The Journal of Supercomputing
https://doi.org/10.1007/s11227-020-03524-3

K-DBSCAN: An improved DBSCAN algorithm for big data

Nahid Gholizadeh[1] · **Hamid Saadatfar**[1] · **Nooshin Hanafi**[1]

Accepted: 16 November 2020

Abstract
Big data storage and processing are among the most important challenges now. Among data mining algorithms, DBSCAN is a common clustering method. One of the most important drawbacks of this algorithm is its low execution speed. This study aims to accelerate the DBSCAN execution speed so that the algorithm can respond to big datasets in an acceptable period of time. To overcome the problem, an initial grouping was applied to the data in this article through the K-means++ algorithm. DBSCAN was then employed to perform clustering in each group separately. As a result, the computational burden of DBSCAN execution reduced and the clustering execution speed increased significantly. Finally, border clusters were merged if necessary. According to the results of executing the proposed algorithm, it managed to greatly reduce the DBSCAN execution time (98% in the best-case scenario) with no significant changes in the qualitative evaluation criteria for clustering.

Keywords Data mining · Clustering · Big data · DBSCAN algorithm · K-means++ algorithm

1 Introduction

The age of big data has resulted in the development and application of technologies and methods aimed at utilizing large amounts of data to support decision-making and knowledge discovery activities [1]. Large amounts of data have made

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s11227-020-03524-3) contains supplementary material, which is available to authorised users.

✉ Hamid Saadatfar
saadatfar@birjand.ac.ir

Nahid Gholizadeh
nahid.gholizadeh@birjand.ac.ir

Nooshin Hanafi
nooshinhanafi@birjand.ac.ir

[1] University of Birjand, Birjand, South Khorasan, Iran

Published online: 26 November 2020

researchers and industries reconsider computational solutions for the analysis of big data. For instance, great emphasis has been put on the design of new algorithms, which are more efficient in computation, for the analysis of data on Twitter, Google, Facebook, and Wikipedia [2]. This enormous amount of data can be very useful for individuals and companies; however, analysis and recovery operations can become too time-consuming because of the high computational costs of data processing.

A category of common methods for data analysis is referred to as data mining which means the identification of useful, reliable, simple, and understandable data patterns turning raw data into useful data or information [3]. One of the data mining techniques is clustering. Data clustering is considered an important area of unsupervised learning in which data can be divided into different groups based on their similarities from an informed perspective on the entire dataset [4]. Clustering is used in a wide range of areas such as vehicle re-identification [5], image denoising [6], time-series processing [7], and Web-based social network analysis [8]. Today, these conventional methods are not efficient enough in dealing with big data; thus, they are not applicable due to the high computational complexity [9].

The DBSCAN algorithm [10] is one of the most popular, versatile clustering algorithms. However, this algorithm fails to function properly in confrontation with big data. One of its most important drawbacks is its low execution speed. Given the rate of progress in today's world, time is one of the most important parameters and should receive special attention. To overcome the problem of big data processing, a variety of methods have been suggested, including the optimization of conventional algorithms in terms of computation and even memory consumption, data reduction through reduction of dimensions or data samples, data stream processing, and use of heavy-computation machines.

This paper seeks to improve the execution speed of the DBSCAN algorithm through the idea of reducing computation by dividing the workspace into a number of separate areas. To divide data into smaller groups, we utilize another famous clustering algorithm known as K-means++ [11]. This algorithm exhibits a proper convergence speed and can provide us with good data division. It should be mentioned that running K-means a few iterations also lets us achieve the desired results without a high computational burden imposed on the system because K-means++ is employed not to obtain the final clustering of the data but simply to divide them.

The proposed algorithm functions as follows in general. First, the data are grouped by the K-means++ algorithm with a small number of iterations applied. Then, the DBSCAN algorithm is applied separately to each part, and the data are clustered. Finally, the different parts are integrated. For obtaining the core points in the DBSCAN algorithm, it has to be analyzed for each piece of data if there are MinPts pieces of data within a radius of ε. Therefore, the distances between each piece of data and all the others existing in the dataset need to be calculated. Thus, a great deal of computation is carried out for a large dataset. In the proposed method, however, the data space to be considered is smaller because DBSCAN is applied separately in each group, and there are consequently a smaller number of points, the distances of which must to be calculated. This reduces execution time considerably.

The rest of this paper consists of the following sections. Section 2 reviews the research literature. Section 3 presents the proposed algorithm. Section 4 analyzes

the experiments and the analysis results of running the new algorithm. Section 5 draws a conclusion. Finally, Sect. 6 makes suggestions for future research.

2 Related work

Today, organizations are seeking to record and analyze data through significant methods to be able to realize data-based decision making [1]. Big data can be turned into structured, regular data for easier processing. In this paper, we deal with structured and large-sized data.

As defined in [12], "big data" is a term referring to the data that require new methods and architectures to be processed and analyzed for knowledge extraction. Given the large size of these data, it is very difficult to analyze them efficiently through the existing conventional techniques.

According to the studies reviewed by [13, 14], what we mean by big data in this research is the datasets that cannot be managed and processed by conventional software tools on a single machine in an acceptable time.

In recent years, a large number of clustering methods have been introduced for big data. Used for enforcement of clustering algorithms to work with big data through improvement of their scalability and speeds, these methods can be classified into two main groups [15]: methods based on a single machine and methods based on multiple machines.

Clustering algorithms based on a single machine run on one machine and can use only the resources of that machine. Clustering algorithms based on multiple machines run on two or more machines and can utilize the resources of several machines simultaneously.

2.1 Clustering methods based on a single machine

This section reviews the methods proposed in previous research for resolving the problem of processing big data. They can be divided into three groups. The first technique is the data size reduction technique, which can be carried out through the reduction of samples, dimensions, or both. The second technique is data stream processing, which makes it possible to process a large amount of data on one machine but suffers from certain drawbacks. The third technique is data processing optimization algorithms, in which data can be processed with a lower computational burden. The idea of this paper lies in the third group of methods. A few studies of these methods are reviewed here.

Published in 2006, [16] proposes a fast density-based clustering algorithm based on the DBSCAN method utilizing the idea of reducing computation. The algorithm first sorts all the data by dimensional coordinates and then increases the total execution speed of DBSCAN by decreasing the execution time of each region query or the frequency of region queries. A drawback of this method is that the algorithm is run only over a dataset of limited dimensions.

In 2007, Zhang et al. presented a linear DBSCAN algorithm based on LSH (Locality-Sensitive Hashing) to increase the rate at which the nearest neighboring nodes were found [17]. The advantage of using LSH is that it reduces time complexity and data scale. The algorithm considered two parts. The LSH index was constructed in the first part, whereas clustering was carried out in the second part by the DBSCAN algorithm based on the LSH index of retrieval. A drawback of the algorithm was the difficulty in specifying values for the input parameters.

A new clustering algorithm was introduced by Dogan et al. in 2013 [18]. This algorithm was proposed to improve clustering accuracy by integrating the k-means++ algorithm with a self-organizing map (SOM). In fact, k-means++ first determined the initial weights and initial centroids. The SOM was then used to find an appropriate solution to the final clustering of data. Aiming to propose optimal and high-speed clustering, the SOM++ algorithm performed properly in terms of training time saving and error rate. In other words, it can be used effectively for defect prediction. Nevertheless, the main problem with k-means++ is that it is inherently sequential; as a result, its efficiency is limited for the big data. However, the problem was resolved in this paper. Due to its nature, there is no need to iterate k-means++ until the end in the proposed algorithm because the iterations were limited.

Another reviewed paper presented a method of the incremental density-based type of algorithms [19]. Proposed in 2015 for big datasets, the method aimed to improve the incremental DBSCAN algorithm by restricting the search space to a few parts rather than using the entire dataset. This accelerates the clustering process.

The hierarchical clustering algorithm H–K-means was proposed in 2017 to find the high-quality initial centroids for big datasets [20]. In this algorithm, data are reduced in a hierarchical structure so it is considered in the group of data reduction techniques. The centroids are obtained through the hierarchical k-means structure consisting of several layers, the first of which includes the main dataset. The next layer includes smaller datasets created through models similar to the main dataset. In other words, the centers of each cluster represent the data of that cluster and are transferred to the next level. Computationally compact, hierarchical algorithms are usually used as a baseline or to optimize the basic clustering algorithm. They fail to yield very good results for large-scale computations.

The I-K-means-+method has been presented in 2018 with an iterative approach for increasing the quality and speed of clustering resulting from K-means and K-means++ through removal of one cluster and division of another [21]. Clusters are removed and added based on comparisons of the profit and cost functions, as presented in the paper. Thus, the phase is gone through for pairs of clusters for which the profit obtained from division is more than the cost of removal. A problem with the l-k-means++ algorithm is that it spends too much on these computations, which slows down the execution.

Brown et al. designed an algorithm in 2019 for increasing the speed of DBSCAN when run over big data [22]. The algorithm first divides the data space to a network structure and then assigns a density value to each network cell. Then, it examines the network spaces to find the neighboring spaces and specify with the densest neighbor in each space. Next, *i.e.,* in the cluster formation step, the algorithm forms a chain

of the densest neighbors and continues this so that the clusters develop. For this purpose, it considers the network parts and adds them and their densest neighbors to the cluster. A great part of the execution time of the algorithm is spent finding the densest neighbors, which is regarded as a drawback of the algorithm.

The HCA-DBSCAN algorithm, presented in 2019, considers hypercubes like grids with diameters of ε over the data space [23]. The data in each hypercube are located in the same cluster. In each hypercube, a few points that are closest to the borders will be selected as agents. Only the agents of each pair of networks rather than all the points are examined and compared. If the distance between two agents from two grids is smaller than ε, the two networks are merged, and their data are placed in a single cluster. Using these agents, the number of comparisons that need to be made for calculation of the distances between data is reduced, and the algorithm is eventually run at a higher speed. Because of several search attempts of the indices carried out for obtaining the adjacent ones, this algorithm needs to spend a great deal of time, something which is regarded as a drawback.

An idea that was proposed in 2019 to implement DBSCAN on big data is a sampling method for decreasing data size [24]. This paper proposes two methods for better sampling in DBSCAN; as a result, the DBSCAN runtime decreases due to reducing the number of samples. One of the methods is a modified version of the Rough DBSCAN, whereas another method is a heuristic technique.

Yewang Chen et al. proposed an approximate method named KNN-BLOCK DBSCAN for big data clustering in 2019 [25]. For this purpose, they employed a KNN technique named FLANN to identify core blocks (CBs), noncore blocks (NCBs), and noise blocks (NOBs) which include main points, border points, and noise points, respectively. The density-reachable CBs were then merged, whereas NCBs were allocated to an appropriate cluster. They also proposed a grid-based clustering method named BLOCK-DBSCAN in two versions (L_∞ and L_2) for high-dimensional big data in 2020 [26]. Using a computation reduction method, this idea proposes two techniques. The first technique is named $\frac{\epsilon}{2}$–norm ball and employed to identify the inner core blocks in which all points are the core points. The second technique is a high-speed approximate algorithm that predicts whether two inner core blocks are density-reachable. In addition, a cover tree was employed to accelerate density computations for unvisited points. An advantage of this method is that it can cluster high-dimensional data relatively fast.

2.2 Clustering methods based on multiple machines

In [27], published in 2014, attempts were made to improve big data clustering through the K-means algorithm, and the MapReduce technique was used for this purpose. This paper focused on the optimization of selection of initial clustering centers to increase accuracy and speed. For that purpose, a number of subsets were selected from the big data through sampling. The subsets were processed in parallel to obtain the centers that could be used for clustering the main dataset. Integration took place after the required samplings were performed. In this method, K-means++ was used for the selection of the initial centers.

In [28], presented in 2016, the dataset was divided into smaller parts and distributed to several nodes in a cluster of machines. Apache Hadoop was used as a scalable, powerful platform for this purpose. K-means is implemented on machines in two phases, in the first of which initial clusters are created by selecting a large number of clusters. For this purpose, probabilistic clustering is employed to select better initial centroids and reduce the number of convergence periods. In the second phase, a threshold is used to merge the clusters obtained from the first phase. The runtime of this method is longer than those of the novel methods proposed above. In fact, it is nearly equal to that of k-means, something which is considered a disadvantage.

Song et al. proposed a plan for running DBSCAN in parallel in 2018. It divided small cells of data stochastically into parallels [29]. They presented a new parallel DBSCAN algorithm, known as RP-DBSCAN, in which they used the plan of stochastic division of data and a two-level cell dictionary that was a compressed summary of the entire dataset. The algorithm simultaneously found local clusters for each part of the data and then merged them for obtaining the final clustering. RP-DBSCAN was implemented within the Spark framework in Microsoft Azure. Spatial division costs are reduced due to the use of random data distributions; however, this prolongs the runtime.

Based on neighborhood similarity, a modified DBSCAN was introduced by [30] in 2020. Using the cover tree method, this algorithm retrieves the parallel neighbors of every point and filters many unnecessary computations for distance measurement through the triangular inequality principle. The noticeable advantage of this algorithm is its high speed. In fact, it classifies high-density areas as one category and finds the desired spatial clustering quickly. Since this method is parallel, if such techniques as map reduction or Spark had been used, the proposed algorithm would have been executed much faster.

3 Proposed algorithm

The structure of the proposed method is broadly composed of three steps: grouping the data using the K-means++ algorithm, running the DBSCAN algorithm over the data of each group in the previous step, and merging the clusters generated in different groups. We have named the proposed algorithm K-DBSCAN because the data are divided at the beginning into K groups using the K-means++ algorithm, and the DBSCAN algorithm is then employed in each group. Each step of the algorithm is separately detailed in the following.

3.1 First step: grouping the data through the K-means++ algorithm

In this step, the K-means++ algorithm is executed on the entire dataset to divide the data into several smaller parts. Since the DBSCAN algorithm functions according to the calculation of the density of each piece of data and aims to place data located close to each other in the same cluster, like other clustering methods, data cannot be divided arbitrarily or stochastically to obtain proper efficiency. This paper employed

the K-means++ algorithm, which is a clustering algorithm. The results demonstrate that running K-means++ in a small number of iterations can also provide a proper division over the data, a finding which also exhibits an acceptable balance.

The only difference between K-means++ and ordinary K-means is that the initial centers are selected more intelligently in the former than in the latter; therefore, it can propose a more appropriate division. As mentioned earlier, the only purpose of applying K-means++ is to divide the data, a task which can be performed with no high computational burden imposed on the system even when the algorithm is run in a small number of iterations.

Like other clustering methods, the K-means++ algorithm clusters the data. In this paper, however, K-means++ clusters are named along with the group titles, not to be mistaken for clusters obtained from DBSCAN.

3.2 Second step: running the DBSCAN algorithm in each group

In this step, DBSCAN is executed separately on the data in each of the K-means++ groups. To determine whether a point is a core in DBSCAN, its distances to all the other points in the dataset must be measured. A great deal of computation is made simply to obtain the number of points located at a distance of ε. Therefore, dividing data into smaller groups and using DBSCAN separately in each group can help considerably reduce computations required for measurement of distance to the other points. As also clear from Fig. 1, the distances from a point to all the others in the dataset must be measured to determine whether it is a core.

The dashed circle represents a range with a radius of ε. The grouping provided by K-means++ causes the heavy computation particularly on large datasets to be confined to groups rather than take place over the entire dataset. This computational pruning greatly reduces the computational complexity resulting from the detection of core points and formation of clusters.

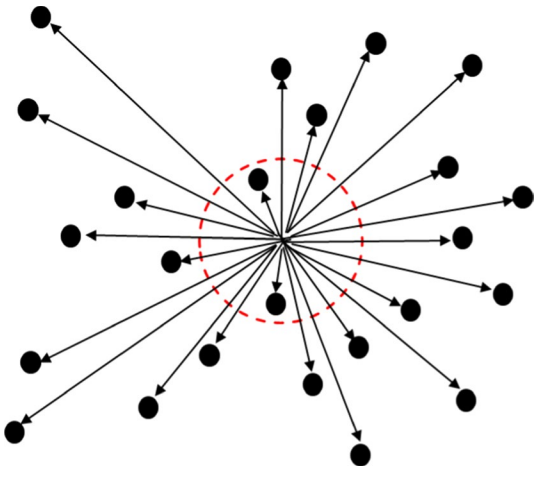


Fig. 1 Calculation of the distances from each point to all those in the dataset for detection of core data

For instance, if we divide a dataset consisting of one million pieces of data into ten groups through the K-means++ , only a hundred thousand calculations will be made on average rather than calculation of one million distances when DBSCAN is executed to determine whether each instance of data is a core. This reduces the number of calculations by 1/10. It should be determined now if there are DBSCAN clusters in adjacent K-means++ groups must be merged. Since DBSCAN regards each group separately and carries out clustering considering only the data in the same group, the clusters in one group need to be examined along with those in the adjacent groups. Thus, if there were no borders between groups, and DBSCAN could see the data in the adjacent groups when carrying out clustering, would the data on the group borders be located in the same cluster? In other words, would border clusters merge?

3.3 Third step: merging the selected border clusters

In this step, it is to be examined whether the clusters obtained from the DBSCAN algorithm would still look as they do if there were no group borders between data, or the border clusters would merge to one. To reduce the computation in this step, in this step we also propose two rounds of pruning for cases that need to be examined. The first round involves the analysis of distances between the K-means++ groups and application of pruning to groups long distances (more than ε) between which makes it impossible to merge the internal clusters. The second round involves pruning carried out when the internal clusters of the two groups are examined for merging and removing clusters that are impossible to merge. After these two rounds of pruning are performed, the selected clusters of groups are merged by the DBSCAN algorithm, re-executed over the data in these clusters.

This section addresses a theorem, on the basis of which the proposed method and the pruning to be proposed below function.

Theorem *If the distance between two groups of data, calculated with the single-link method, is longer than ε, the two groups will not form one cluster after the DBSCAN method is applied.*

Proof The single-link method is employed to determine the distance between two datasets based on the distance between the nearest points of those datasets. Therefore, if the distance between the two sets exceeds ε, no points of the second set will definitely be placed in the ε neighborhood of points from the first set, and vice versa. Obviously, it will be impossible to create a cluster if DBSCAN is implemented on the data of these two sets.

The procedure for merging the designated border clusters is then explained in three steps. Distances between sets are determined in the first step, and the sets are pruned if their distances exceed ε. This is the first pruning process in this step. After that, distances between the clusters existing in the sets remaining from the first pruning process are calculated two by two. According to the proposed theorem, if the

distance between two clusters exceeds ε, the second pruning process occurs. Finally, DBSCAN will be executed again of the data of the designated clusters in the third step.

First Step: Examination of the Distances between groups. This pruning is applied to the K-means++ groups before the possibility of merging clusters is examined. Based on the idea proposed in this paper, the merging possibility needs to be examined for clusters located in adjacent groups. Clearly, it is impossible to merge the clusters of pairs of groups that are far from rather than in the neighborhoods of each other, and these groups can be excluded from the process of computation for examination of the possibility of merging group clusters.

Therefore, the groups obtained from the K-means++ algorithm are considered two by two, and the following formula is calculated for them.

$$dis\_K_{ij} = dis\_KC_{ij} - \left(dis\_r_i + dis\_r_j\right). \tag{1}$$

In Eq. (1), dis_KC_{ij} represents the distance between the centers of the two groups in K-means++ and dis_r_i and dis_r_j indicate the distances between the centers of the two groups and the farthest-away points on them, correspondingly. The equation obtains the distance between the two groups, *i.e.,* dis_K_{ij}, considering the worst case. These parameters can be calculated and stored for each group without considerable computation when the K-means++ algorithm is applied. The distances are illustrated schematically in Fig. 2. If the value of dis_K_{ij} is greater than ε, according to the theorem it will definitely be unlikely to merge the border clusters of the two groups. Thus, there is no need for the internal clusters of all the groups to be examined for being merged.

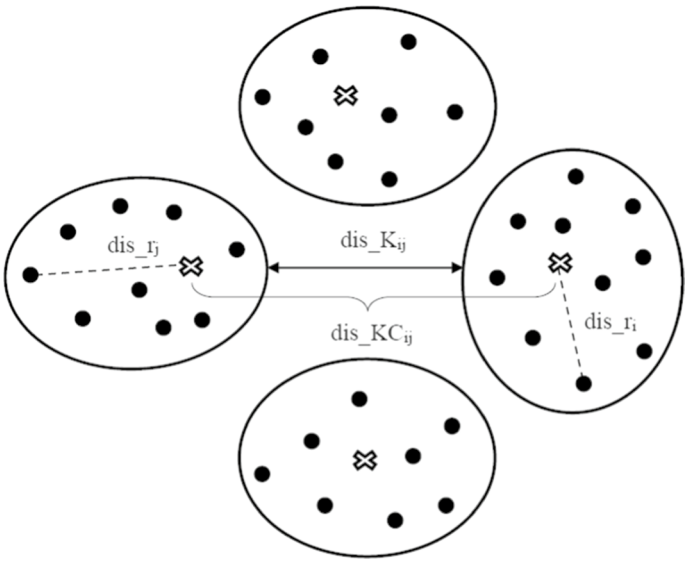


Fig. 2 An illustration of how the distance between two groups is calculated

Only groups located at distances less than or equal to ε from each other are examined. This condition causes many of the computations to be pruned. In general, this pruning is the second applied to the DBSCAN algorithm in the proposed algorithm after the data are divided into smaller groups.

Second Step: Analysis of the probability that DBSCAN clusters are merged. In this step, the possibility of merging the internal clusters is analyzed for the groups not pruned in the previous step. The distances between the clusters need to be obtained for finding those that can be merged. If the distance is smaller than or equal to ε, merging can take place; otherwise, the probability is zero, and the data concerning those clusters are not considered in the merging phase.

For application of this pruning, a center needs to be considered for each DBSCAN cluster, calculated based on the mean data in the cluster. It should be noted, of course, that the center is not calculated for all the clusters and that it does not impose high computational complexity as compared to the computation that is pruned later.

Another parameter that is needed for application of the pruning is the distance between the cluster center and the farthest-away piece of data in the cluster. The parameter can be calculated with a slight modification in the DBSCAN algorithm and no considerable computational burden imposed. As stated earlier, DBSCAN begins at a random point and examines whether or not that point is a core. If the point is identified as a core, the algorithm repeats the procedure at the points at a distance of ε from the core. That is, first the data located at a maximum distance of ε and then all those at a distance of 2ε are examined. The process continues until the entire cluster is specified. While running the algorithm, however, one can store the distance the data located at which are being calculated, and this can therefore be the upper bound for the distance between the starting point and that farthest away from it.

Figure 3 shows a DBSCAN cluster. Assume that cluster formation has begun stochastically at point a. The farthest point away from a is named d, which is definitely also a border point.

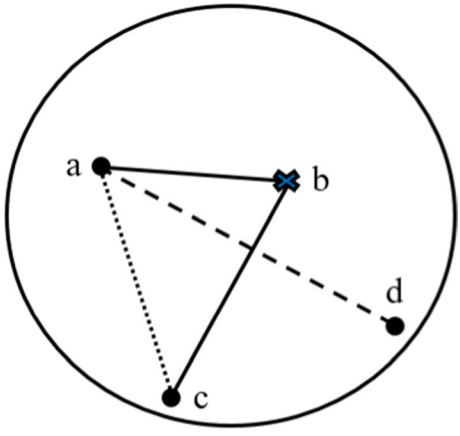


Fig. 3 An illustration of how the distance between the cluster center and farthest-away point (bc) is calculated

Based on the process stated above, the upper bound of d can be calculated upon application of DBSCAN with an inconsiderable computational burden (ad is indicated as a dashed line).

To calculate the distance between the cluster center b and the point farthest away from it, represented as c, we draw a line from a to c, and form the abc triangle. Given the above triangle and the triangle inequality theorem, Eq. (2) holds as follows:

$$bc < ab + ac. \tag{2}$$

If we assume the cluster center to be b, we can calculate the distance between a and the center of b. Moreover, since d is the farthest point from a, the following holds.

$$ac \leq ad. \tag{3}$$

Due to the unknown value of ac and for avoidance of extra computation, Eq. (2) can be rewritten as follows based on Eq. (3)

$$bc < ab + ad. \tag{4}$$

It is clear from Eq. (4) that the value of bc is definitely less than the sum of ab and ad. Therefore, the upper bound for the value of bc, *i.e.,* the distance between the cluster center and the farthest-away piece of data in the cluster, can be calculated based on Eq. (4).

Considering two clusters from two groups, as in Fig. 4, we use Formula (5) to obtain the distance between the two clusters.

$$dis\_D_{\mathrm{ij}} = dis\_DC_{\mathrm{ij}} - \left(dis_{\mathrm{rj}} + dis_{\mathrm{rj}}\right). \tag{5}$$

In the above equation, dis_r_i and dis_r_j represent the distances between the cluster centers and the points farthest away from them in the same clusters, dis_D_{ij} indicates the distance between the two DBSCAN clusters, and dis_DC_{ij} denotes the distance between the centers of the two DBSCAN clusters in the two K-means++ groups. If the value of dis_D_{ij} is less than or equal to that of ε, the two clusters can be merged, and they will be pruned otherwise.

Third step: Merging Border Clusters by Rerunning the DBSCAN algorithm. After the possibility of merging clusters is determined, and the proposed pruning is carried out, the data concerning those that can be merged are considered together, and the DBSCAN algorithm is reapplied to them regardless of any borders. Then, either the clusters continue to hold the same arrangement or two or more of them are merged into one cluster. Thus, the entire data space is finally clustered by the DBSCAN algorithm, except that the computations that are made are much less complex than the heavy computations in conventional DBSCAN. For a better understanding of the proposed method, the pseudocode of the algorithm is shown in Fig. 5.

Calculation of Time Complexity. The time complexity of the proposed approach is calculated by considering the complexity of all its three steps. Equation (6) states the time complexity of the proposed approach as a combination of the complexity of

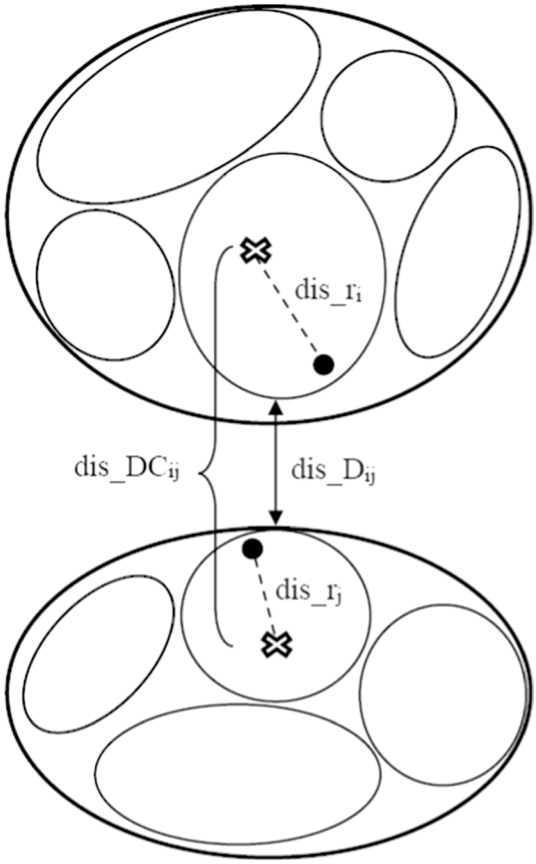


Fig. 4 An illustration of how the distance between two clusters is calculated

these steps. Since the distances from k center points to n points need to be obtained in the K-means++ algorithm, and the computations are iterated Iter times, the time complexity of the algorithm is O(Iter**k***n*). The number of iterations, the number of initial centers, and data size affect the execution time of the algorithm. The time complexity of the DBSCAN algorithm is also O(n^2) because the distances between the points in n pairs must be obtained.

Given the steps suggested for the proposed method and the algorithm developed for the method, the time complexity of the proposed method is as stated in Eq. (6), which is composed of three parts.

$$O\left((Iter * k * n) + \left(\frac{n}{k}\right)^2 + \frac{k(k-1)}{2}m^2\right). \tag{6}$$

Since the K-means++ algorithm is first run in the proposed method, and the data are grouped, the first part of time complexity is that of K-means++ . The second part concerns the DBSCAN algorithm, which is applied separately to the data available in k groups. The third part of the time complexity addresses the

```
Algorithm: K-DBSCAN
Input: Data set D, number of K-means clusters k, number of K-means iteration
MaxIter, eps, MinPts.
Output: A set of clusters.
– – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –
/* Step 1: Normalization of D */
D = Normalization(D);

/* Step 2: Data grouping with K-means Algorithm */
clu = K-means(D, k, MaxIter);  /* clu is set of K-means clusters (groups) */

/* Step 3: Run DBSCAN Algorithm in each group */
For i=1:k
   IDX = DBSCAN(clu, eps, MinPts);  /* IDX is set of DBSCAN clusters */
   numclu(i) = number of DBSCAN clusters in group i
End

/* Step 4: Merge border clusters */
Counter = 1;
For a=1:k-1
   For b=1:k
      dis_k = distance between two group
      If dis_k <= eps
         For c=1: numclu(a)
            For d=1: numclu(b)
               dis_db = distance between two clusters
               If dis_db <= eps
                  If flag(ac) == 0 && flag(bd) == 0  /* flag(ac) means flag of cluster c  in group a */
                     flag(ac) = counter;
                     flag(bd) = counter;
                     counter = counter+1;
                  Else
                     flag(ac) = max(flag(ac), flag(bd));
                     flag(bd) = max(flag(ac), flag(bd));
                  End
               End
            End
         End
      End
   End
End
For i=1:counter-1
   Run DBSCAN algorithm in clusters with flag i
End
```

Fig. 5 Pseudocode of the proposed algorithm (K-DBSCAN)

final step in the proposed method. In this step of the presented method, the clusters available in the groups are searched two by two for selection of clusters that can be merged. Then, DBSCAN is applied to the data in these selected groups;

the total number of data of these selected groups is considered m. Clearly, Eq. (7) holds.

$$m \leq \frac{n}{k}. \tag{7}$$

It is clear that the time complexity of the presented method is less than that of DBSCAN, and execution time is greatly reduced when the proposed algorithm is run over different datasets, particularly on big data.

Since we consider the longest distances for the pruning suggested by the method proposed in this paper, we actually proceed according to the worst possible conditions. Clearly, therefore, less pruning may be carried out based on the computations suggested in this section, which increases the computations a bit, but there are no two clusters of groups that are pruned incorrectly if merged. Therefore, the proposed method can be regarded as belonging to the class of exact methods proposed for big data.

4 Experiments

In this section, we state the results obtained from the proposed algorithm and compare them to the performance of the conventional DBSCAN [10], Density-Grid [22] and HCA-DBSCAN [23] and along with an examination and evaluation.

4.1 Datasets

The presented method has been experimented with over nine real-world datasets selected from the UCI [34] and GitHub [35] websites. For a more comprehensive, more accurate evaluation, attempts have been made to select datasets with various data sizes and numbers of features. Complete information on the utilized datasets is given in Table 1. Some of the sets contain imperfect data, which have been removed and ignored before the experimentation.

Table 1 Specifications of the datasets

Dataset	Number of data points	Number of dimensions	ε	MinPts
Abalone	4177	8	0.035	4
MAGIC	19,020	10	0.06	4
Statlog (Shuttle)	58,000	9	0.02	5
MoCap Hand Postures	78,095	11	0.007	4
FARS	106,565	10	0.08	4
Skin Segmentation	245,057	3	0.01	4
3D Road Network	434,874	4	0.01	4
Cover type	581,012	5	0.01	4
Poker hand	1,000,000	10	0.25	8

Table 2 shows the parameters in the proposed method and the values that they are assigned. The effect of each parameter on the proposed algorithm has been examined, and the most appropriate values for the parameters have been specified.

Neighborhood radius (ε) and minimum data size within the neighborhood radius (MinPts) have been set for each dataset based on its features, but the assigned values have been assumed to be the same for all the methods, so that the results of their clustering can be compared.

We have assumed the number of iterations to be 4 for the K-means++ algorithm. The value has been proposed based on the experimental results and the purpose of the proposed method of applying K-means++ , which is simply to divide the data. The increase in the number of iterations has only increased execution time, with no considerable effects on quality. Moreover, since the initial centers have been selected using the K-means++ algorithm, it has been decided from the very beginning that they should be far apart from each other for the space to be divided ideally.

After the effect of k on the execution time of the proposed algorithm was analyzed, it was found that k played a determining role in reduction of execution time. In other words, it can be stated that the greater the data size, the greater the value of k needs to be for a higher algorithm execution speed. The best value of k for each dataset can naturally be obtained through trial and error and examination of different values, but we have considered Eq. (8) for specification of the values of k given the conducted experiments. In this equation, n represents the number of samples from the dataset.

Clearly, the greater the value of k for the initial grouping, the smaller the size of the groups on which DBSCAN is to be run, and the much lower the intra-group computation resulting from the application of DBSCAN.

$$k \approx \sqrt{n} \tag{8}$$

4.2 Evaluation

In addition to time which is an important criterion for comparing the proposed algorithm to other relevant algorithms, clustering quality in the proposed method should be compared to that in the other methods. The following two metrics have been used for the evaluation of clustering quality in the proposed algorithm [31].

Table 2 Specifications of the parameters

Notation	Description	Value
$\boldsymbol{n}$	Number of data	Variable
ε	Neighborhood radius	Variable
MinPts	Minimum number of data within the neighborhood radius	Variable
Iter	Number of iterations in the initial grouping	4
k	Number of initial groups	$\sqrt{n}$

Davies–Bouldin (DB) Metric [32]. This metric is defined as follows based on within-to-between cluster distance ratio. The metric is defined as follows:

$$DB = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left\{D_{i,j}\right\}. \tag{9}$$

In this equation, k represents the number of clusters and $D_{i,j}$ indicates within-to-between cluster distance ratio, which is defined as follows:

$$D_{i,j} = \frac{\left(\overline{d_i} + \overline{d_j}\right)}{d_{i,j}}. \tag{10}$$

Here, $\overline{d_i}$ denotes mean distance between all the pieces of data in cluster i and the center of cluster i. $\overline{d_j}$ represents mean distance between all the pieces of data in cluster j and the center of cluster j. $d_{i,j}$ is Euclidean distance between the centers of clusters i and j. A smaller value for the DB metric shows higher clustering quality.

Silhouette Metric [33]. The silhouette value for any data sample is a metric that measures the similarity of the data sample to samples from the same cluster and its difference from samples from other clusters. The value of this metric, abbreviated as SI, for the ith data sample is defined as follows.

$$SI_i = \frac{b_i - a_i}{\max\left(a_i.b_i\right)}. \tag{11}$$

Here, parameter a_{i} indicates mean distance between the ith piece of data from one cluster and the other pieces of data in that cluster and parameter b_{i} represents minimum mean distance between the ith piece of data and the pieces of data in the other clusters. a_i is calculated as follows:

$$a_i = \frac{1}{n_i}\sum_{i=1}^{n_i} d_{i,j}. \tag{12}$$

n_i denotes the number of pieces of data available in the ith cluster, and $d_{i,j}$ is the distance between the ith piece of data to the jth from the same cluster. Moreover, $\mathrm{b_i}$ is obtained as follows:

$$b_i = \min_{1<l<k}\frac{1}{n_l}\sum_{j\,\int\, l} d_{i,j}. \tag{13}$$

n_l stands for data size in the lth cluster, and $d_{i,j}$ is the distance between the ith piece of data to the jth in cluster l. SI is a value between -1 and 1. The closer the value to 1, the better the clustering, and the closer the value to -1, the worse the clustering.

For the evaluation of the performance of the proposed method, the results have been compared to those of the DBSCAN, HCA-DBSCAN, and Density-Grid algorithms. All the four algorithms have been implemented on a quad-core computing

machine with a disk size of 40 gigabytes and 64 gigabytes of RAM. It should be mentioned that no other programs have been in progress during the runs, and unnecessary services of the operating system have also been deactivated.

The presented algorithm and three others have been run with the same parameter values over the nine introduced datasets. When run over these datasets, the DBSCAN algorithm has achieved the final solutions only on the first five datasets and has encountered memory shortage for the rest, with larger sizes. The algorithm proposed in this paper, however, has successfully clustered all the datasets in shorter times. For reducing the errors as far as possible and obtaining more accurate results, all the four algorithms have been run ten times over each dataset, and the mean solutions have been considered as the final ones. Table 3 shows the results obtained from the proposed algorithm, DBSCAN, HCA-DBSCAN, and Density-Grid.

According to this table, the proposed method managed to considerably reduce the time used by DBSCAN, something which is more evident for larger datasets. At the same time, it provided relatively proper, desirable results like DBSCAN in terms of quality. The slight decrease in quality observed in our method can be caused by the ignorance of the noise that occurs as the border clusters are merged. This does not affect the final results greatly. Furthermore, the execution time of the HCA-DBSCAN and Density-Grid algorithms is much shorter than that of DBSCAN but longer than that of the proposed algorithm.

4.3 Analysis of results

Based on the obtained times reported in Table 3, the percentage of performance improvement (PPI) for the proposed algorithm in terms of execution time with respect to the DBSCAN method can be calculated through Eq. (14).

$$PPI = \frac{|t_{DBSCAN} - t_A|}{t_{DBSCAN}} \times 100. \tag{14}$$

In this equation, t_{DBSCAN} indicates the DBSCAN execution time and t_A represents time of the proposed algorithm.

Table 4 shows the improvement made by the proposed algorithm in the DBSCAN execution time. According to this table, the larger the data size, the higher the performance of the presented algorithm and the rate at which it reduces execution time.

Figures 6 and 7 show the information concerning the first five datasets available in Table 3. Figure 6 contains four bar charts for the comparison of the performance of the proposed algorithm to those of DBSCAN, HCA-DBSCAN, and Density-Grid. The four methods were examined in separate diagrams in terms of the DB and SI metrics.

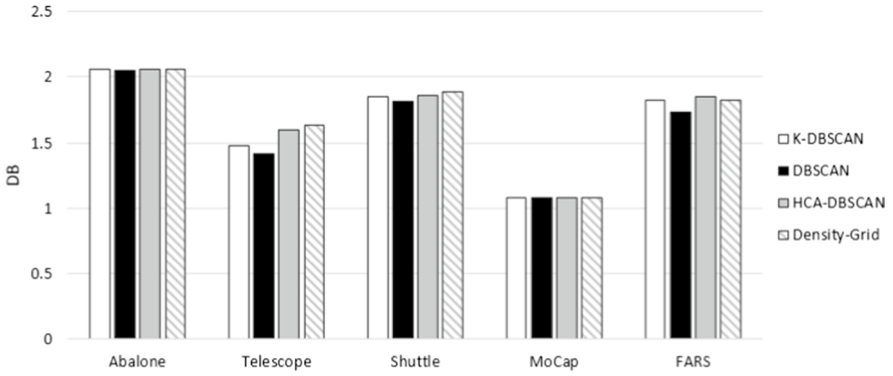
Figure 7 depicts the diagrams involving the execution times of the algorithms in terms of data size in the first five datasets. The green line concerns the proposed method, whereas the blue line indicates the execution time of the HCA-DBSCAN algorithm, the yellow line concerns the Density-Grid, and the red line shows the incremental trend of the execution time obtained from the DBSCAN algorithm.

Table 3 Comparison of the results of the K-DBSCAN algorithm to those of DBSCAN, HCA-DBSCAN, and Density-Grid

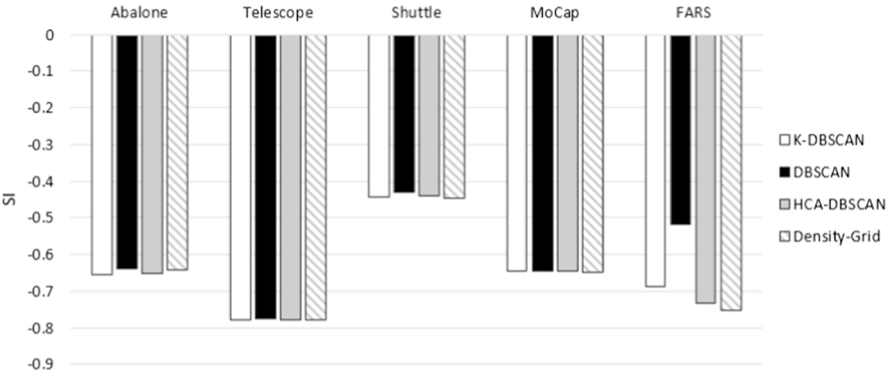
Dataset Evaluation criteria		Abalone	MAGIC	Shuttle	MoCap	FARS	Skin segmentation	Road Network	Cover type	Poker hand
DB	K-DBSCAN	2.0626	1.4795	1.8535	1.0783	1.8238	1.3912	1.2299	1.142	1.2381
	DBSCAN	2.0524	1.4197	1.8097	1.0775	1.7289	–	–	–	–
	HCA-DBSCAN	2.0560	1.5963	1.8602	1.0779	1.8516	1.4283	1.1746	1.132	1.2506
	Density-Grid	2.0615	1.6359	1.8823	1.0786	1.8235	1.4681	1.2194	1.1415	1.2578
SI	K-DBSCAN	− 0.655	− 0.778	− 0.443	− 0.646	− 0.688	− 0.146	− 0.535	− 0.714	− 0.833
	DBSCAN	− 0.638	− 0.776	− 0.431	− 0.645	− 0.519	–	–	–	–
	HCA-DBSCAN	− 0.651	− 0.778	− 0.439	− 0.646	− 0.731	− 0.148	− 0.541	− 0.712	− 0.832
	Density-Grid	− 0.642	− 0.779	− 0.445	− 0.647	− 0.751	− 0.148	− 0.544	− 0.712	− 0.834
Time (s)	K− DBSCAN	0.4537	6.2015	82.893	177.96	299.15	462.47	806.19	733.51	1789.7
	DBSCAN	0.3195	4.4409	128.28	3775.8	16,950.74	–	–	–	–
	HCA-DBSCAN	0.3726	4.0123	104.95	586.70	1193.5	2058.1	3967.5	4108.3	12,413.42
	Density− Grid	0.3905	5.5327	114.68	672.98	1627.6	3575.4	4756.7	4825.7	13,526.63

Table 4 Percentage of improvement made by the proposed algorithm in the execution time of DBSCAN

FARS	MoCap	Shuttle	Telescope	Abalone	PPI	Dataset
98.23%	95.28%	35.38%	39.64%-	42.00%	PPI	



(a) Bar diagrams of the datasets in terms of the DB metric



(b) Bar diagrams of the datasets in terms of the SI metric

Fig. 6 Diagrams of the two qualitative metrics DB and SI for comparison of the proposed method, DBSCAN, HCA-DBSCAN, and Density-Grid. a) Bar diagrams of the datasets in terms of the DB metric. b) Bar diagrams of the datasets in terms of the SI metric

According to this figure, the slopes of the diagrams for the presented method, HCA-DBSCAN, and Density-Grid exhibit a very slow incremental trend, whereas the execution time increased with a steep slope in the DBSCAN method as data size increased. This indicates the low execution speed of the algorithm, especially for big data.

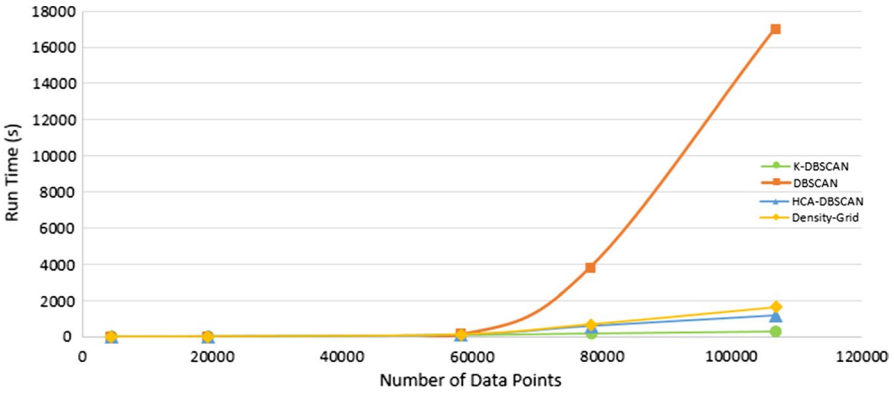


Fig. 7 Diagram for comparison of execution time in the proposed method, DBSCAN, HCA-DBSCAN, and Density-Grid

5 Conclusions

This paper proposed an algorithm to improve the execution speed of the DBSCAN algorithm using the idea of reducing computation by dividing the workspace into a number of separate areas so that it would be possible to run the algorithm on big data. Among the several algorithms existing for data clustering, the K-means++ algorithm is popular in data mining. It can find data groups in the data space and separate them. This algorithm can provide proper space division over the entire dataset and result in balance. Hence, the K-means++ algorithm was utilized to divide the DBSCAN workspace. It exhibited proper convergence speed and provided proper data division.

Thus, the data were first grouped by the K-means++ algorithm with a small number of iterations applied. Then, the DBSCAN algorithm was applied separately in each part, and the data were clustered. Finally, the results were integrated, and the different parts were merged. Since DBSCAN was applied separately in each group, the considered data space was smaller, and there were consequently a smaller number of points, the distances of which were calculated. This reduced execution time significantly. According to the results of implementing the proposed algorithm and the DBSCAN, HCA-DBSCAN, and Density-Grid algorithms, the proposed algorithm clustered data successfully in a shorter period of time with no considerable changes in the other evaluation criteria.

An advantage of our method is that it does not require many hardware resources and can be implemented on a single machine. It would be particularly efficient to use the proposed algorithm to deal with a lack of powerful equipment and hardware resources. Furthermore, the use of the proposed algorithm could preserve plenty of resources and time in the system and reduce its depreciation in the long run by avoiding unnecessary system activities.

6 Future works

In computations, the proposed method ignores noise so that border clusters can be merged. If the method can be improved to include noise, clustering quality will increase.

As stated in earlier sections, the algorithm corresponding to the idea presented in this paper is based on one machine. However, the idea features the ability to be parallelized and run on multiple machines. The execution time of the method can, therefore, be reduced even further. The DBSCAN algorithm is executed on each data group in one step of the algorithm, which can be taken in parallel with the others. For this purpose, the data in each group can be given to a separate machine through the MapReduce technique for the application of DBSCAN to groups after the initial grouping.

References

1. Storey V, Song I (2017) Big data technologies and management: What conceptual modeling can do. Data KnowlEng 108:50–67
2. Ianni M, Masciari E, Mazzeo G, Zaniolo C (2018) Efficient big data clustering. In: 22nd International Database Engineering & Applications Symposium, pp 103–109. ACM
3. Arora P, Deepali D, Varshney S (2016) Analysis of K-Means and K-Medoids algorithm for big data. ProcediaCompuSci 78:507–512
4. Jain A, Murty M, Flynn P (1999) Data clustering: a review. ACM ComputSurv 31(3):264–323
5. Zhu J, Zeng M, Huang J, Liao S, Cai C, Zheng L (2020) Vehicle re-identification using quadruple directional deep learning features. IEEE Trans IntellTranspSyst 21(1):410–420
6. Liu S, Liu M, Li P, Zhao J, Zhu Z, Wang X (2017) SAR image denoising via sparse representation in Shearlet domain based on continuous cycle spinning. IEEE Trans Geosci Remote Sens 55(5):2985–2992
7. Pei S, Shen T, Wang X, Gu C, Ning Z, Ye X, Xiong N (2020) 3DACN: 3D augmented convolutional network for time series data. InfSci 513:17–29
8. Qiao S, Li T, Li H, Peng J, Chen H (2012) A new blockmodeling based hierarchical clustering algorithm for web social networks. EngApplArtifIntell 25(3):640–647
9. Che D, Safran M, Peng Z (2013) From big data to big data mining: challenges, issues, and opportunities. In: International Conference on Database Systems for Advanced Applications, pp 1–15. Springer
10. Ester M, Kriegel H, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: 2nd International Conference on Knowledge Discovery and Data Mining, pp 226–231.
11. David A, Sergei V (2007) k-means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pp 1027–1035. ACM
12. Katal A, Wazid M, and Goudar R (2013) Big data: Issues, challenges, tools and good practices. In: 2013 6th International Conference on Contemporary Computing, pp 404–409. IEEE
13. Shahrivari S (2014) Beyond batch processing: Towards real-time and streaming big data. Computers 3(4):117–129
14. Chen M, Mao S, Liu Y (2014) Big data: A survey. Mob NetwAppl 19(2):171–209
15. Shirkhorshidi A, Aghabozorgi S, Wah T, Herawan T (2014) Big data clustering: a review. In: International Conference on Computational Science and its Applications, pp 707–720. Springer
16. LIU B (2006) A fast density-based clustering algorithm for large databases. In: 2006 International Conference on Machine Learning and Cybernetics, pp 996–1000. IEEE
17. Wu Y, Guo J, ZHANG X (2007) A linear DBSCAN algorithm based on LSH. In: 2007 International Conference on Machine Learning and Cybernetics, Vol 5, pp 2608–2614. IEEE

18. Dogan Y, Birant D, Kut A (2013) SOM++: Integration of self-organizing map and K-Means++ algorithms. In: International Conference on Machine Learning and Data Mining in Pattern Recognition, pp 246–259. Springer
19. Bakr A, Ghanem N, Ismail M (2015) Efficient incremental density-based algorithm for clustering large datasets. Alex Eng J 54(4):1147–1154
20. Xu T, Chiang H, Liu G, Tan C (2015) Hierarchical K-means method for clustering large-scale advanced metering infrastructure data. IEEE Trans Power Delivery 32(2):609–616
21. Ismkhan H (2018) I-k-means++: An iterative clustering algorithm based on an enhanced version of the k-means. PattRecogn 79:402–413
22. Brown D, Japa A, Shi Y (2019) A fast density-grid based clustering method Daniel Brown. In: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference, pp 0048–0054. IEEE
23. Mathur V, Mehta J, Singh S (2019) "HCA-DBSCAN: HyperCube accelerated density based spatial clustering for applications with noise," in *33rd Conference on Neural Information Processing Systems (arXiv preprint)*.
24. Luchi D, Rodrigues A, Varejao F (2019) Sampling approaches for applying DBSCAN to large datasets. Pattern RecognLett 117:90–96
25. Chen Y, Zhou L, Pei S, Yu Z, Chen Y, Liu X, Du J, Xiong N (2019) KNN-BLOCK DBSCAN Fast Clustering for Large-Scale Data. IEEE Transactions on Systems, Man, and Cybernetics: Systems, pp 1–15.
26. Chen Y, Zhou L, Bouguila N, Wang C, Chen Y, Du J (2020) BLOCK-DBSCAN Fast clustering for large scale data. PattRecogn 109:107627
27. Cui X, Zhu P, Yang X, Li K, Ji C (2014) Optimized big data K-means clustering using MapReduce. J Supercompu 70(3):1249–1259
28. Sinha A, Jana P (2016) A novel K-means based clustering algorithm for big data. In: Conference on Advances in Computing, Communications and Informatics, pp 1875–1879. IEEE
29. Song H, Lee J (2018) RP-DBSCAN: A superfast parallel DBSCAN algorithm based on random partitioning. In: 2018 International Conference on Management of Data, pp 1173–1187.
30. Li S (2020) An Improved DBSCAN Algorithm Based on the Neighbor Similarity and Fast Nearest Neighbor Query. IEEE Access 8:47468–47476
31. José-García A, Gómez-Flores W (2016) Automatic clustering using nature-inspired metaheuristics: A survey. Appl Soft Compu 41:192–213
32. Davies D, Bouldin D (1979) A cluster separation measure. IEEE Trans Pattern Anal Mach Intell 1(2):224–227
33. Rousseeuw P (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J ComputAppl Math 20:53–65
34. UCI. http://archive.ics.uci.edu/ml/index.php. Accessed 1 June 2020
35. GitHub. https://vincentarelbundock.github.io/Rdatasets/datasets.html. Accessed 1 June 2020