triangle inequality we have:  $\forall q \in N_\varepsilon(p)$  such that  $N_\varepsilon(q) \subseteq N_{2\varepsilon}(p)$ , and then  $|N_\varepsilon(q)| < |N_{2\varepsilon}(p)| < MinPts$ . Hence,  $\forall q \in N_\varepsilon(p)$ ,  $q$  is non-core point.

From Theorem 3, it is inferred that if  $|N_{2\varepsilon}(p)| < MinPts$  holds, then all points within  $N_\varepsilon(p)$  are all non-core points. Therefore, it is quite useful to identify a large number of non-core points.

## B. ALGORITHM DESCRIPTION

Based on the triangle inequality and cover tree mentioned above, in this paper we improve the DBSCAN algorithm, as shown in algorithm 1, the detail descriptions are list as below.

**Step 1 (Tree Building):** it establishes a hierarchical covering tree,  $T_1$ , for the source data set;

**Step 2 (Initialization):** initializes unprocessed data, set the neighbors of each point as *NULL*, and set the label for each data point as “-2” which means unlabeled or unprocessed.

**Step 3 (Querying):** The algorithm selects *seedsnum* data points (for example, 1000 or more) from the unprocessed data,  $UD$ , as queries, and build a query tree,  $TQ$ , for them; then, the cover tree is used to parallel retrieve the nearest neighbors of each query point; and then, according to Theorem 3 a part of outliers can be identified directly, and by Properties 1, 2 and 3, some core points are also found that is unnecessary to search in the global data. Repeat the above process until the types (core, outlier, and border) of all points identified.

**Step 4 (Merging Core Into a Cluster):** in this step, each core point  $p$  will be merged into other core points which is density-reachable from  $p$ , and forms different clusters.

**Step 5 (Assigning Borders):** For each border point, it will be assigned to a nearest cluster.

## C. COMPLEXITY ANALYSIS

As we can see from Algorithm 1, during the process of clustering, the main operation of the proposed algorithm is to retrieve the neighbors of query points.

There are two ways to query in the proposed algorithm: one is to use cover tree to do CoverTree::RangeQuery, and the other is to search only within the local neighborhood  $N_{2\varepsilon}(p)$  or  $N_\varepsilon(p)$ . Therefore, the complexity mainly consists of two parts:

(1) Step 3.3: If  $n \gg 2^d$  holds, the average time complexity for CoverTree::RangeQuery is  $O(\log n)$  [14]. Therefore, the complexity of using CoverTree to perform neighbor searching for the whole dataset is less than  $O\left(\frac{n}{SeedsNum} \log n\right)$ .

(2) Step 3.4: all queries are performed within  $N_\varepsilon(p)$  and  $N_{2\varepsilon}(p)$ , in which the number of data points is closely (linearly) related to  $MinPts$ . Hence, the complexity is about  $O(\alpha \times MinPts^2)$ , in which  $\alpha < n$  is the execution times.

Comprehensively, the total complexity is less than  $O\left(n \times \left[\frac{1}{SeedsNum} \log n + MinPts^2\right]\right)$ .

## V. EXPERIMENTAL SIMULATION

### A. ALGORITHMS AND MACHINE SET UP

In order to test the effect of different algorithm parameters on different dimension datasets, the original DBSCAN algorithm and  $\rho$ -approximate DBSCAN algorithm are selected and compared with the improved algorithm proposed in this paper.

All the experimental machines are configured as follows: the CPU model is Intel (R) core (TM) i5-4590 3.30 GHz, and the memory is 8G. They are all running with MATLAB code under Windows 10 64 bit operating system.

### B. DATA SETS

All data sets used in the following experiments come from UCI (<http://archive.ics.uci.edu/ml/index.php>). Some toy data sets are used to test the correctness of the proposed algorithm.

**Algorithm 1** ImprovedDBSCAN

---

**Input:** data set  $D$ , radius  $\varepsilon$ , density threshold  $MinPts$ ,  $seedsnum$       // $seedsnum$  is the number of points that are used as seeds  
**Output:** category labels  $Lables$

1 Build a hierarchical covering tree  $T1$  for dataset  $D$ ;

2 Initialization:

  2.1 Set unprocessed data set  $UD = D$

  2.2 Set the neighbor set of each point as  $TempN_\varepsilon(p) := \{\}$

  2.3  $Lables := \{-2, -2, \dots, -2\}$

3 While  $UD$  is not NULL

  3.1  $Seeds :=$  select  $seedsnum$  data points from  $UD$  as seeds ;

  3.2 Bulid query tree  $TQ$  for  $Seeds$

  3.3 Use CoverTree:: RangeQuery ( $T1, TQ, 2\varepsilon$ ) to parallel retrieve  $2\varepsilon$ -neighbors for each seed point in  $Seeds$

  3.4 For each data point  $p$  in seeds

    3.4.1 If  $|N_{2\varepsilon}(p)| < MinPts$ ,

      (1)  $Label(N_\varepsilon(p)) = -1$    //Set the label of each point in  $N_\varepsilon(p)$  to be “-1” which means it is an outlier (Theorem 3)

      (2)  $UD := UD - N_\varepsilon(p)$

    End If

    3.4.2 If  $|N_\varepsilon(p)| \geq MinPts$

      (1) Mark  $p$  as a core point,  $UD := UD - \{p\}$

      (2)  $\forall s \in N_\varepsilon(p)$ , find all  $\varepsilon$ -nearest neighbors of  $s$  within  $N_\varepsilon(p)$  according to Property 1-3, and add them into  $TempN_\varepsilon(s)$

      (3) If  $|TempN_\varepsilon(s)| > MinPts$ , then

        Mark  $s$  as a core point

$UD := UD - \{s\}$

      End If

    End If

  End For

End While

4. While there is a core  $p$  unlabeled      //The label of core point  $p$  is less than 0

  4.1 current  $Tmlabel := \text{Max} (Label) + 1$

  4.2  $aCluster = \{\text{all core points that are density-reachable from } p\} \cup \{p\}$

  4.3  $Label(aCluster) = Tmlabel$

End While

5. For each non-outlier and non-core point  $p$

  5.1  $aCore =$  Find the nearest core point within  $N_\varepsilon(p)$

  5.2  $Label(p) = Label(aCore)$  // assigned to the category of the core point

End For

**End**

---

Furthermore, a total of 5 large scale real datasets are applied to compare the speed of our method with  $\rho$ - approximate DBSCAN and original DBSCAN.

These datasets includes *PAM4D*(4D), *HOUSE* (7D), *REACTION(HAND POSTURE,* 28D), *MOCAP*(36D), *FONT(BODONI,* 256D). For each data set, all duplicate points are removed to make each point unique, all missing values are set to 0, and all data value are normalized to [0,100000]. The brief descriptions of the real data sets are list as below.

- (1) *PAM4D* is a 4-dimensional real dataset with cardinality  $n = 3,850,505$ ;
- (2) *HOUSE (Household)* is an another low dimensional real data set with dimension 7 and cardinality 2,049,280;
- (3) *MoCap* is a relative high dimensional data set with dimension 36, cardinality 65,536. It has 5 types of hand

postures from 12 users, the data were recorded by using unlabeled markers attached to fingers of a glove in a motion capture environment;

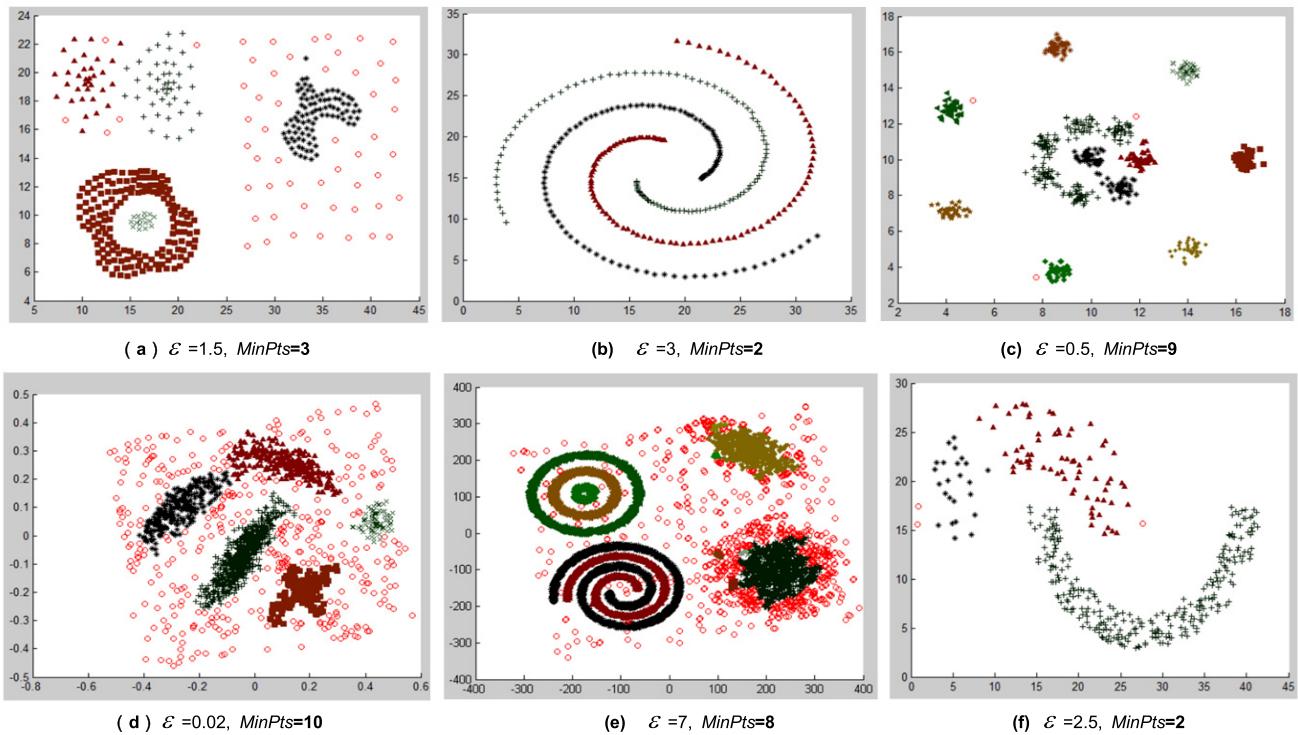
(4) *Font (CALIBRI)* is character font images data set with dim = 256, n = 19,068, which is made of images from 153470 character fonts (in this paper, we only use *CALIBRI*);

(5) *APS(APS Failure at Scania Trucks)* is a data set with dim = 170, n = 30,000, which consists of component failures for a specific component of the *APS* system.

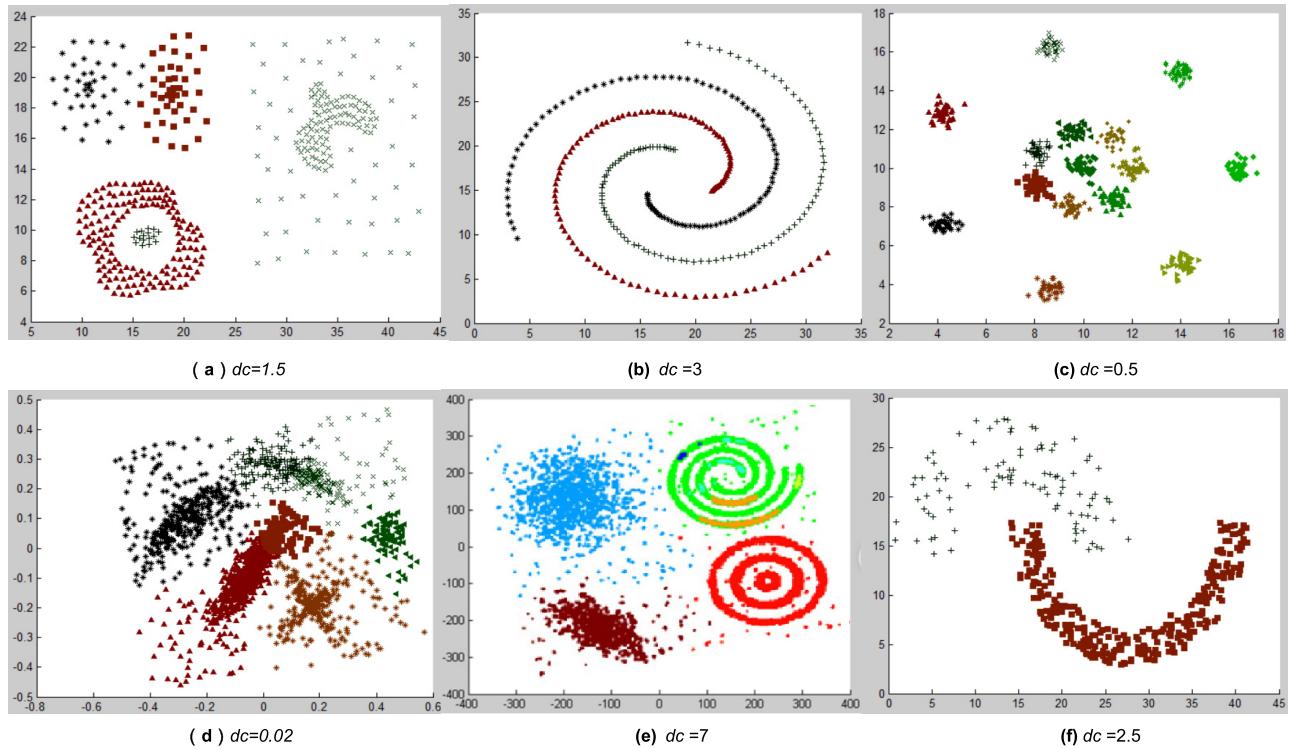
**C. EXPERIMENTS ON SOME TOY DATA SETS**

In this subsection, we demonstrate experiments on some toy data sets to show the clustering results as plotted in Figure 1, where red hollow circle are noise.

From Figure 1, we can see that the proposed algorithm works well on these data sets, and obtains good results as



**FIGURE 1.** The clustering results of the proposed algorithm on some experiments on toy data sets. Red hollow circles are noise identified by the proposed algorithm.



**FIGURE 2.** The clustering results of DPeak on some experiments on toy data sets, where  $dc$  is the scanning radius,  $\varepsilon$ , of DPeak.

original DBSCAN, i.e., the proposed algorithm can recognize the main shapes and clusters. For simplicity, maybe the parameter pair [ $\varepsilon$ ,  $MinPts$ ] is not be best, here only one clustering result is given for each data set.

Figure 2 shows the clustering results of DPeak on the same data sets. In these experiments, we do not identify noise for simplicity, and select 5, 3, 15, 5, 5 and 2 peaks, respectively. We can see that DPeak doesn't perform well

**TABLE 1.** Runtime comparison on different data sets (unit: second).

| [ $\varepsilon$ , MinPts] | The Proposed Method |                  | DBSCAN   | $\rho$ -approximate |
|---------------------------|---------------------|------------------|----------|---------------------|
|                           | Runtime (second)    | Noise identified | (second) | (second)            |
| REACTION                  | [3000,5]            | 21.64            | 452      | 264.32              |
| N=18,535                  | [5000,5]            | 20.65            | 336      | 243.15              |
| dim=28                    | [3000,50]           | 22.66            | 1378     | 304.57              |
|                           | [5000,50]           | 19.68            | 896      | 273.91              |
| MoCap                     | [1000,10]           | 211.83           | 1027     | 1203.23             |
| N=65,536                  | [5000,10]           | 146.03           | 276      | 1131.11             |
| dim=36                    | [1000,50]           | 267.34           | 2491     | 1467.22             |
|                           | [5000,50]           | 144.91           | 486      | 1334.01             |
| CALIBRI                   | [5000,10]           | 220.64           | 74       | 378.5               |
| N=3,964                   | [10000,10]          | 318.43           | 32       | 432.19              |
| dim=256                   | [5000,50]           | 351.35           | 126      | 543.25              |
|                           | [10000,50]          | 420.53           | 44       | 487.09              |
|                           |                     |                  |          | 622.12              |

for (e), on the other data sets, it works similarly to the proposed method.

#### D. EXPERIMENTS ON LARGE SCALE REAL DATA SETS

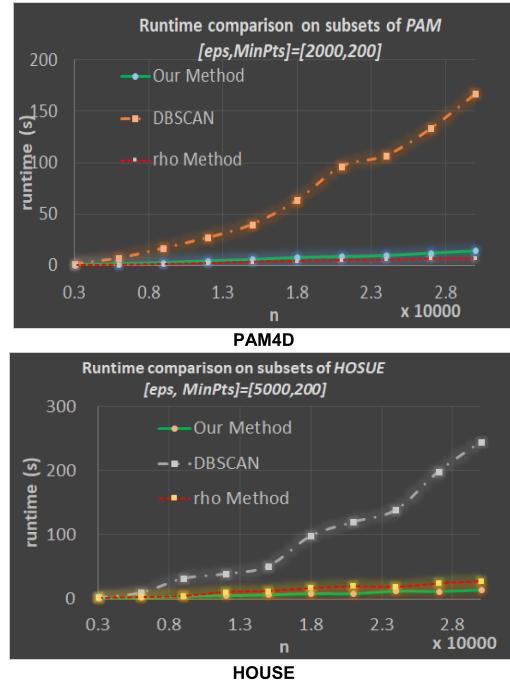
In this subsection, we conduct some experiments on some real large scale data sets, i.e., *PAM4D*, *HOUSE*, *MoCap*, *REACTION*etc.

Firstly, 10 subsets are extracted from *PAM4D* and *HOUSE*, and the cardinalities of each subset are 3000, 6000, ..., 30000, respectively. The two DBSCAN parameters are  $\varepsilon = 2000$ ,  $\text{MinPts} = 200$  on *PAM4D*, and  $\varepsilon = 5000$ ,  $\text{MinPts} = 200$  on *HOUSE*, respectively.

The runtime comparisons of the proposed algorithm with original DBSCAN and  $\rho$ -approximate DBSCAN algorithm are shown in Figure 3. It is observed that the proposed algorithm is similar to  $\rho$ -approximate DBSCAN in 4-dimensional data *PAM4D*, which indicates that it runs in a linear expected time. But in 7-dimensional data set *HOUSE*, the proposed algorithm outperforms  $\rho$ -approximate DBSCAN, while DBSCAN runs in  $O(n^2)$  clearly.

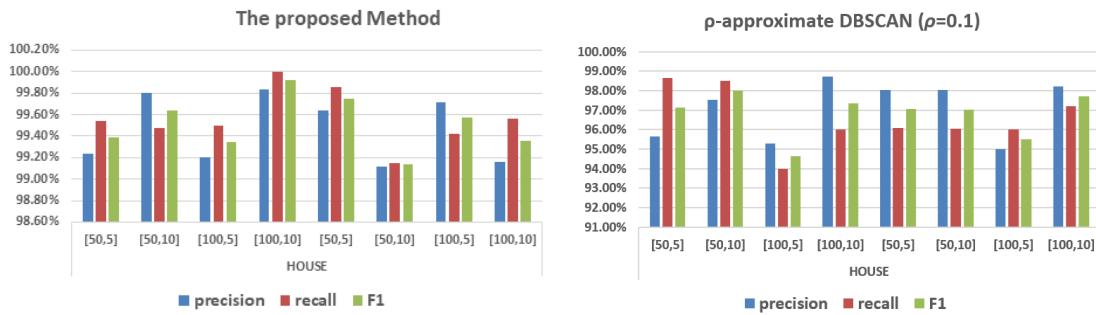
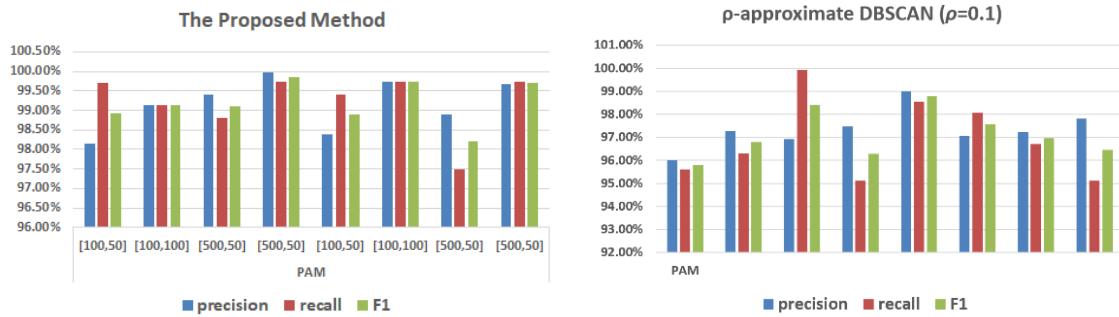
Secondly, in order to make more comparisons on some relative high-dimensional datasets, we also conduct experiments on *REACTION*, *MoCap* and *CALIBRI*.

The results are shown in Table 1, we can see that  $\rho$ -approximate DBSCAN algorithm degenerates to be an  $O(n^2)$  algorithm on these high-dimensional datasets, and it runs even slower than the original DBSCAN algorithm due to its redundant cost. However, the improved algorithm proposed runs much faster than original DBSCAN, which indicates that it has remarkable improvement and has better performance in high-dimensional data sets.

**FIGURE 3.** Runtime comparison on *PAM4D* and *HOUSE*.

#### E. ACCURACY TEST

The improved algorithm proposed in this paper is to speed up the original DBSCAN algorithm, so that it can deal with large-scale data. The precondition is that the clustering results should be consistent with, or as similar as possible to, the original DBSCAN algorithm under the same parameters, i.e., [ $\varepsilon$ ,  $\text{MinPts}$ ].

**FIGURE 4.** The comparison on subset of *House*.**FIGURE 5.** The comparison on subset of *PAM*.**TABLE 2.** Example: The evaluation of clustering result.

|                |   |   |   |   |   |   |   |   |
|----------------|---|---|---|---|---|---|---|---|
| $R_{ori}$      | 1 | 1 | 1 | 2 | 3 | 2 | 2 | 3 |
| $R_{new}$      | 4 | 2 | 4 | 4 | 5 | 5 | 2 | 5 |
| Correct or not | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |

In fact, the proposed algorithm in this paper uses Range-Query of CoverTreer to search the nearest neighbor accurately, so the algorithm in this paper is accurate. However, because DBSCAN is a non-determinant algorithm:

(1) The cluster assignments for a border point may vary depending on the order it appears. For example, suppose that point  $s$  is a non-core point,  $p$  and  $q$  are core points, and both  $p$  and  $q$  are not density-reachable from each other, but  $s$  is density-reachable from  $p$  and  $q$ . In this case,  $s$  can be assigned to either cluster of  $p$  or  $q$ .

(2) While for core points, the classification of DBSCAN is deterministic.

As we know that the clustering results of the two different clustering algorithms may use different label values, for example, clustering algorithm A represents a category by using “1”, while clustering algorithm B uses the same category by “2”. Therefore, it is necessary to match the clustering results of the two algorithms first, before comparing them.

In order to match the different clustering results obtained by different clustering algorithms, Chen *et al.* [29] used Hungarian algorithm [30] to maximize matching the clustering results of the proposed method and  $\rho$ -approximate DBSCAN

to that of the original DBSCAN, and then compute the accuracy, as follows:

Suppose there are 8 data points, the clustering label obtained by the original DBSCAN is  $R_{ori} = \{1, 1, 1, 2, 3, 2, 2, 3\}$ , and the clustering result of a new algorithm is  $R_{new} = \{4, 2, 4, 4, 5, 5, 2, 5\}$ . If the matching pairs retrieved from Hungarian algorithm are (1,4) (2,2) and (3,5), then the right and wrong judgment of the new algorithm is shown in Table 2, 4 results are correct, 4 errors, hence the accuracy is 50%. Due to the high complexity of the original DBSCAN algorithm, which cannot deal with large-scale data, we randomly select 10000 data points from *PAM* and *HOUSE* as test subsets. Figure 4 and Figure 5 show the accuracy comparison between the proposed algorithm and  $\rho$ -approximate DBSCAN on two subsets.

It can be seen that the proposed algorithm proposed in this paper is basically consistent with the original DBSCAN algorithm, and the difference is mainly caused by the different processing order of some boundary points.

## VI. CONCLUSION

The existing density based clustering method DBSCAN, because of redundant calculation, has too much complexity and cannot be used on large-scale data, which limits its application scope to a certain extent.

In this paper, we propose a DBSCAN algorithm based on nearest neighbor similarity and fast nearest neighbor search. By means of triangle inequality, neighbor similarity and fast neighbor search algorithm, the number of distance calculation in clustering process is greatly reduced, and the

efficiency of DBSCAN algorithm is effectively improved. Experiments show that compared with the original DBSCAN algorithm and its current fastest version  $\rho$ -Approximate DBSCAN algorithm, the algorithm is faster and more suitable for large-scale data sets.

Our future work is to apply other technique such Map-reduce or Spark to accelerate the proposed algorithm.

## REFERENCES

- [1] J. Zhu, H. Zeng, J. Huang, S. Liao, Z. Lei, C. Cai, and L. Zheng, "Vehicle re-identification using quadruple directional deep learning features," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 1, pp. 410–420, Jan. 2020.
- [2] Z. Chen, W. Fan, B. Zhong, J. Li, J. Du, and C. Wang, "Corse-to-fine road extraction based on local Dirichlet mixture models and multiscale-high-order deep learning," *IEEE Trans. Intell. Transp. Syst.*, to be published, doi: [10.1109/tits.2019.2939536](https://doi.org/10.1109/tits.2019.2939536).
- [3] S. Pei, T. Shen, X. Wang, C. Gu, Z. Ning, X. Ye, and N. Xiong, "3DACN: 3D augmented convolutional network for time series data," *Inf. Sci.*, vol. 513, pp. 17–29, Mar. 2020.
- [4] S. Liu, J. Wang, Y. Lu, H. Li, J. Zhao, and Z. Zhu, "Multi-focus image fusion based on adaptive dual-channel spiking cortical model in non-subsampled Shearlet domain," *IEEE Access*, vol. 7, pp. 56367–56388, 2019.
- [5] S. Liu, M. Liu, P. Li, J. Zhao, Z. Zhu, and X. Wang, "SAR image denoising via sparse representation in Shearlet domain based on continuous cycle spinning," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 5, pp. 2985–2992, May 2017, doi: [10.1109/tgrs.2017.2657602](https://doi.org/10.1109/tgrs.2017.2657602).
- [6] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, Jun. 2014.
- [7] Y. Chen, X. Hu, W. Fan, L. Shen, Z. Zhang, X. Liu, J. Du, H. Li, Y. Chen, and H. Li, "Fast density peak clustering for large scale data based on kNN," *Knowl.-Based Syst.*, vol. 187, Jan. 2020, Art. no. 104824.
- [8] D. Cheng, S. Zhang, and J. Huang, "Dense members of local cores-based density peaks clustering algorithm," *Knowl.-Based Syst.*, to be published.
- [9] L. Ni, W. Luo, W. Zhu, and W. Liu, "Clustering by finding prominent peaks in density space," *Eng. Appl. Artif. Intell.*, vol. 85, pp. 727–739, Oct. 2019.
- [10] 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. KDD*, 1996, pp. 226–231.
- [11] J. Shen, X. Hao, Z. Liang, Y. Liu, W. Wang, and L. Shao, "Real-time superpixel segmentation by DBSCAN clustering algorithm," *IEEE Trans. Image Process.*, vol. 25, no. 12, pp. 5933–5942, Dec. 2016.
- [12] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, Sep. 1977.
- [13] Y. Chen, L. Zhou, Y. Tang, J. P. Singh, N. Bouguila, C. Wang, H. Wang, and J. Du, "Fast neighbor search by using revised K-D tree," *Inf. Sci.*, vol. 472, pp. 145–162, Jan. 2019.
- [14] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proc. 23rd Int. Conf. Mach. Learn. (ICML)*, 2006, pp. 97–104.
- [15] Y. Chen, L. Zhou, N. Bouguila, B. Zhong, F. Wu, Z. Lei, J. Du, and H. Li, "Semi-convex hull tree: Fast nearest neighbor queries for large scale data on GPUs," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2018, pp. 911–916.
- [16] D. A. White and R. Jain, "Similarity indexing with the SS-tree," in *Proc. 12th Int. Conf. Data Eng.*, Dec. 2002, pp. 516–523.
- [17] A. Gunawan, "A faster algorithm for DBSCAN," M.S. thesis, Eindhoven Univ. Technol., Eindhoven, The Netherlands, 2013.
- [18] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," in *Proc. ACM Sigmod Rec.*, 1999, pp. 49–60.
- [19] C. Böhm, K. Kailing, P. Kröger, and A. Zimek, "Computing clusters of correlation connected objects," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2004, pp. 455–466.
- [20] V. Chaoji, M. A. Hasan, S. Salem, and M. J. Zaki, "SPARCL: Efficient and effective shape-based clustering," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 93–102.
- [21] Y.-P. Wu, J.-J. Guo, and X.-J. Zhang, "A linear DBSCAN algorithm based on LSH," in *Proc. Int. Conf. Mach. Learn.*, Aug. 2007, pp. 2608–2614.
- [22] S. Mahran and K. Mahar, "Using grid for accelerating density-based clustering," in *Proc. 8th IEEE Int. Conf. Comput. Inf. Technol.*, Jul. 2008, pp. 35–40.
- [23] J. Gan and Y. Tao, "On the hardness and approximation of Euclidean DBSCAN," *TODSACM Trans. Database Syst.*, vol. 42, no. 3, pp. 1–45, Jul. 2017.
- [24] Y. Chen, L. Zhou, S. Pei, Z. Yu, Y. Chen, X. Liu, J. Du, and N. Xiong, "KNN-BLOCK DBSCAN: Fast clustering for large-scale data," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published, doi: [10.1109/tsmc.2019.2956527](https://doi.org/10.1109/tsmc.2019.2956527).
- [25] Y. Chen, S. Tang, N. Bouguila, C. Wang, J. Du, and H. Li, "A fast clustering algorithm based on pruning unnecessary distance computations in DBSCAN for high-dimensional data," *Pattern Recognit.*, vol. 83, pp. 375–387, Nov. 2018.
- [26] A. Lulli, M. Dell'amicò, P. Michiardi, and L. Ricci, "NG-DBSCAN: Scalable density-based clustering for arbitrary data," *Proc. VLDB Endow.*, vol. 10, no. 3, pp. 157–168, Nov. 2016.
- [27] D. Han, A. Agrawal, W.-K. Liao, and A. Choudhary, "A novel scalable DBSCAN algorithm with spark," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2016, pp. 1393–1402.
- [28] K. Mahesh Kumar and A. Rama Mohan Reddy, "A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method," *Pattern Recognit.*, vol. 58, pp. 39–48, Oct. 2016.
- [29] Y. Chen, S. Tang, L. Zhou, C. Wang, J. Du, T. Wang, and S. Pei, "Decentralized clustering by finding loose and distributed density cores," *Inf. Sci.*, vols. 433–434, pp. 510–526, Apr. 2018.
- [30] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logistics*, vol. 52, pp. 7–21, Mar. 2005.



**SHAN-SHAN LI** received the bachelor's degree from the Computer Science and Technology, Huaqiao University, Quanzhou, China, in 2003, and the master's degree from the Computer Application, Huaqiao University, in 2006. She is currently a Staff Member with the Department of IT Development & Management, Huaqiao University, Xiamen, China. She is also with the Provincial Key Laboratory for Computer Information Processing Technology, Soochow University, China. Her current research interests are in machine learning and pattern recognition.