h-DBSCAN: A simple fast DBSCAN algorithm for big data

Shaoyuan Weng WENGSHAOYUAN@STU.HQU.EDU.CN
College of Computer Science and Technology Huaqiao University Xiamen, China

Jin Gou GOUJIN@HQU.EDU.CN
College of Computer Science and Technology Huaqiao University Xiamen, China

Zongwen Fan ZONGWEN.FAN@HQU.EDU.CN
College of Computer Science and Technology Huaqiao University Xiamen, China

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

DBSCAN is a classical clustering algorithm, which can identify different shapes and isolate noisy patterns from a dataset. Despite the above advantages, the bottleneck of DBSCAN is its computation time for high dimensional datasets. This work, thus, presents a simple and fast method to improve the efficiency of DBSCAN algorithm. We reduce the execution time in two aspects. The first one is to reduce the number of points presented to DBSCAN and the second one is to apply the HNSW technique instead of the linear search structure for improving its efficiency. The experimental results show that our proposed algorithm can greatly improve the clustering speed without losing or even obtaining better accuracy, especially for large-scale datasets.

Keywords: Optimization of Neighborhood Query, Big Data, Density-based Clustering, HNSW, DBSCAN

1. Introduction

Clustering is one of the major methods that have been intensively studied and widely used in data mining and knowledge discovery for decades (Xu and Tian, 2015). In contrast to supervised approaches (e.g., classification algorithms) (Caruana and Niculescu-Mizil, 2006), clustering is an unsupervised technique (Jain et al., 1999) that does not rely on any prior knowledge or ground truth of the data. To be more specific, clustering is able to group the similar property data into the same cluster to achieve high similarity within clusters and low similarity between clusters. As a result, the clustering technique is a major tool in many engineering and scientific applications, including image analysis (Deutsch and Horn, 2018; Yu et al., 2014), document analysis (Carullo et al., 2009), and so forth (Xie et al., 2018).

Among the clustering algorithms, Density Based Spatial Clustering of Applications with Noise (DBSCAN) (Ester et al., 1996) is the pioneer of density-based clustering technique which can discover clusters of arbitrary shapes and handle noise or outliers effectively. In DBSCAN, the density is associated with a point obtained by the number of neighbor points within a given radius of the range. The clusters will be construct according to two parameters density threshold $MinPts$ and ε range, when the density threshold of a point is higher than a given value. DBSCAN mainly consists of two steps. The first step is to

execute the *ε-neighbor* (region) query for each data point in a given dataset. This step results in $O(n^2)$ time complexity, where n is number of data points. The second step is to group the data points into different clusters based on the distance among them. As the dimensional and volume of data increase, the efficiency of DBSCAN in big data becomes a bottleneck. To address this problem, many algorithms (Ester et al., 1996; Mahran and Mahar, 2008; Kim et al., 2019; Chen et al., 2018; He et al., 2017; Tsai et al., 2009; Kumar and Reddy, 2016; Mai et al., 2016; Rodriguez and Laio, 2014; Bryant and Cios, 2017; Sarma et al., 2019) have been proposed to enhance the performance of DBSCAN.

For example, a couple of approximate algorithms (Mahran and Mahar, 2008; Kim et al., 2019) were proposed to optimize the runtime performance of DBSCAN in low-dimensional problems. For high-dimensional problems, sampling-based DBSCAN algorithms (Chen et al., 2018; He et al., 2017; Tsai et al., 2009) were introduced based on approximate neighborhood query computations. Although the above methods can improve the clustering efficiency, they compromise the clustering quality. Recently, a couple of algorithms (Kumar and Reddy, 2016; Mai et al., 2016; Rodriguez and Laio, 2014) have been reported to accelerate the neighbor query while preserving the clustering quality. They can be extended to large datasets by reducing time complexity using spatial index structures like kd-trees (Rodriguez and Laio, 2014) for finding neighbors of a pattern. Furthermore, the G-DBSCAN (Kumar and Reddy, 2016), a fast DBSCAN clustering algorithm, was proposed. It uses group to optimize neighborhood queries, and eliminates noise in advance in the process of constructing clusters to reduces the number of distance calculations. To optimize the ability of handling large variations in cluster density (heterogeneous density), RNN-DBSCAN (Bryant and Cios, 2017) was proposed. It is a new density-based clustering algorithm that uses reverse nearest neighbor counts as an estimation of observation density. It only uses a only one single parameter (choice of k nearest neighbors) to determine the density of each point, which reduces the complexity of the algorithm and improves the quality of clustering.

The above algorithms are utilised for optimizing the clustering quality of DBSCAN. The approximate DBSCAN algorithm was first proposed to optimize the running time efficiency of DBSCAN, and they improved the running time of the algorithm. But when considering high-dimensional data sets, these algorithms are not very applicable. Based on the above studies, a sampling-based method is proposed to improve the efficiency fo DBSCAN in large-scale datasets. A common problem for clustering algorithm is that when the running rate is relatively increased in a high-dimensional dataset, the quality of clustering cannot be guaranteed. Therefore, in order to ensure the quality of the clustering results and increase the speed, some exact DBSCAN algorithms have been proposed. Based on the above research, we propose a novel, simple but fast DBSCAN algorithm, *h*-DBSCAN. It can accelerate the algorithm's neighbor query while ensuring the quality of clustering, making it suitable for high-dimensional problems. The main contributions of this article are as follows:

(1) High efficiency. In the *ε-neighbor* region query, empty points are deleted in advance to reduce the redundant calculation, which improve the running efficiency of the algorithm and provide the exact DBSCAN clusters.

(2) Suitable for high-dimensional problems. Our proposed algorithm can not only speed up indexing but also make it suitable for solving high-dimensional problems by using Hierarchical Navigable Small World (HNSW) (Malkov and Yashunin, 2018) as an index structure.

The rest of this paper is organized as follows: in Section 3, we briefly introduce the DBSCAN and characteristics of HNSW; Section 4 presents the details of our proposed clustering algorithm; Section 5 demonstrates the experimental results of the proposed algorithms on various datasets, and Section 6 draws the conclusion and highlights our future works.

2. Related Work

Due to the high running time of the original DBSCAN, researchers have developed many methods to improve the performance of DBSCAN (Boonchoo et al., 2019). Here we broadly categorize the related papers in the literature into four categories as follows.

The study of improving the runtime of DBSCAN has been received a considerable amount of attention for decades. For example, Mahran and Mahar (2008) proposed GriDBSCAN using grid partitioning and merging to enhance the performance of DBSCAN. It is a well-developed algorithm whose complexity is improved to $O(nlogn)$ in 2D space, while requiring $\Omega(n4/3)$ computation time when dimension $\geq$3. Kim et al. (2019) presented an approximate algorithm by proposing a new tree structure based on a quadtree to accelerate the runtime performance. However, these DBSCAN variants cannot work well on large datasets with high performance.

To solve the problems with high dimensional data, NQ-DBSCAN (Chen et al., 2018) was proposed. It improves the traditional DBSCAN by applying an efficient neighbor query to reduce the search space. It assumes that any nearby points should have similar neighboring points, and uses the information of neighboring points to speed up the algorithm by performing the distance calculation with only necessary points to expand the cluster. Recently, sampling-based DBSCAN algorithms are deigned to improve the efficiency of ε-neighbor region query operation in DBSCAN. He et al. (2017) proposed a seed point selection method based on influence space and k neighborhood similarity. It can improve the efficiency by selecting some seed points instead of all the neighborhood during cluster expansion and thus decreasing the number in region queries. Another sampling-based method, called GF-DBSCAN (Tsai et al., 2009), is a grid-based clustering algorithm that can reduce the number of searches. Nevertheless, these methods cannot produce the exact results as required.

A graph-based DBSCAN (G-DBSCAN) (Kumar and Reddy, 2016) was introduced to optimize the neighborhood query computations by making groups of data points (initial clusters) and pruning noise points based on the group information. It performs neighborhood queries for all the points in the dataset (except noise points) in an optimized way and obtains exact DBSCAN clustering results. The authors claimed its time complexity is $O(n^d)$, where d is the average number of points in 5*ε region of a point. However, d can be as large as n. In addition, AnyDBC (Mai et al., 2016) is an anytime density-based clustering algorithm which gives clustering results at anytime with some approximation. The algorithm executes in an iterative fashion. The greater number of iterations, the better accuracy/closeness of the clustering to the DBSCAN. Exact DBSCAN clustering is obtained after a large number of iterations. To speed up the range query process, Rodriguez and Laio (2014) proposed a clustering algorithm named "Clustering by fast search and find of

density peaks". It uses kd-trees for indexing data, and performs substantially fewer range queries compared to DBSCAN while still guaranteeing the exact final results of DBSCAN.

A scalable RNN-DBSCAN (Bryant and Cios, 2017) solution was investigated to improve DBSCAN by using an approximate KNN algorithm. RNN-DBSCAN needs to traverse each data in the dataset to determine whether it is the core point according to the number of similar positions in the first place of each point. The computational complexity of RNN-DBSCAN depends on its solution to the nearest neighbor problem. More specifically, the reverse nearest neighbor approaches to RNN-DBSCAN is dependent on the all K nearest neighbor problems, while the time complexity is $O(k * n^2)$.

3. Background

3.1. Density-Based Clustering Algorithm

DBSCAN is designed to discover arbitrary-shaped clusters in any dataset X with n points n=$|X|$, and at the same time can distinguish noise points. More specifically, DBSCAN accepts two parameters radius value ε and density threshold $MinPts$, measure distance between two points based on distance function, $dist(x, y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$, where $x \in X$: $x \in R^d$. Some concepts and terms to explain the BDSCAN algorithm can be defined as follows (Mai et al., 2016).

Definitions 1(ε-*neighbor*): The ε-*neighbor* of a point $x \in X$, is a set of points inside an ε radius around x. ε-*neighbor*=$\{y \in X \mid dist\,(x, y) \leq \varepsilon\}$

We denote $neighbor_x$ as x's ε-*neighbor*, and $|neighbor_x|$ is the number of ε-*neighbor*.

Definitions 2(*Core Property*): An observation x is called a:

(1) Core point, denoted as $core_x$, iff $|neighbor_x| \geq MinPts$.

(2) Border point, denoted as $border_x$, iff $|neighbor_x| < MinPts \wedge \exists neighbor_x$ is $core_x$.

(3) Noise point, denoted as $noise_x$, otherwise.

Definitions 3(directly $Density - reachable$): A point x is directly density-reachable from a point y if:

(1) $x \in neighbor_y$

(2) y is a core point.

Directly density-reachable is non-symmetric for non-core observations, and not guaranteed to be symmetric in the case of core observations. The latter of the two cases being due to the fact that the nearest neighbor relationship is non-symmetric, and the former as no observation is reachable from a non-core observation (Bryant and Cios, 2017).

Definitions 4(*Density-reachability*): A point x is directly *density-reachable* from point y, iff x is $core_x$ and $dist(x, y) \leq \varepsilon$.

Definitions 5(*Density-connected*): Two points x and y are *density-connected*, iff there is a point t such that both x and y are *density-reachability* from t. A cluster is a maximal set of density-connected points (Sarma et al., 2019).

Definitions 6(*Cluster*) (Mai et al., 2016): A cluster C with respect to ε and $MinPts$ is a $non - empty$ subset of x satisfying the following conditions:

(1) $\forall x, y$: if $x \in C$ and x is *density-reachable* from x with respect to ε and $MinPts$, then $y \in C(Maximality)$.

(2) $\forall x, y \in C : x$ is *density-connected* y with respect to ε and $MinPts(Connectivity)$.

The algorithms starts with the first point x in dataset X, and find the neighbor within ε distance. If the number of $neighbor_x$ more than $MinPts$ x is core point, then create a new cluster. The $neighbor_x$ and core x is assigned into this new cluster. Then, it iteratively collects the neighbors within ε distance from the core points. The process is repeated until all of the points have been processed.

3.2. Hierarchical NSW

HNSW is an algorithm for approximating the K-nearest neighbor search based on navigable small world graphs with controllable hierarchy. It is a multi-layer approximate graph structure. The graph is constructed by inserting data points in consecutive. The integer maximum layers l is determined by random selection with an exponentially decaying probability distribution. The insertion process mainly consists of two steps. The first step is to find the K closest neighbors in current layer from the top layer to the bottom layer. After that, the algorithm continues the search entry point into the next layer from the K closest neighbors. Repeating the procedure until all the data points have found K closest neighbors. The structure of HNSW is show in Fig. 3.

Give a dataset with N data points, the expected time complexity of the maximum layer index by the construction scales is $O(log(N))$ while the overall complexity scaling of the overall complexity scaling is $O(log(N))$. This is in agreement with the simulations on low dimensional datasets. The insertion complexity scaling with respect to N is the same as the one for the search. This means that, at least for relatively low dimensional datasets, the construction time scale is $O(N \cdot log(N)))$ (Malkov and Yashunin, 2018).

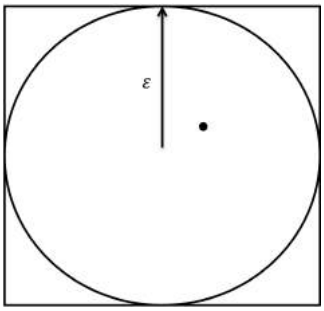


Figure 1: Empty point with radius=ε

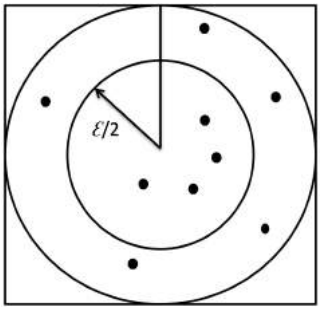


Figure 2: $Inner\text{-}Core_x$ with radius=ε, $MinPts$=4

4. The Proposed Algorithm: *h*-DBSCAN

4.1. *h*-DBSCAN Definitions

Definitions 7($Empty_x$): An observation x is an empty point if it does not contain any ε-*neighbor* points, (see Fig. 1). $Empty_x = \{x \in X | neighbor_x = \emptyset\}$

Definitions 8($Inner\text{-}Core_x$): The $Inner\text{-}Core_x$ of $core_x$ is defined as follows, (see Fig. 2): $Inner\text{-}Core_x = \{y | y \in neighbor_x \wedge dist(x, y) \leq \frac{\varepsilon}{2}\}$

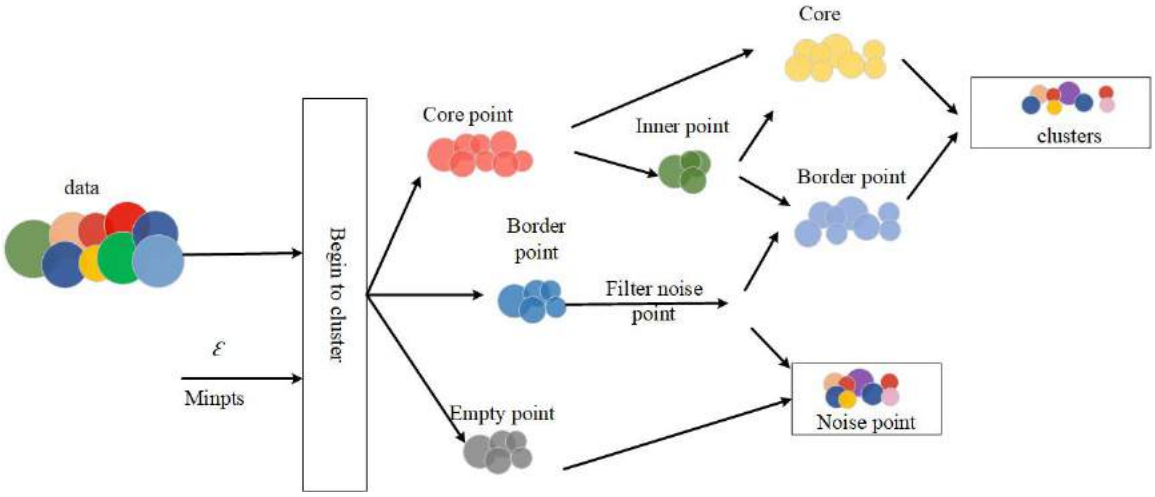


Figure 3: Illustration of the h-DBSCAN steps.

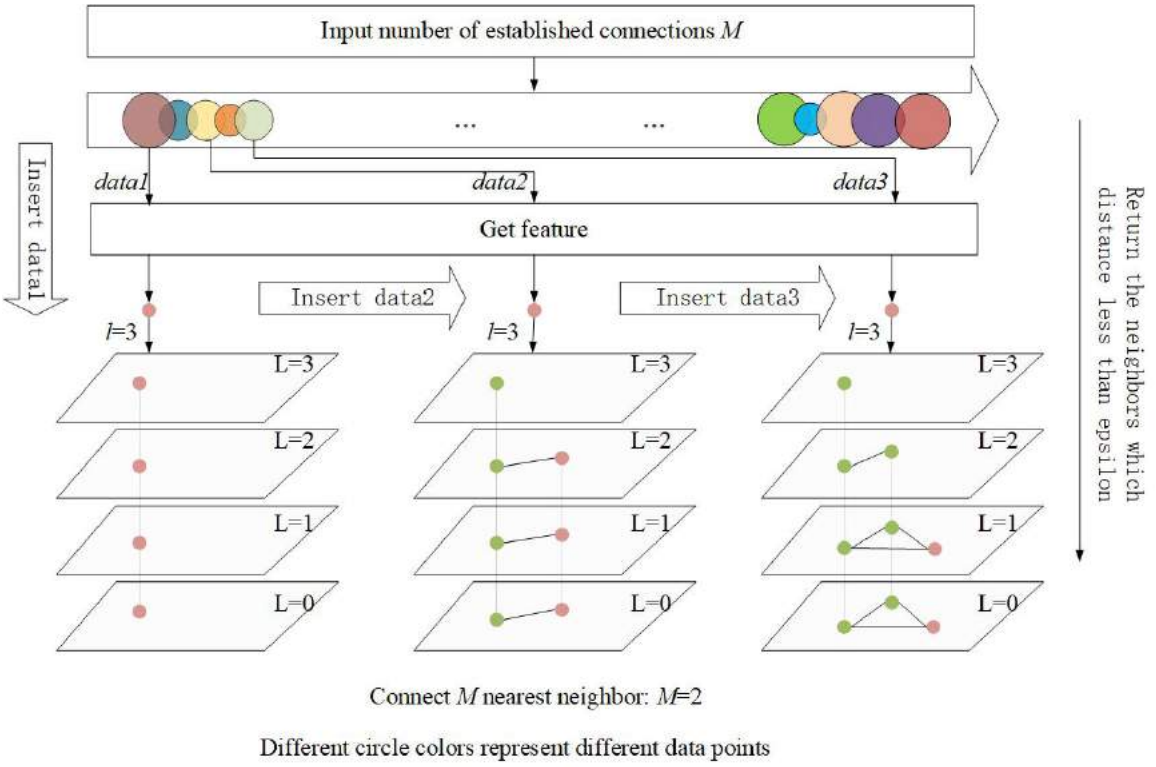


Figure 4: Illustration of the HNSW structure for range query.

4.2. Overview of *h*-DBSCAN

Algorithm 1 and Fig. 3 show the process of h-DBSCAN performing clustering, as defined in Section 3, given dataset X, radius ε and density threshold *MinPts*. After traversing all the observations in an arbitrary order to find the ε-*neighbor* (Algorithm 2), the observation will be assigned to a new cluster when it is a core observation and has yet assigned to any cluster. This new cluster is expanded by a breadth first search of all reachable observations and density-connected cluster will be merged (Algorithm 3). Finally, the border point will filter by Algorithm 4 without perform ε range query.

Algorithm 1: h-DBSCAN

Require: Data List X, $[\varepsilon, MinPts]$;
Ensure: Cluster;

```
Initialize CoreSet=∅, BorderSet=∅, Initial Cluster=∅, cluster[∀x ∈ X]=
UNCLASSIFIED;
K=MinPts
for x ∈ X do
    if cluster[x]=UNCLASSIFIED then
        neighbor_x=HNSW::RangeQuire(x, K) //find ε-neighbor alg. 2;
        if |neighbor_x| ≥ K then
            if |inner-core_x| ≥ K then
                CoreSet = CoreSet∪ inner-core_x
            end
            Cluster = Cluster ∪ [neighbor_x]
        end
        if |neighbor_x| < K then
            if |neighbor_x| ≠ 0 then
                BorderSet = BorderSet ∪ x;
            end
            else
                cluster[x] = noise;
                Delete x from the data set X that still needs to be queried;
            end
        end
    end
end
MergeCore(Cluster, CoreSet) //merge clusters alg. 3;
FiltNoise(BorderSet) //filter noise point from border set alg. 4;
return Cluster;
```

Step 1: Building initial cluster. In this step, we insert each point in dataset to construct graph structure of HNSW. Thereafter, all point ε-*neighbor* range queries are based on this structure. Then, all the points in the dataset are traversed to determine their type. In this step, we process each point $x \in X$ and do the following:

(1) If $0 < |neighbor_x| < MinPts$: x is marked as a border point, and it will be assign to the set of $BorderSet$. Otherwise x will be assigned to the set of the $CoreSet$, and if x has not yet assigned to any cluster, then create a new cluster and merge the $neighbor_x$ and x into it.

(2) If $|neighbor_x| = 0$: then x is an empty point and marked as noise. At the later range queries ε-$neighbor$, other points do not operate on it. Remove it from the HNSW structure.

(3) If $|inner\text{-}core_x| \geq MinPts$: then put $\forall x \in inner\text{-}core_x$ into the core set, and they are deleted from the remaining dataset that needs to be queried, as show in Fig. 2.

Algorithm 2: FindNeighbor

Require: query point x; the K value of HNSW;
Ensure: $neighbor_x$;
1: $neighbor_x = \emptyset$;
2: $neighbor$ = HNSW :: $RangeQuire(x, K)$;
3: **for** i *in* $neighbor$ **do**
    if $dist(i, x) \leq \varepsilon$ **then**
        $neigbor_x = neigbor_x + i$
    end
    end
4: **return** $neighbor_x$

Algorithm 3: MergeCore

Require: $Cluster$, $CoreSet$;
Ensure: $cluster$;
1: k=-1
2: **for** x *in* $Cluster$ **do**
    $k = k + 1$;
    for *each* i *in* $neighbor_x$ **do**
        if $i \in CoreSet$ **then**
            $neighbor_x = neighbor_x \cup neighbor_i$;
            $cluster[neighbor_x] = k$;
        end
    end
    end
3: **return** $cluster$

Step 2: Clusters Merging. After finishing the above steps, we have built the initial cluster including the core point and itsε-$neighbor$, and find the core point in $inner\text{-}core_x$ that satisfy the definition of core point. In the rest points, if they belong to the ε-$neighbor$ and besides $inner\text{-}core_x$, we have not identify their property, they may be the core point.

Algorithm 4: FilterNoise

Require: *BorderSet*;
Ensure: *cluster*;
1: **for** *each x in BorderSet* **do**
    if $neighbor_x \cap CoreSet \neq Null$ **then**
        return $cluster[x] = cluster[core]$ **else**
            return $cluster[x] = noise$
        end
    end
    end
2: **return** *cluster*

i.e. the *density-connected* cluster have not merged yet. Therefore, in this step, our main task is to merge points that belong to the same cluster. After the above steps, we have all the core points in the *CoreSet*. Therefore, when identify their property, we only need to query whether they are in *CoreSet* (Algorithm 3).

Step 3: Filter Noise. After the above steps, we have clustered the core points and their density-connected points into a cluster. In this step, our main task is to filter the border points that have not assigned to any cluster yet. We traverse through the neighbors of the border point, select the border point with the core point, and then assign the border point to the cluster where the core point neighbors are located. This step doesn't require any additional neighborhood queries, as they are pre-computed and stored in the previous step (Algorithm 4).

4.3. *h*-DBSCAN Complexity

The time complexity of h-DBSCAN (worst case) =initial cluster construction + merging time + filtering noise= $O(log(N)))$ + $O(N \cdot log(N)))$+$O(m * k)$ + $O(l)$, where m is the initial cluster number, k is the number of $|neighbor_x|$ and l is the number of border point. The space occupied by graph construct, noise set, core set and border set. Thus the total space complexity is $O(n + m + l)$.

5. Experiments and Performance Evaluation

In this section, we present the experimental results based on both real-world and synthetic datasets.

5.1. Experimental Settings

In this section, to evaluate the correctness and effectiveness of the proposed approach, several experiments are conducted on different datasets using the environment of Intel Core i7-3630 CPU @2.50 GHz, 8G RAM. For the comparison experiments, we mainly compare the proposed algorithm with DBSCAN (Ester et al., 1996), RNN-DBSCAN (Bryant and Cios, 2017), G-DBSCAN (Kumar and Reddy, 2016) and pure kd-trees based DBSCAN. We programmed the proposed h-DBSCAN using the Python programming language.

Table 1: Artificial Datasets

data	number	classes	dimensions
Aggregation	788	5	2
t4.8k	8000	7	2
D31	3100	30	2

Table 2: Real-World Datasets

data	number	dimensions	classes
Iris	150	4	3
htru2	17898	8	2
Ecoil	336	7	8
Ctg	2126	19	10
Digits	1797	64	10
HIGGS13D	10000	13	2
HIGGS28D	1800	28	2
3DSRN3D	10873	3	-
HOUSEHOLD	18000	7	-

5.2. Datasets

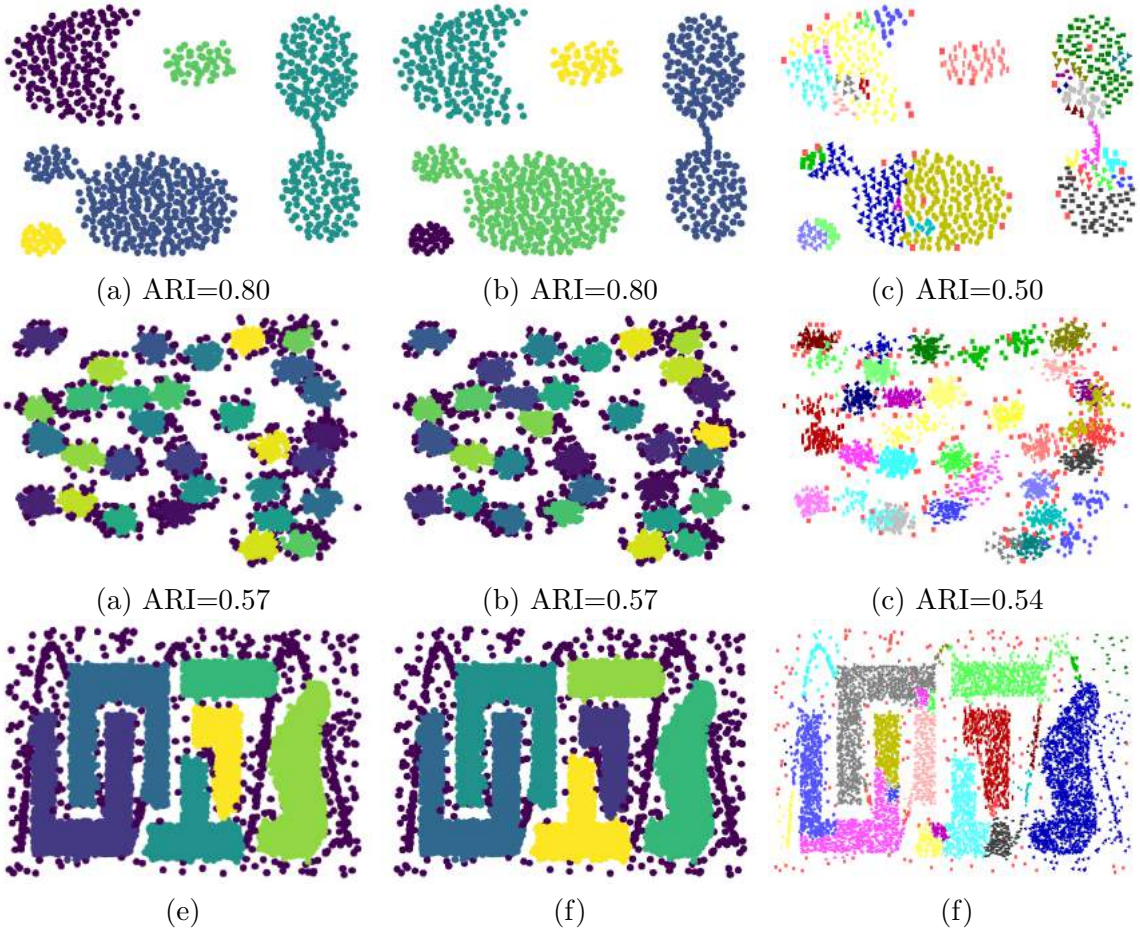
We benchmark our proposed algorithms with their respective existing solutions to various real-world datasets commonly used in the literature for evaluating density-based clustering algorithms. The artificial, shaped-based clustering datasets were downloaded from (https://cs.joensuu.fi/sipu/datasets/), including Aggregation, t4.8k and D31 were used. We also downloaded a few real-world datasets from UCI(http://archive.ics.uci.edu/ml/datasets), including 3D Road Net-work (3DSRN) (Kaul et al., 2013) contains vehicular GPS data; HOUSEHOLD Power (HOUSEHOLD); HIGGS (Baldi et al., 2014), HIGGS datasets have been sampled for various dimensions (13 and 28); iris (iris plant type); ctg (cardiotocography fetal state); digits (optical recognition of handwritten digits); ecoli (ecoli protein localization sites); htru2 (pulsar candidates) (Lyon et al., 2016). The details of the artificial datasets are shown in Table 1 while the real-world datasets are shown in Table 2. For comparative experimental study, we used the implementations to compare algorithms that were made publicly available by their respective authors except for programming G-DBSCAN by us.

5.3. Experimental Results

In this section, we demonstrate the experimental results. All the reported running time of the compared methods are based the average of 3 runs of clustering. It is worth noting that when comparing the running time of algorithms, three steps are included, namely, cluster construction, cluster merging, and noise filtering.

5.3.1. Clustering Result Quality

To test whether clustering algorithm can cluster the same clusters as DBSCAN (clustering accuracy), we use the number of clusters in DBSCAN as a benchmark. The clustering results are in Table 3, which shows our algorithm and K-DBSACNA produce exactly the same results as the original DBSCAN on all the measured datasets. This means our proposed algorithm is able to group all the points into the clusters exactly the same as the original DBSCAN does while G-DBSCAN and RNN-DBSCAN have similar results to DBSCAN.



(a) ARI=0.80 (b) ARI=0.80 (c) ARI=0.50

(a) ARI=0.57 (b) ARI=0.57 (c) ARI=0.54

(e) (f) (f)

Figure 5: Clustering results for artificial dataset DBSCAN (right) h-DBSCAN (center) RNN-DBSCAN (left). Aggregation dataset (1): ε=1.6 $MinPts$=3; D31 dataset (2): ε=0.8 $MinPts$=30; t48.k dataset (3): ε=8.5 $MinPts$=15.

To further test the performance of our algorithm, two indicators are used, ARI (Hu.Be.Rt and Arabie, 1985) and Normalized Mutual Information (NMI) (Bryant and Cios, 2017), (Cover et al., 1991), for evaluation. ARI represents the similarity measure between two clustering that is adjusted for chance and is related to accuracy, while NMI quantifies the amount of information obtained about one clustering, through the other clustering (i.e., the mutual dependence between the two). In the case of observations being identified as noise,

each noise observation was treated as a distinct singleton cluster for both ARI and NMI. For ARI results, clustering Purity is also presented which is a weighted average of the percentage of observations belonging to the dominant class in each cluster. Noise observations were ignored in the calculation of purity (Bryant and Cios, 2017).

Table 4 shows the ARI performance, Purity, number of clusters, and number of observations identified as noise for DBSCAN, h-DBSCAN, RNN-DBSCAN, G-DBSCAN, and k-DBSCAN on real-world datasets. It can be seen from the results that our algorithm is basically consistent with the results of DBSCAN, and the results of G-DBSCAN are very close to DBSCAN. The performance of RNN-DBSCAN in the digits dataset is slightly better than DBSCAN. In addition, we can see the consensus that all the measures confirm that our algorithm can group all the points into the clusters exactly the same as the original DBSCAN does. We also provide the 2D visualization of the compared methods in Fig. 6. As we can see from the figure, the accuracy of RNN-DBSCAN is slightly lower than our method with the same parameters used (Bryant and Cios, 2017).

Table 3: Clusters produced by the clustering methods.

Dataset	Paremeters				Clusters				
	dimensions	number	ε	$MinPts$	DBSCAN	h-DBSCAN	RNN-DBSCAN	G-DBSCAN	K-DBSCAN
3DSRN3D	3	10873	0.1	20	4	4	8	4	4
HOUSEHOLD	7	18000	1	40	9	9	11	9	9
HIGGS13D	13	10000	2	5	2	2	2	2	2
HIGGS28D	28	1800	2.4	5	2	2	2	2	2

Table 4: Cluster quality performance on real-world dataset.

DataSet	Method				Method				Method				Method				Method			
	DBSCAN				h-DBSCAN				RNN-DBSCAN				G-DBSCAN				K-DBSCAN			
	ari	clu	pur	noi	ari	clu	pur	noi	ari	clu	pur	noi	ari	clu	pur	noi	ari	clu	pur	noi
iris	**0.703**	7	0.978	16	**0.703**	7	0.978	16	0.644	4	0.963	16	0.7	6	0.969	16	**0.703**	7	0.978	16
htru	**0.552**	4	0.977	2289	**0.552**	4	0.977	2288	0.334	204	0.976	236	0.55	5	0.977	2189	**0.552**	4	0.977	2289
ecol	**0.639**	3	0.582	100	**0.639**	3	0.581	98	0.526	8	0.736	10	0.64	3	0.581	89	**0.639**	3	0.582	100
ctg	**0.992**	13	1	5	**0.992**	13	1	5	0.951	10	1	91	0.992	13	1	5	**0.992**	10	1	6
digits	0.684	21	0.983	355	0.684	21	0.983	355	**0.739**	34	0.936	104	0.684	21	0.983	355	0.684	15	0.989	155

5.3.2. Runtime Comparisons

In this section, to further validate our proposed method for large-scale dataset, the big datasets of 3DSRN, HOUSEHOLD and HIGGS with different dimensions are used in the experiments. We compare the runtime performance of h-DBSCAN with that of original DBSCAN and its variants that use kd-trees as indexing structure (K-DBSCAN), G-DBSCAN, as well as the latest proposed RNN-DBSCAN. The results presented in Table 5 show that h-DBSCAN has performed consistently better than the remaining four algorithms in compared for all the datasets.

We also report the peak memory consumption of h-DBSCAN and compare with other algorithms for various datasets (see Table 6). The results show that the peak memory consumption for h-DBSCAN is much less than DBSCAN, especially for high dimensional datasets. The peak memory consumption of K-DBSCAN, RNN-DBSCAN and G-DBSCAN

Table 5: Runtime comparison (in seconds) of h-DBSCAN with other algorithms

Dateset			Parameter		Time				
	dimensions	number	ε	*MinPts*	DBSCAN	h-DBSCAN	RNN-DBSCAN	G-DBSCAN	K-DBSCAN
Aggregation	2	788	1.5	6	4.381	**0.21**	0.35	2.03	1.998
t48.k	2	8000	8.5	15	456.262	**4.045**	5.297	132.45	188.576
D31	2	3100	0.8	30	65.716	**0.958**	1.019	15.79	21.576
Iris	4	150	0.4	9	0.165	**0.014**	0.136	0.099	0.016
htru2	8	17898	0.3	15	2350.679	**13.855**	125.108	92.353	39.588
Ecoil	7	336	0.8	30	0.861	**0.193**	0.218	0.432	0.363
Ctg	19	2126	1.5	6	4.381	**0.21**	0.35	2.03	1.998
Digits	64	1797	8.5	15	456.262	**4.045**	5.297	132.45	188.576
3DSRN3D	3	10873	0.1	20	798.537	**4.755**	37.766	252.72	613.309
HOUSEHOLD	7	18000	1	40	2164.763	**38.869**	130.086	1115.86	1274.393
HIGGS13D	13	10000	2	5	692.274	**7.344**	14.745	256.17	416.324
HIGGS28D	28	1800	2.4	5	22.182	**0.160**	0.917	16.22	19.462

Table 6: Peak memory consumption of h-DBSCAN and others algorithms

Dataset			Parameter		Memory				
	dimensions	number	ε	*MinPts*	DBSCAN	h-DBSCAN	RNN-DBSCAN	G-DBSCAN	K-DBSCAN
3DSRN3D	3	10873	0.1	20	33.44MB	**4.04MB**	6.52MB	7MB	5.21MB
HOUSEHOLD	7	18000	5	10	66.87MB	**6.57MB**	9.11MB	26.33MB	35.24MB
HIGGS13D	13	10000	2	5	28.12 MB	**4.13MB**	32.18177	10.24MB	16.25MB
HIGGS28D	28	1800	2.4	5	0.64MB	0.61MB	1.23MB	**0.55MB**	0.78MB

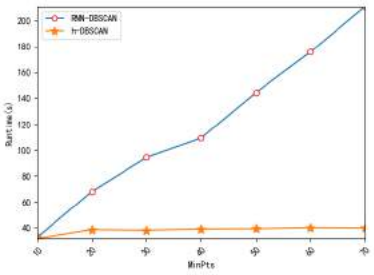
Table 7: Split-up of execution time of various steps of h-DBSCAN (in seconds)

Dataset			Steps			
	dimensions	number	HNSW Construction and Building initial cluster	Clusters Merging	Filter Noise	Total time
3DSRN	3	10873	**22.670181**	3.58	0.24	26.490181
HOUSEHOLD	7	18000	**95.371852**	7.542595	0.001001	103.308221
HIGGS13D	13	10000	**22.19886**	7.0554	0.673404	29.927664
HIGGS28D	28	1800	**7.56304**	0.04996	0.01	7.67296

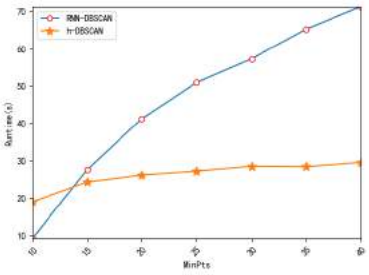
are less than DBSCAN as well. This is because K-DBSCAN uses a simple kd-tree that occupies less memory than original DBSCAN while RNN-DBSCAN and G-DBSCAN do not need any indexing structure. As we can see from the table, the runtime performance of h-DBSCAN has been far superior to the methods being compared and the memory required is less than the others for all datasets except for HIGGS28D dataset with similar memory consumption with G-DBSCAN.

In order to check the split-up execution times of each step in h-DBSCAN algorithm, we also record the calculation time for each step of the algorithm. As the results shown in Table 7, the initial cluster build takes significant portion of the execution time because this step includes the neighborhood range query.

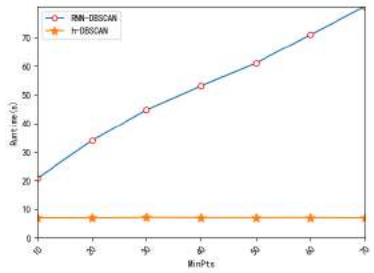
To verity the importance of $MinPts$ on runtime performance of DBSCAN, and RNN-DBSCAN, we present the results based on the change of $MinPts$ in Fig. 6. As we can see from the figures, the runtime increases with the increase of $MinPts$. This is because, with the increase in $MinPts$, the time for initial clusters formation and dentist-reachable cluster identification decreases, and time for post processing of *inner-core* points increases.



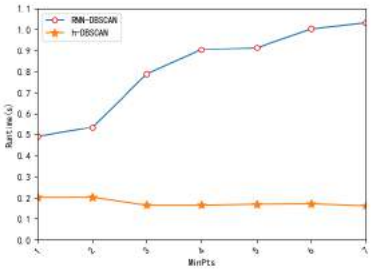
(a)HOUSEHOLD



(b)3DSRN



(c)HIGGS13D



(d)HIGGS28D

Figure 6: Runtime distrubutions with the changeing $MinPts$ on four datasets respectively.

From Table 7 and Fig. 6, we can infer that: 1) the complexity of h-DBSCAN mainly depends on the execution ε-*neighbor* query; 2) h-DBSCAN prefers less $MinPts$ in the same case as ε, which yields less executions of $core_x$ identified;

6. Conclusions

In this study, we proposed an improved DBSCAN (h-DBSCAN) to improve the clustering efficiency of the DBSCAN. The strategies used for reducing the computation time include: 1) using improved HNSW instead of linear search for region query, which is an approximate multi-layer graphic structure that distributes data points into different layers, and it retains all the data point in the structure and can run faster without sacrificing the quality of the clustering; 2) recognizing $empty_x$ as noise and $inner\text{-}core_x$ as core in advance to reduce ε-*neighbor* queries. By comparing our proposed algorithm with DBSCAN and its variants, our proposed method outperformed the others methods in terms of efficiency, especially for large-scale datasets. In addition, in most cases, the accuracy of our h-DBSCAN is better than G-DBSCAN and RNN-DBSCAN algorithms.

Although the runtime performance of the proposed algorithm can be significantly improved, there is still room for improving the clustering accuracy. Therefore, in the future work we will investigate different ways to improve the accuracy of the algorithm without losing efficiency (e.g. weights for data points). Moreover, considering the clustering qual-

ity of traditional DBSCAN is greatly influenced by ε and $MinPts$, we will explore the optimization algorithms to automatically select the parameters for DBSCAN.

References

P. Baldi, P. Sadowski, and D. Whiteson. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Communications*, 5:4308, 2014.

Thapana Boonchoo, Xiang Ao, Yang Liu, Weizhong Zhao, Fuzhen Zhuang, and Qing He. Grid-based DBSCAN: Indexing and inference. *Pattern Recognition*, 90:271–284, 2019.

Avory Bryant and Krzysztof Cios. RNN-DBSCAN: A density-based clustering algorithm using reverse nearest neighbor density estimates. *IEEE Transactions on Knowledge and Data Engineering*, 30(6):1109–1121, 2017.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 161–168, 2006.

Moreno Carullo, Elisabetta Binaghi, and Ignazio Gallo. An online document clustering technique for short web contents. *Pattern Recognition Letters*, 30(10):870–876, 2009.

Yewang Chen, Shengyu Tang, Nizar Bouguila, Cheng Wang, Jixiang Du, and HaiLin Li. A fast clustering algorithm based on pruning unnecessary distance computations in DBSCAN for high-dimensional data. *Pattern Recognition*, 83:375–387, 2018.

T. M. Cover, J. A. Thomas, J. Bellamy, R. L. Freeman, and J. Liebowitz. *Elements of Information Theory WILEY SERIES IN Expert System Applications to Telecommunications.* Elements of Information Theory, 1991.

Lior Deutsch and David Horn. The weight-shape decomposition of density estimates: a framework for clustering and image analysis algorithms. *Pattern Recognition*, 81:190–199, 2018.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. *AAAI Press*, 96(34): 226–231, 1996.

Qing He, Hai Xia Gu, Qin Wei, and Xu Wang. A novel DBSCAN based on binary local sensitive hashing and binary-KNN representation. *Advances in Multimedia*, 2017, 2017.

L. Hu.Be.Rt and P. Arabie. Comparing partitions. *J. Classif*, 2(1):193–218, 1985.

Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

Manohar Kaul, Bin Yang, and Christian S Jensen. Building accurate 3d spatial networks to enable next generation intelligent transportation systems. In *2013 IEEE 14th International Conference on Mobile Data Management*, volume 1, pages 137–146. IEEE, 2013.

Jeong-Hun Kim, Jong-Hyeok Choi, Kwan-Hee Yoo, and Aziz Nasridinov. AA-DBSCAN: an approximate adaptive DBSCAN for finding clusters with varying densities. *The Journal of Supercomputing*, 75(1):142–169, 2019.

K Mahesh Kumar and A Rama Mohan Reddy. A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method. *Pattern Recognition*, 58:39–48, 2016.

Lyon, R, J, Stappers, B, W, Cooper, S, Brooke, and J. Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 2016.

Shaaban Mahran and Khaled Mahar. Using grid for accelerating density-based clustering. In *2008 8th IEEE International Conference on Computer and Information Technology*, pages 35–40. IEEE, 2008.

Son T Mai, Ira Assent, and Martin Storgaard. AnyDBC: an efficient anytime density-based clustering algorithm for very large complex datasets. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1025–1034, 2016.

Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2018.

Alex Rodriguez and Alessandro Laio. Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496, 2014.

Aditya Sarma, Poonam Goyal, Sonal Kumari, Anand Wani, Jagat Sesh Challa, Saiyedul Islam, and Navneet Goyal. μDBSCAN: An exact scalable DBSCAN algorithm for big data exploiting spatial locality. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11. IEEE, 2019.

Cheng-Fa Tsai, Chien-Tsung Wu, and Shengyong Chen. Gf-dbscan; a new efficient and effective data clustering technique for large databases. *World Scientific and Engineering Academy and Society (WSEAS)*, 2009.

Haiming Xie, Guangyu Tian, Hongxu Chen, Jing Wang, and Yong Huang. A distribution density-based methodology for driving data cluster analysis: A case study for an extended-range electric city bus. *Pattern Recognition*, 73:131–143, 2018.

Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

Jun Yu, Richang Hong, Meng Wang, and Jane You. Image clustering based on sparse patch alignment framework. *Pattern Recognition*, 47(11):3512–3519, 2014.