Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A fast DBSCAN algorithm for big data based on efficient density calculation

Nooshin Hanafi, Hamid Saadatfar *

Computer Engineering Department, University of Birjand, Birjand, Iran

ARTICLE INFO

Keywords:
Data Mining
Clustering
Big Data
DBSCAN Algorithm

ABSTRACT

Today, data is being generated with a high speed. Managing large volume of data has become a challenge in the current age. Clustering is a method to analyze data that is generated in the Internet. Various approaches have been presented for data clustering until now. Among them, DBSCAN is a most well-known density-based clustering algorithm. This algorithm can detect clusters of different shapes and does not require prior knowledge about the number of clusters. A major part of the DBSCAN run-time is spent to calculate the distance of data from each other to find the neighbors of each sample in the dataset. The time complexity of this algorithm is $O(n^2)$; Therefore, it is not suitable for processing big datasets.

In this paper, DBSCAN is improved so that it can be applied to big datasets. The proposed method calculates accurately each sample density based on a reduced set of data. This reduced set is called the operational set. This collection is updated periodically. The use of local samples to calculate the density has greatly reduced the computational cost of clustering. The empirical results on various datasets of different sizes and dimensions show that the proposed algorithm increases the clustering speed compared to recent related works while having similar accuracy as the original DBSCAN algorithm.

1. Introduction

Today, data is a valuable capital in the world. The volume of data generate every day is increasing dramatically. Big data and its analysis are of great importance. One of the essential issues about big data is the ability to process them. Clustering is a data mining and data analysis technique that aims to detect clusters and groups in a dataset. Clustering is a type of data modeling that originates from statistics and mathematics (Berkhin, 2006), and is classified as unsupervised learning methods. Clustering is applied in image processing (Hou, Liu, E, & Cui, 2016), medicine (Ananthi, Balasubramaniam, & Kalaiselvi, 2015), knowledge extraction (Feyyad, 1996), analysis of web social networks (Qiao, Li, Li, Peng, & Chen, 2012), and wireless sensor networks (Baranidharan & Santhi, 2016). The clustering algorithms are classified as hierarchical, partitioning, density-based, model-based, and grid-based (Kesavaraj & Sukumaran, 2013).

In recent years, much effort has been done to improve the performance of the existing algorithms to make them applicable for big data. The DBSCAN algorithm is a pioneer and well-known technique in density-based clustering (Ester, Kriegel, Sander, & Xu, 1996). This algorithm has some advantages over other classical clustering algorithms. Among advantages of this algorithm, ability to detect clusters of arbitrary shapes, efficient noise detection, and automatic detection of number of clusters can be mentioned. But one disadvantage of this algorithm is the high time complexity. It's low speed in encountering big data has attracted the attention of researchers to reduce the execution time of this algorithm in recent years. The run-time of the DBSCAN algorithm is significantly affected by finding the neighbors and obtaining density. Therefore, as the data size increases, this algorithm slows down and its run-time increases. Therefore, with the ever-increasing data volume, this algorithm should be improved so that it can operate on big data in a reasonable time with an acceptable quality. The current study aims to reduce the run-time of the DBSCAN algorithm, and in this regards, breaks down the dataset into operational and potential datasets. The operational dataset includes adjacent data with the sample whose density is being calculated. In fact, this set is explored to find the neighbors in the ε-neighborhood instead of exploring the whole data. Since the distance calculations are carried out with the samples in the operational dataset instead of the whole dataset, the required calculations for finding the neighbors are largely pruned. After scanning all samples of the operational dataset, the dataset should be updated. The proposed approach creates and updates the operational set with low computational overhead. Also, this update is done in such a way that the density of the samples is calculated accurately. The method proposed in

* Corresponding author.
E-mail addresses: nooshinhanafi@birjand.ac.ir (N. Hanafi), saadatfar@birjand.ac.ir (H. Saadatfar).

https://doi.org/10.1016/j.eswa.2022.117501
Received 27 December 2021; Received in revised form 10 April 2022; Accepted 1 May 2022
Available online 6 May 2022

this paper is classified as computation reduction methods, and can be applied to big datasets.

This paper is structured as follows: Section 2 reviews the previous studies in the context of big data clustering. The pros and cons of the previous methods are also examined in this section. Section 3 presents the proposed method for big data clustering; the algorithm is described and the related definitions are given. Section 4 presents the experiments setup include the datasets used in the experiments, evaluation conditions and qualitative evaluation metrics. Section 5 presents the evaluation results of the proposed method. At the end of this section, the proposed method is compared with other previous recent studies. Finally, the paper is concluded and some suggestions are given for future studies.

2. Literature review

With the development of web, social network, and mobile phones, there exists more data than before, and it is growing every day (Zerhari, Lahcen, & Mouline, 2015). Clustering is a tool used for big data analysis. The traditional clustering techniques cannot handle this large volume of data due to high complexity and computational costs (Mahesh, 2020). Therefore, the main purpose of the reviewed studies in recent years has been to increase the clustering speed. The DBSCAN algorithm (Ester, Kriegel, Sander, & Xu, 1996) is a pioneer technique in the context of density-based clustering. DBSCAN has several advantages over other classical clustering algorithms. Unlike supervised approaches (e.g., classification algorithms), clustering is an unsupervised technique that does not rely on any prior knowledge. The DBSCAN algorithm is a traditional density-based clustering method. This algorithm makes it possible to identify clusters of different shapes with the ability to manage noise patterns in the data. DBSCAN usually offers good results.

High time-complexity of the original DBSCAN algorithm makes it inefficient for high-dimensional databases of large volume. Various methods have been presented in recent years in order to improve the performance of the DBSCAN in handling big data.

In general, the fast clustering techniques presented for big data can be divided into two main groups (Shirkhorshidi, Aghabozorgi, Wah, & Herawan, 2014):

Single machine clustering techniques and multiple machine clustering techniques (parallel clustering algorithms). In the following, the papers and studies in the context of the two above techniques are reviewed. Considering these classes, the proposed method is a single machine clustering method. This class of methods is studied more accurately.

2.1. Single Machine clustering

This class of algorithms is implemented on a single machine and employs the computational power of one machine (Shirkhorshidi, Aghabozorgi, Wah, & Herawan, 2014). These algorithms are based on two main methods: data reduction techniques (reducing the number of samples or dimensions), and the techniques based on reducing the computations by approximating the computations or optimizing the algorithm itself. Some of these algorithms are described below.

The AnyDBC algorithm was presented in 2016 (Mai, Assent, & Storgaard, 2016). In this study, a novel approach, called anytime, has been presented to address the run-time of the DBSCAN algorithm, which reduces the range query and decreases the label propagation time of the DBSCAN.

The AnyDBC algorithm compresses data to smaller density-connected subsets, called primitive clusters, and labels the objects based on connected components of the primitive clusters to reduce the label propagation time. Also, the AnyDBC learns the current cluster structure of the data iteratively and actively instead of making range queries for all objects. It selects some of the best samples to filter the cluster in each iteration. Therefore, the number of queries is decreased significantly compared to the DBSCAN algorithm, and the clustering quality of the DBSCAN algorithm is preserved.

Brown et al. (Brown, Japa, & Shi, 2019) presented a method aiming to increase the processing speed for big datasets. This method reduces the number of computations because of using the grid concept and comprises three phases. In the first phase of this algorithm, the feature space of the dataset is divided in to a grid structure such that each data is located in the grid structure. The grid size is considered as the input parameter. Then, it determines which grid cell does the data belong to and the density of each cell is calculated separately. In the second phase, densest neighbor of each cell is specified. Finally, in the third phase, a chain of densest neighbors is formed to constitute a cluster. In this method, a large amount of time is spent to find densest neighbor of each cell, and the clustering quality in some datasets is reduced.

Hahsler et al. (Hahsler & Bolaos, 2016) proposed a method, called DBSTREAM to cluster data streams. The data stream is a sorted and infinite hierarchy of the data points. Since permanent storage of all data in the data stream and frequent access to them are impossible, and the shape and position of the clusters in the data stream changes, clustering algorithms specific for data streams are required. Most data stream clustering algorithms have an online and an offline phase. In the online phase, the data stream is summarized into a large number of micro-clusters in real time and in an online process. Micro-clusters represent a set of similar data points, and they are usually represented as the cluster center with the information, including data density and dispersion. Each new data that enters the system is allocated to the nearest micro-cluster according to the similarity function, and if it is not assigned to the existing micro-clusters, a new micro-cluster is developed. In the offline phase, by considering the center of the micro-clusters as the input points, a clustering algorithm is used to cluster the micro-clusters again. The distinction of this paper with previous papers is that this study considers the data density in the area between the micro-clusters and employs the shared density graph. Using shared density improves the clustering quality compared to other data stream clustering methods.

The G-DBSCAN algorithm (Kumar & Reddy, 2016) employs the Groups concept to speed up finding the nearest neighborhood process. The Groups concept develops a distinct graph-based structure on the data such that each vertex represents a group. There is one edge between two groups that are reachable for each other; the samples that are close to each other are integrated in a group. In this algorithm, each data sample in the dataset is classified as master or slave. The G-DBSCAN is implemented in two phases. In the first phase, the DBSCAN is implemented for a fast epsilon-neighborhood operation. The improper values of the parameter for constructing the hierarchical index reduce the performance compared to real-time implementation.

Another algorithm, called K-DBSCAN was presented in 2020 (Gholizadeh, Saadatfar, & Hanafi, 2020). This algorithm comprises three general steps. In the first step, the K-means++ algorithm is applied to the whole dataset, aiming to divide the data to smaller parts, where each part is called a group. In the second step, the DBSCAN is applied to each K-Means++ group independently. Dividing the data to smaller groups and applying the DBSCAN to each group separately can reduce the computations required to measure the distance from other points to detect if a point is the core. In the third step, the clusters created in different groups are integrated. In other words, this step examines if there are DBSCAN clusters in the adjacent K-means++ groups to be integrated or not. To reduce the computations resulting from finding the clusters that can be integrated, two prunes are presented for the cases that should be checked. The first prune examines the distance of K-means++ groups from each other and eliminates the possibility of integrating the internal clusters of the groups that their distance from each other is large. The second prune examines the internal clusters of two groups to integrate them or eliminate the ones that cannot be integrated. Finally, to integrate the selected clusters, the DBSCAN algorithm is implemented on the data of this cluster. One disadvantage of this method is that it ignores noises while integrating the border clusters, reducing the quality of the clustering process.

In (Gunawan & Berg, 2013), a method has been presented that creates a grid structure on the data and determines the membership of each sample to the grid cells that their size is $\sqrt{\varepsilon}$. If a cell has a minimum of MinPts samples, all samples of the cell are identified as core point (because the maximum distance in each cell is ε). If the samples of a cell are smaller than MinPts, instead of calculating the distance from all points of the database, it is sufficient to calculate the distance of each data sample from any data in a maximum of 21 neighboring cells. This technique reduces the computational costs, but it can only be applied to 2D data space.

The HCA-DBSCAN algorithm (Mathur, Mehta, & Singh, 2019) employs the grid-based approach in the dataset such that all points are in an area with a radius of epsilon. Therefore, if one of the points in the area belongs to a specific cluster, other points of the area also belong to that cluster. This key feature is used to achieve a significant computational speed compared to other improvements of the DBSCAN algorithm. Finally, by specifying the representative points and using the layering concept, the grid is scanned in depth and thus the required computational are reduced. Among advantages of this method is reducing the clustering run-time while preserving the clustering quality and accuracy. However, one disadvantage of this method is that its time order for low-dimensional data is O(nlogn) while its time complexity for high-dimensional data increases to $O(n^{3/2})$. In fact, its run-time for higher dimensional data increases.

In (Li, 2020), an improved DBSCAN algorithm based on neighbor similarity has been presented. Since the time-consuming part of the DBSCAN algorithm is finding the neighbors at the distance of ε for each data, this paper employs the cover tree to restore the neighbors of each data point in parallel. Also, it employs the triangle inequality to filter many of the unnecessary distance calculations, which reduces the distance calculation in the clustering process significantly.

This idea accelerates the original DBSCAN algorithm to a great extent while its results are still accurate. This algorithm comprises four phases. In the first phase, a hierarchical cover tree is created for the dataset. In the second phase, the neighbors of each data are initialized with null value and the unprocessed data are initialized with −2. In the third phase, a number of data among the unprocessed data is selected and a query tree is developed. Then, the cover tree is used to restore the nearest neighbor of each data. Considering the theories presented in the paper, a part of the outliers and the core points are identified without unnecessary search; this operation continues until all data points are identified. In the fourth step, all core points are integrated and different clusters are formed. Finally, in the fifth step, the border points are allocated to the nearest cluster.

The NQ-DBSCAN algorithm (Chen, et al., 2018) is a novel local search technique that reduces all unnecessary distance calculations efficiently accurate. This algorithm selects p random point from the dataset that has not been clustered. If the point p is identified as the core, it is extended to find all density-reachable points, and all these points are considered to belong to one cluster. Two theories have been presented to find the non-core points faster. Also, to find the core points, only the distance 2*ε is examined instead of the whole dataset to find the ε-neighborhood of the points that reduces the computation cost. The indexing method used in this paper is the quadtree hierarchical tree that cannot be applied to distributed datasets of high dimension.

The BLOCK-DBSCAN (Chen, et al., 2021) that was presented in 2021 obtains the correct clusters for big data, even high-dimensional data, effectively and quickly. In this method, additional distance calculations are filtered using two techniques. The first technique employs the ε/2 method to restore the internal core blocks faster. the second technique employs a fast and approximate method to find out if two internal core blocks are density-reachable or not. In addition to these two techniques, a cover tree is used for range exploration algorithm to speed up the density calculation process. The advantage of BLOCK-DBSCAN method is that it outperforms the NQ-DBSCAN and AnyDBC methods. The time complexity of the BLOCK-DBSCAN method in the worst-case is $O(n^2)$ and O(nlogn) in the middle case.

The h-DBSCAN (Weng, Gou, & Fan, 2021) presents a simple and fast method to improve the efficiency of DBSCAN algorithm. This method reduces the execution time in two aspects. The first one is to reduce the number of points presented to DBSCAN and second one is to apply the HNSW technique instead of the linear search structure.

In (Sanchez, Castillo, Castro, & Melin, 2014), a method for finding fuzzy information granules from multivariate data through a gravitational inspired clustering algorithm is proposed. The algorithm named FGGCA, incorporates the theory of granular computing, which adopts the cluster size with respect to the context of the given data. The FGGCA is an unsupervised clustering algorithm, mainly because the algorithm finds and suggests to a number of clusters.

IT2FPCM (Rubio, et al., 2017) is an extension of the Fuzzy Possibilistic C-means (FPCM) algorithm based on Type-2 Fuzzy Logic concepts to improve the efficiency of FPCM algorithm and to enhance its ability of handling uncertainty and make it less susceptible to noise. The parameters used in this article are not the optimal parameters ones.

The BIRCHSCAN (de Moura Ventorim, Luchi, Loureiros Rodrigues, & Miguel Varejão, 2021) presents a new method to apply DBSCAN to a reduced set of elements in order to cluster the entire dataset. This method samples from large datasets to obtain an approximate of the clustering solution for DBSCAN algorithm. The BIRCHSCAN consists of four steps. The initial step is responsible for clustering the data using BIRCH. The second step generates the sample by obtaining the centroids of the elements selected on the previous step. The third step runs the DBSCAN algorithm over this sample. The last step clusters the entire dataset. The main difficulty of applying BIRCHSCAN is regarded to the definition of the parameter δ which generates the value of threshold used in BIRCH that directly influences the sampling and it is not intuitively defined.

2.2. *Clustering based on multiple Machines*

This class includes the algorithms that are implemented on multiple machines and employs the computational power and resources of multiple machines. Parallel clustering and map-reduce based clustering are two main techniques of this class.

Sinha et al. (Sinha & Jana, 2016) have presented a method in which the initial dataset is divided into smaller sections, and distributed among multiple nodes of a cluster of computers. The K-means algorithm is implemented on two phases. In the first phase, by selecting a high value for the number of clusters, the initial clusters are created by the K-means algorithm. In the second phase, the center of the clusters obtained in the first phase, are integrated according to the formula presented in the paper and a predefined threshold. In other words, the distance between all centers of the clusters is calculated and compared with the threshold. If the distance between two or more centers is less than the threshold, they are integrated with each other and form a bigger cluster. Unlike the main K-means algorithm, in which the initial centers are selected randomly. This paper employs probability sampling for specifying the center of the clusters to select better initial centers such that the number of convergence round is decreased. Among disadvantages of this paper, high run-time compared to other methods can be mentioned.

The authors of (He, et al., 2011) have proposed a parallel DBSCAN algorithm, which its procedure is divided into four steps. In the first step, the total data record is summarized and grid-division is created. In the second step, the main DBSCAN is implemented for each subspace. The third step manages the cross-border issues while integrating the subspaces obtained from the previous step. It finds a list of clusters of the adjacent subspaces for each border that should be integrated. In the last step, cluster id mapping is first created for the whole dataset; then, the local id is substituted by a global code for all dataset points.

2.3. The DBSCAN algorithm

The DBSCAN algorithm (Ester, Kriegel, Sander, & Xu, 1996) starts by a random point p of the dataset that has not been visited previously. The number of data in the ε-neighborhood of p is identified. If the number of neighbors in the ε-neighborhood of p is less than MinPts, p is not the core point and it is labeled as noise, and the algorithm selects another random point of the dataset. Otherwise, if the number of neighbors of p in the ε-neighborhood is more than or equal to the MinPts, p is marked as the core point and a new cluster is formed and the points p takes the label of this cluster.

As can be seen in the pseudo-code of the DBSCAN algorithm, the point p and its neighbors in the ε-neighborhood are added to the list N and transmitted to the expandCluster function for more expansion. In the expandCluster function, each neighbor in the ε-neighborhood of p, for example p1, is examined and if they are not visited before and it is a core point, its neighbors are also added to list N for more expansion. Otherwise, if p1 is not the core point, it takes the cluster's label and it is not expanded anymore. This process continues until the list N is emptied. Then, the DBSCAN algorithm selects another random point of the dataset that has not been visited before, and continues identifying another cluster according to the above procedure. This process continues until all points of the database are visited and no cluster is found.

3. The proposed Algorithm: OP-DBSCAN

The main idea in designing the proposed method is to use local data to find neighbors. Due to the often soft and local movement of the DBSCAN algorithm in finding clusters, the implementation of this idea has been achieved by defining a smaller set of data called the operational dataset.

3.1. Using the concept of operational dataset

As mentioned, the DBSCAN algorithm is of time order O(n^2). The run-time of this algorithm is significantly affected by finding the neighbors of each data to obtain the data density. Therefore, the DBSCAN algorithm requires heavy computations for big data, which reduces the clustering speed, and increases the run-time. The proposed method tries to reduce the distance calculations as much as possible. To explain the proposed algorithm, it is first required to describe the operational dataset and potential dataset.

3.1.1. Operational dataset

To find the data samples of a cluster and extend the cluster properly, the DBSCAN algorithm operates locally and there is no hop in this algorithm. In other words, the next data selected by this algorithm to examine if it is core or not, is the neighboring data of the previous examined points.

Since DBSCAN wants to find the neighbors of each data sample, it must calculate the distance of this sample from all other ones. But it should be noted that since the goal is to find the neighbors, too far data samples can be neglected; In other words, calculating the distance from too far data sample can be pruned. To achieve this goal, a local space of closer data samples that will also have a spherical shape with a specific radius is considered as the operational dataset. In this dataset, instead of calculating the distance from all the data points, only the distance from the data of the same operational dataset is calculated to identify its neighbors. Therefore, calculation pruning has occurred, and the additional distance calculations from far data are eliminated. This concept helps to reduce computational overhead and run time of the DBSCAN algorithm significantly. Fig. 1 shows the operational dataset that includes the data that would be examined in an early future according to the DBSCAN algorithm scan.Fig. 2.

In Fig. 1, P is the starting point and the center of the operational dataset. *Size* is the radius and determines the size of the operational dataset. Therefore, the operational dataset, that is called OP in brief, includes all data in a spherical space with radius of size and center of P. The mathematical representation of the operational dataset is shown in Eq. (1):

$$OperationalDataset = \{x|x \in D.\, |x - P| \leq size\} n \in R. size = n^*\varepsilon.\, n > 1 \tag{1}$$

In which, D is the initial database, x is a sample in the database, P is the center of the operational dataset, |x-P| is the Euclidean distance between data P and ×, and finally, n*ε is the radius of the operational dataset.

The radius of the OP can be decreased and increased. its value can be adjusted by the user. The radius of OP is a multiple of epsilon (parameter of the DBSCAN algorithm radius) as n*ε.

If the OP size is large (large n), it includes a larger space of the initial dataset; therefore, more points are likely to exist in this space. Although larger operational dataset can lead to less update overhead, the distance from more data samples must be calculated for determining the label of each data sample.

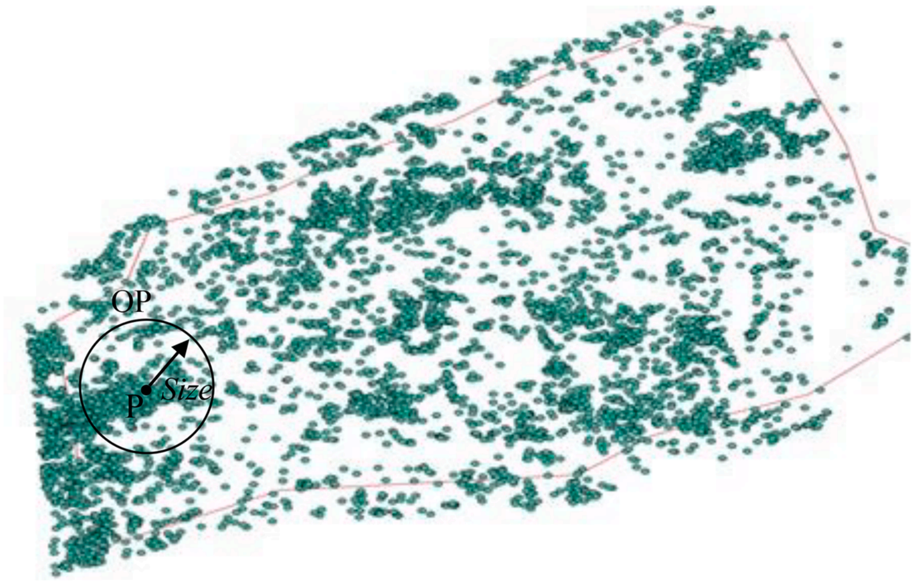

Fig. 1. The operational dataset.

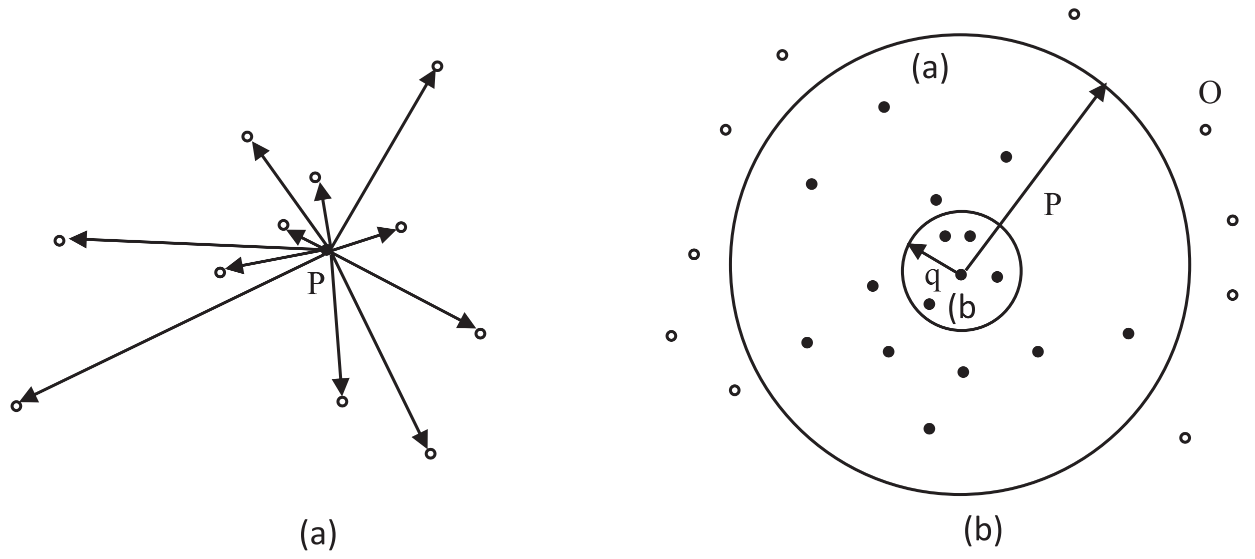

Fig. 2. (a) Calculating distance of P from all points of the dataset. (b) Constituting the OP with center of P and radius of n*ε.

On the contrary, the smaller the OP size is, the less the distance calculation cost would be. However, as the size of the OP dataset decreases, the overhead associated with updating it will increase. Therefore, there is a trade-off between the radius of the OP and the update overhead of the OP. The optimal radius of OP is calculated through calculating time for various radiuses of the OP via trial and error. Increasing or decreasing the OP radius affects the algorithm's run-time, but it does not affect the clustering quality. The important point is that although the size of the OP affects the time of the algorithm, using this set with any reasonable size greatly reduces the run time compared to the original DBSCAN algorithm.

3.1.2. *The potential dataset*

The potential dataset includes the data that is too far from the samples that are currently under investigation and thus ignored temporarily in order to reduce the calculation. In other words, the potential dataset includes all data of the initial dataset, except the data existing in the OP. The name potential dataset is adopted because the data of this set has the potential to constitute another OP in an early future. The mathematical representation of the potential dataset is also given in Eq. (2):

$$Potential\,Dataset = \{x|(x \in D)\&\&(x \notin OP)\}orPD = D - OP \qquad (2)$$

Where D is the initial dataset, OP is the operational dataset, PD is the abbreviation of the potential dataset, and × is a sample in the initial dataset.

3.2. *Phases of OP-DBSCAN algorithm*

Now, we describe the proposed algorithm. The proposed method is comprised of three main phases, which are described in the following.

3.2.1. ***First Phase:*** *Constituting the OP and PD*

In this step, the main dataset is divided into OP and PD. Similar to DBSCAN, a random data P is selected from the initial dataset, and distance of P from all data existing in the dataset is calculated. By calculating these distances and considering the definition of OP, the OP and PD can be obtained from the main dataset without any additional calculations compared to the conventional DBSCAN.

Considering the parameters ε and MinPts, if the number of data existing in the neighborhood of P is less than MinPts, the point P is labeled as noise, and the algorithm goes to another random point of the database. If the data P is identified as core, depending on the value of the parameter selected by the user as the radius of OP, all data that their distance from P is smaller than or equal to this input parameter is considered as the OP and the rest of data is considered as PD. Therefore, to detect whether the data in the OP is core or not (to identify the neighbors of each data), instead of calculating the distance from the whole initial dataset samples, it is calculated from just OP dataset samples. This causes a significant reduction in computation cost.

3.2.2. ***Second Phase:*** *Running the DBSCAN algorithm on the OP*

After generating the OP, the DBSCAN algorithm is applied to it. In this algorithm, the samples are scanned in depth first. When the neighbors of a sample are identified and it turns out that the sample is a core one, all its neighbors are entered to the stack. Then, a neighbor is removed from the stack, its neighbors are found and added to the stack under the same conditions. This process continues until the neighbors of all data inside the stack are restored and there is no data in the stack (the stack is empty). Thus, a list of the data that belongs to one cluster is identified. The ExpandCluster function in Fig. 7 shows the execution process of DBSCAN.

The important point while scanning the samples is that the distance of each sample form the center of the current OP should be smaller or equal to (n-1)*ε. In other words, the data of OP that their distance from the center of the OP is smaller than or equal to (n-1)*ε are scanned, and the rest of the data, which are between n*ε and (n-1)*ε, are considered as the candid dataset, which is used to update the OP.

As shown in Fig. 3, scanning the samples between n*ε and (n-1)*ε is neglected temporarily because considering the current OP, only a part of the neighbors of these samples (the candid set) is accessible and other neighbors are outside the OP. For example, in Fig. 3, the point M is between n*ε and (n-1)*ε. As can be seen, a part of the ε-neighbors of M are outside the OP. Since only the distance with samples of the OP is calculated to identify the neighbors, the neighbors of M that are outside the OP are not accessible (r and q).Fig. 4..

Theorem: In the OP, all the neighbors of the sample that is less than or equal to (n-1)*eps from the center of the OP, are accessible.

Proof: According to Fig. 3, the point A is on a circle of radius (n-1)*ε and center of P. The circle OP includes the center P and radius of n*eps, and the circle OP-1 includes the center P and radius of (n-1)*ε. The difference of the radius of the two circles (OP and OP-1) is epsilon. The ε-neighborhoods of the point A include all samples in a circle with center of A and radius of ε, where this circle is internal tangent with OP; therefore, all ε-neighbors of A are inside the OP circle, but if a point is farther than (n-1)*ε from the center P (the point is between (n-1)*ε and

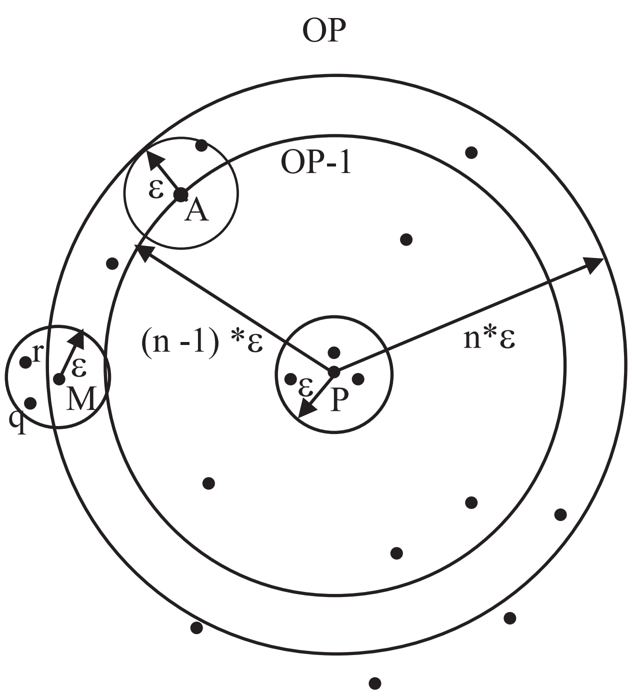

Fig. 3. Scanning the samples of the OP and constituting the candid set.

ε), some ε-neighbors of that point would be outside the OP circle. If a point is at a distance smaller than (n-1)*ε from the central data P (it is inside the OP-1 circle), in the worst case, a part of its ε-neighbors are inside the OP-1 circle and the other neighbors are inside the OP circle; since the OP-1 circle is the subset of the OP circle, all neighborhoods would be inside the OP circle.

3.2.3. ***Third Phase:*** *Updating the operational dataset*

Calculating the distance from other samples and finding the neighbors for each data point is limited to the operational dataset that sample belongs to. Therefore, to scan and restore the neighbors of the data that are not inside the current OP, the OP should be updated. Two main questions are: “when should the OP be updated?”, and “which sample should be used to update the OP (as the center of the new OP)?”.

3.2.3.1. Updating location of the OP. First, we should introduce the candid set.

Candid set: the samples with distance between n*ε and (n-1)*ε from the sample P, are called the candid set. They are called candidate because while updating the OP, if the candid set is not empty, the center of the next OP is selected from the samples of this set; if the candid set is empty, the center of the next OP would be a random point of the initial dataset. As mentioned before, the samples of the OP that their distance from the center of the OP exceeds (n-1)*ε, enter the candid set. The mathematical representation of the candid set is given in Eq. (3).

$$Candid\ Set = \{Vx \in D | d(x.P) \langle size \&\& d(x.P) \rangle size - \varepsilon \} \tag{3}$$

$$Candid\ Set = \left[OP_{n^*\varepsilon}\ U\ OP_{(n-1)^*\varepsilon}\right] - \left[OP_{n^*\varepsilon}\ \cap OP_{(n-1)^*\varepsilon}\right] \tag{4}$$

On the other hand, the candid set can be represented as in Eq. (4) using the union and intersection of the sets. The symbol OP $_{n*\varepsilon}$ represents the OP of radius n*ε with the center of P and OP $_{(n-1)*\varepsilon}$ represents the OP of radius (n-1)*ε with center of P.

If at the time of the update, the center of the new OP is selected as a point outside the candid set and the old OP, the cluster might be sliced and the algorithm will not have a correct number of the clusters. In fact, it identifies more clusters than the real number of the clusters. Therefore, to prevent cluster breakdown and perform clustering correctly and efficiently, the next OP center is selected among the samples of the candid set.

As seen in Fig. 5, the clusters A and B continue to the outside of the current OP (OP1). The point P1 belongs to the cluster A. In Fig. 5(a), while updating, the OP2 center is selected among the candid set points (P1) and all accessible points of the new center adopt the center cluster number (cluster A). In Fig. 5(b), while updating the OP, a random point of the initial dataset (P2) is selected as the center of the second OP. As can be seen, the density-connected samples of P2 have constituted the independent cluster of C, while all samples of clusters A and C belong to the same cluster. Similarly, for clusters B and D, all samples of these two clusters belong to one cluster.

3.2.3.2. Update time of the operational dataset. The operational dataset is updated when the neighbors of all samples of OP$_{(n-1)*\varepsilon}$ or all samples at a distance of (n-1)*ε or smaller from the OP center are restored.

After updating and obtaining the new OP, obviously this set overlaps with the previous OP (As shown in Fig. 6). Therefore, there are data that have been scanned before in the new OP. Eventually, the OP is updated

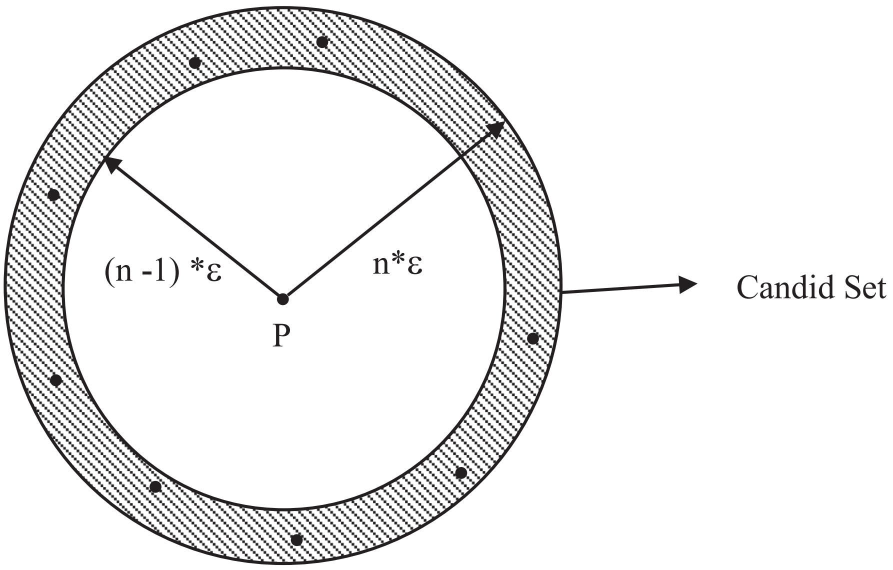

Fig. 4. Representation of the Candid set.

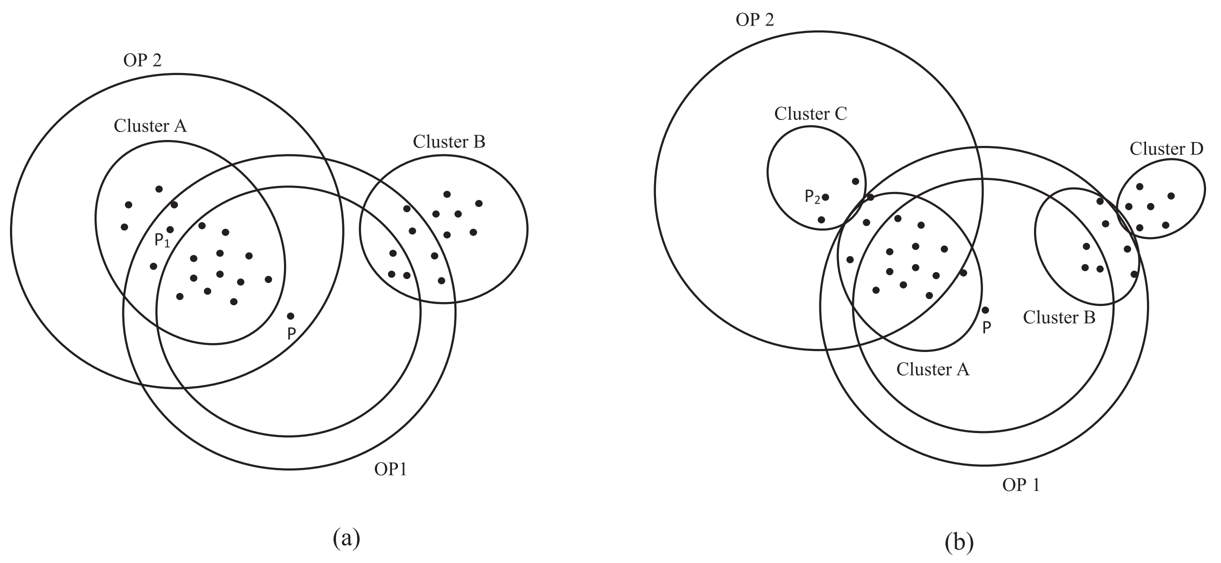

Fig. 5a. Using the samples of the candid set as the OP center while updating.

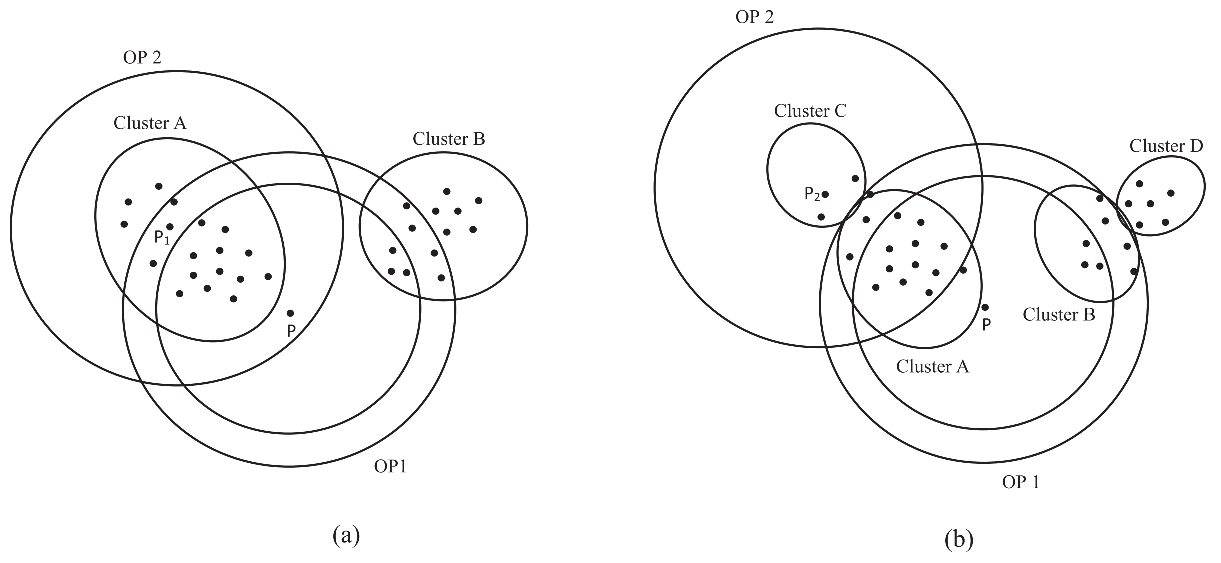

Fig. 5b. Using a random point of the initial dataset as the center of another OP while updating (instead of using the candid set).

provided that inequality (5) is satisfied.

$$V_{OP} > M - V_B \quad (5)$$

In Eq. (5), V_{OP} is a variable that counts the data scanned in the current dataset and M is the total number of samples at the distance of (n-1)*ε or smaller from the OP center, V_B is the number of samples that have been scanned in the previous OP. The sum of V_{OP} and V_B is always equal to M. The pseudo-code of the OP-DBSCAN algorithm is shown in Fig. 7.

3.3. *Time complexity analysis of the proposed algorithm*

The DBSCAN algorithm calculates the distance of each sample from all samples of the main dataset. Therefore, its time complexity is $O(n^2)$. But in the OP-DBSCAN algorithm, the distance of each sample of the OP from the other members of the same OP is calculated. On the other hand, in this algorithm, there exists the OP update overhead. Therefore, the time complexity of the proposed algorithm is as given in Eq. (6).

$$O(nk + nu).k + u << n \quad (6)$$

Since in each OP, the distance of each data from all other data of the same OP is calculated, the time complexity of this section is n*k, where k is the size of the OP. While updating the OP, and constituting new OPs, the distance of a sample from all other samples of the main dataset is calculated; since the number of update operations is considered as u, the time complexity of this phase is n*u. Therefore, the time complexity of the OP-DBSCAN algorithm is O(nk + nu). The larger is the OP (larger k), the update overhead would be smaller (smaller u) and vice versa, the smaller the top size is (smaller k), the larger the update overhead would be. According to Eq. (6), it is clear that the sum of k and u is very smaller than n.

Table 1 shows the effect of changes of radius of the OP on the run-time of the OP-DBSCAN algorithm in several datasets. As can be inferred from Table 2, the radius of the OP should not be very large or very small, because it increases the run-time of the OP-DBSCAN.

4. Experiments setup

In this section, some evaluation metrics are presented first. Then, the proposed method is compared with DBSCAN (Ester, Kriegel, Sander, &

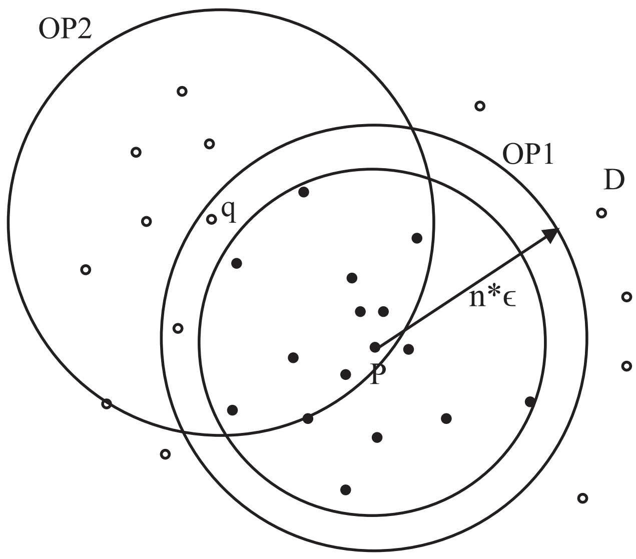

Fig. 6. Overlapping samples of two sets (OP1 and OP2).

Xu, 1996), HCA-DBSCAN (Mathur, Mehta, & Singh, 2019), fast Density-Grid (Brown, Japa, & Shi, 2019), and K-DBSCAN (Gholizadeh, Saadatfar, & Hanafi, 2020) in terms of qualitative evaluation metrics and run-time.

4.1. *The datasets used in the experiments*

Since the main idea of this paper is to reduce the run-time of the DBSCAN algorithm, to compare the proposed method with four other methods, several datasets of different dimensions and different number of samples are used. The datasets using in this study are adopted from UCI and GitHub such that the required diversity in terms of number of dimensions and number of samples is satisfied.

Table 2 shows the characteristics of 9 employed datasets, including the name of the dataset, number of samples, number of dimensions, epsilon and MinPts parameters for each dataset.

Considering the type of the dataset, the epsilon and MinPts of each dataset are specified. When comparing the proposed algorithm with the other four algorithms, in order to have the same test conditions, the value of epsilon and MinPts parameters is determined for each dataset based on its properties and is considered the same for all methods during the experiments (Table 2). Then, the proposed method is compared with four other methods considering these parameters under the same conditions.

4.2. *Evaluation conditions*

To compare the proposed algorithm with DBSCAN and the three mentioned methods, each method implemented and executed under the same conditions on a computational machine with a quad-core CPU, 40 MB disc, and 64 GB RAM. To obtain more accurate results, all unnecessary services of the operating system are disabled during execution of the algorithms. All datasets are normalized in the range of [0,1), and the OP-DBSCAN algorithm and four other algorithms are applied to the normalized datasets. The purpose was to compare the clustering quality (regarding clustering evaluation metrics) and run-time of the algorithms under the same conditions regarding hardware and common parameters. The results presented in all experiments are the average of ten different runs.

4.3. *Qualitative evaluation metrics*

Using proper evaluation metrics for examining the efficiency of a clustering method in finding exact clusters and comparing the performance of different methods is essential. To compare efficiency and quality of the proposed method with four other methods, two clustering quality validation metrics are used, which are described in the following:

4.3.1. ***Davies-Boulding index*** *(Davies & Bouldin, 1979)*

This index, which is called DB in brief, does not depend on the number of clusters or the clustering algorithm. This index calculates the average maximum intra-cluster scattering to the inter-cluster scattering ratio. The smaller is DB, the clustering performance is better. In other words, the DB index employs the similarity between two clusters (R_{ij}), which is defined based on the scattering of a cluster (S_i) and the dissimilarity between two clusters (D_{ij}).

The DB index is described as in Eq. (7):

$$DB = \frac{1}{k}\sum_{i=1}^{k} \max_{i \neq j} R_{ij} \tag{7}$$

Where k is the number of clusters, R_{ij} is the intra-cluster scattering to inter-cluster scattering ratio of clusters i and j, and R_{ij} is defined as follows:

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \tag{8}$$

In which S_i is the mean distance between each point of the i^{th} cluster and its center, and d_{ij} is the distance between the center of two clusters i and j.

4.3.2. ***Silhouette index*** *(Rousseeuw, 1986)*

Another method used to evaluate clustering, is the Silhouette index, which is represented by SI. This measure depends on the cohesion of the clusters and their separability. The SI value of each point represents its membership to its cluster compared to the adjacent cluster. To calculate SI, we need two main concepts:

The mean distance of one point of the cluster from other points of the cluster: this value is represented by a(i) and calculated as in Eq. (9):

$$a(i) = \frac{1}{n_i}\sum_{l=1}^{n_i} d(x_i.x_l) \tag{9}$$

In which, n_i is the number of members of the i^{th} cluster, and $d(x_i,x_l)$ is the distance of data x_i from other data of the same cluster. a(i) can be considered as a measure to evaluate the membership of x_i to its cluster. The smaller is a(i), the membership of the point to its cluster is higher.

Mean distance of a point from other clusters: for a point x_i, its mean distance from points of other clusters is calculated. The cluster with minimum mean distance for x_i, is called the adjacent cluster. The mean distance of x_i from the points of the adjacent cluster is represented by b(i).

$$b(i) = min_{1 \le l \le k} \frac{1}{n_l} \sum_{y_m \in c_l} (d(x_i.y_m)) \tag{10}$$

Where x_i is the data of the i^{th} cluster, $d(x_i,y_m)$ is the distance of x_i from y_m in l^{th} cluster and n_l is the number of members of the l^{th} cluster. Therefore, the SI for x_i is calculated using Eq. (11).

$$s(i) = \frac{b(i) - a(i)}{\max(b(i), a(i))} \tag{11}$$

Therefore, if a(i) is smaller than b(i), SI is positive and if b(i) is smaller than a(i), SI is negative, indicating weak clustering; because x_i is similar to the adjacent cluster instead of being similar to its cluster. Considering the above equation, SI varies between -1 and $+1$. Values close to 1 describe good match between the point and its cluster compared to the adjacent cluster. If SI is close to 1 for all points of the clusters, clustering is performed correctly. While small SI values indicate

```
Function ExpandCluster (Epsilon, Minpts, P, Neighbor, C)

While (Neighbor is not empty)
    V=Top of Neighbor
    If (V is not visited) && (distance between V & center of OP equal or less than
    (n-1) *epsilon
        Visited (V) and $V_{OP} = V_{OP} + 1$;
        NB= calculate distance between point V and samples of OP
          and find ε-neighborhood of V;
        If (number of NB) > Minpts
            N=N+ all member of NB;
        end
    Else if (distance between V & center of OP > (n-1) *epsilon)
        Candid =Candid+V;
    end
    Add V to cluster of C number;
end
While ($V_{OP} < M-V_B$)
    Find and expand cluster;
end
```

```
Function UpdateOP (Epsilon, Minpts, candid)
If (candid is empty)
    Break;
End

While (candid is not empty)
    V= top of candid;
    C=cluster number of point V;
    If (V is not visited)
        Visited(V)=true;
        $V_{OP} = V_{OP} + 1$;
        N=calculate distance point V to all samples of DB and
          find ε-neighborhood of V;
        If (V is a core Point)
            Build OP with center of point V;
            ExpandCluster (Epsilon, Minpts, V, N, C);
        end
    end
end
```

Fig. 7. Pseudo-code of the OP-DBSCAN Algorithm.

Table 1
Effect of changes of radius of the OP on the run-time of the OP-DBSCAN.

Dataset	OP = 2*ε	OP = 4*ε	OP = 6*ε	OP = 8*ε	OP = 10*ε	OP = 20*ε
Abalone	0.6072	0.4850	0.4328	0.4254	0.5085	0.58017
MAGIC	20.5358	8.4419	7.4395	7.0835	7.8031	28.2929
FARS	103.26	48.0892	180.5895	265.3311	368.0851	789.6889
3D Road Network	882.57992	289.1028	134.7067	88.0222	87.1958	195.3034

Table 2
Characteristics of the dataset used in this study.

Dataset	Number of samples	Number of dimension	Epsilon	MinPts
Abalone	4177	8	0.035	4
MAGIC	19,020	10	0.06	4
Educational Process mining	230,318	13	0.05	4
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
News Aggregator	422,937	5	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

weak clustering, this might be due to improper selection of the number of clusters.

In other words, if the mean SI is calculated for the samples of each cluster, an index is obtained to evaluate each cluster. The average SI is also a metric to evaluate the clustering performance.

5. Results analysis

The results of comparing the evaluation metrics and run-time for the 5 algorithms are given in Table 3 and Table 4.

According to Table 3 and Table 4, it should be noted that the original DBSCAN algorithm runs out of memory in large datasets (datasets with more than 200,000 samples), while other methods do not face this problem. The proposed method, in general, has a shorter execution time compared to other competitive methods, especially in the case of large data sets. For example, in the case of the FARS dataset (the largest dataset on which the original DBSCAN execution was successful), the execution time of the proposed method is more than 140 times that of the original DBSCAN and more than 6 times that of the K-DBSCAN method, Decreased. Regarding the two smaller data sets (Abalone and MAGIC), due to the small number of samples, the overhead caused by the update has overcome and the execution time of the proposed method is, in general, longer than other methods.

Based on the times specified in Table 4, the percentage of performance improvement (PPI) for the OP-DBSCAN algorithm with respect to the DBSCAN can be calculated through Eq. (12).

$$PPI = \frac{|t_{DBSCAN-}t_p|}{t_{DBSCAN}} \tag{12}$$

In this equation, t_{DBSCAN} represents the DBSCAN execution time, t_p represents the execution time of the proposed algorithm. Table 5 shows the percentage of performance improvement for the proposed algorithm.

By comparing the qualitative evaluation metrics of the proposed algorithm and 4 other methods, it can be concluded that the proposed algorithm is almost the same as the DBSCAN algorithm regarding the clustering quality, while it performs clustering faster than DBSCAN and three other algorithms.

Fig. 8 compares the run-time of the 5 algorithms. Fig. 9 compares SI and Fig. 10 compares DB for the 5 algorithms. These diagrams are drawn to better show the trend of changes in the performance of the proposed method compared to other methods on different datasets and by changing the data size.

6. Conclusion and future works

In this paper, a clustering algorithm for big data, called OP-DBSCAN is presented, which is an improved version of the well-known DBSCAN algorithm. The main idea employed in this paper is using the operational dataset and potential dataset concept to prune the data space and reduce the calculations used to find the neighbors, ensuring that the search space of a sample for finding the ε-neighborhood is always restricted to a smaller space compared to the total data space.. Due to the often soft and local movement of the DBSCAN algorithm in finding clusters, the implementation of this idea has been achieved by defining a smaller set of data called the operational dataset.

The OP-DBSCAN consists of three phases. The first phase is the Constituting the Operational dataset and Potential dataset. The second one is the implementation of DBSCAN algorithm and next phase is the updating of the operational data set. Therefore, the OP-DBSCAN algorithm is classified as calculation reduction methods. The experiments on different datasets show that the proposed algorithm has a higher speed compared to DBSCAN and three other algorithms, including HCA-DBSCAN, fast Density-Grid, and K-DBSCAN, while preserving clustering quality.

Among advantages of the proposed algorithm, increasing the clustering speed while preserving its quality can be mentioned. Also, instead of parameters of the DBSCAN, the OP-DBSCAN requires only one input parameter (operational dataset size) from the user.

Since the multiple machine-based clustering techniques are more scalable and faster than single machine-based techniques, using the map-reduce framework in the proposed algorithm can be studied in future studies. Also, the samples' density in bigger radiuses can be used to guess the location of samples in a cluster. Therefore, the process of labeling samples (for example, as cores or noises) can be accelerated and a considerable amount of calculation omitted. This idea can be used to improve the proposed algorithm in future studies. Considering a dynamic size for the operational dataset based on local density is another idea which can be followed in the future works.

Table 3
Comparing the evaluation metrics for 5 algorithms on 9 different datasets.

Dataset Evaluation Criteria		Abalone	MAGIC	Educational	Shuttle	MoCap	FARS	Skin Segmentation	New aggregator	Road Network	Cover Type	Poker hand
DB	OP-DBSCAN	1.4170	1.0916	1.022	0.7693	0.7546	1.4351	0.8411	1.035	0.8085	0.9015	1.012
	DBSCAN	2.0524	1.4197	1.054	1.8097	1.0755	1.7289	–	–	–	–	–
	HCA-DBSCAN	2.0560	1.5963	1.365	1.8602	1.0779	1.8516	1.4283	1.319	1.1746	1.132	1.2506
	K-DBSCAN	2.0626	1.4795	1.289	1.8535	1.0783	1.8238	1.3912	1.218	1.2299	1.142	1.2381
	Density-grid	2.0615	1.6359	1.519	1.8823	1.0786	1.8235	1.4681	1.345	1.2194	1.1415	1.2578
SI	OP-DBSCAN	−0.276	−0.516	−0.418	−0.405	−0.468	−0.436	−0.128	−0.212	−0.325	−0.596	−0.561
	DBSCAN	−0.638	−0.776	−0.430	−0.431	−0.645	−0.519	–	–	–	–	–
	HCA-DBSCAN	−0.651	−0.778	−0.437	−0.439	−0.646	−0.731	−0.148	−0.456	−0.541	−0.712	−0.832
	K-DBSCAN	−0.655	−0.778	−0.432	−0.443	−0.646	−0.688	−0.146	−0.325	−0.535	−0.714	−0.833
	Density-grid	−0.642	−0.779	−0.465	−0.445	−0.647	−0.751	−0.148	−0.418	−0.544	−0.712	−0.834

Table 4
Comparing the run-time of the proposed method and 4 other methods on 11 different datasets.

Running Time (s)	Abalone	MAGIC	Educational	Shuttle	MoCap	FARS	Skin Segmentation	New aggregator	Road Network	Cover Type	Poker hand
OP-DBSCAN	0.4197	7.0835	9.458	20.2553	56.8786	48.0892	123.3931	75.45	77.5716	122.1612	615.4721
DBSCAN	0.3195	4.4409	9.895	128.28	3755.8	16950.74	–	–	–	–	–
HCA-DBSCAN	0.3726	4.0123	9.756	104.95	586.70	1193.5	2058.1	3745.8	3967.5	4108.3	12413.42
Density-Grid	0.3905	5.5327	8.569	114.68	672.98	1627.6	3575.4	4536.7	4756.7	4825.7	13526.63
K-DBSCAN	0.4537	6.2015	8.781	82.893	177.96	299.15	462.47	789.2	806.19	733.51	1789.7

Table 5
Percentage of improvement by the proposed algorithm in the execution time of DBSCAN.

Dataset	Educational	Shuttle	MoCap	FARS
PPI	4.41%	84.21%	98.48%	99.71%

CRediT authorship contribution statement

Nooshin Hanafi: Methodology, Software, Investigation, Validation, Writing – original draft. **Hamid Saadatfar:** Conceptualization, Formal analysis, Writing – review & editing, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial

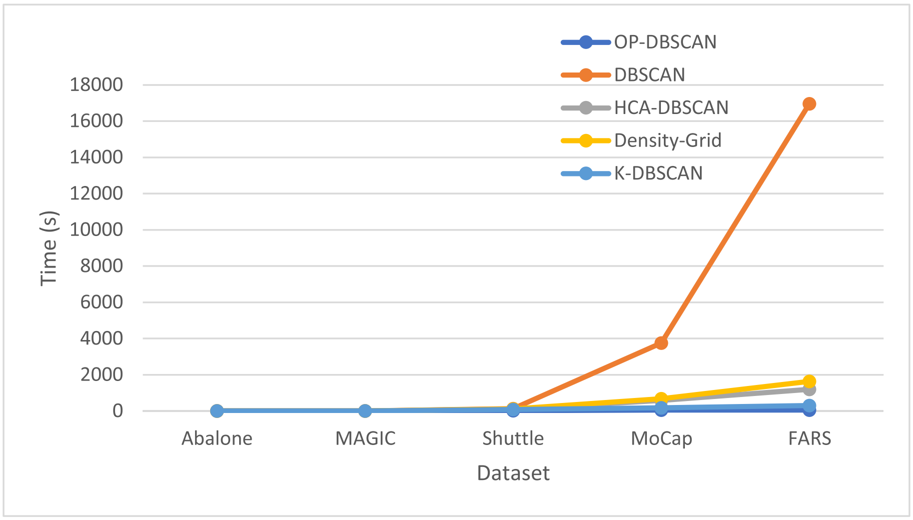

Fig. 8. Comparing run-time of 5 algorithms (DBSCN, HCA-DBSCAN, Density-Grid DBSCAN, K-DBSCAN, and OP-DBSCAN).

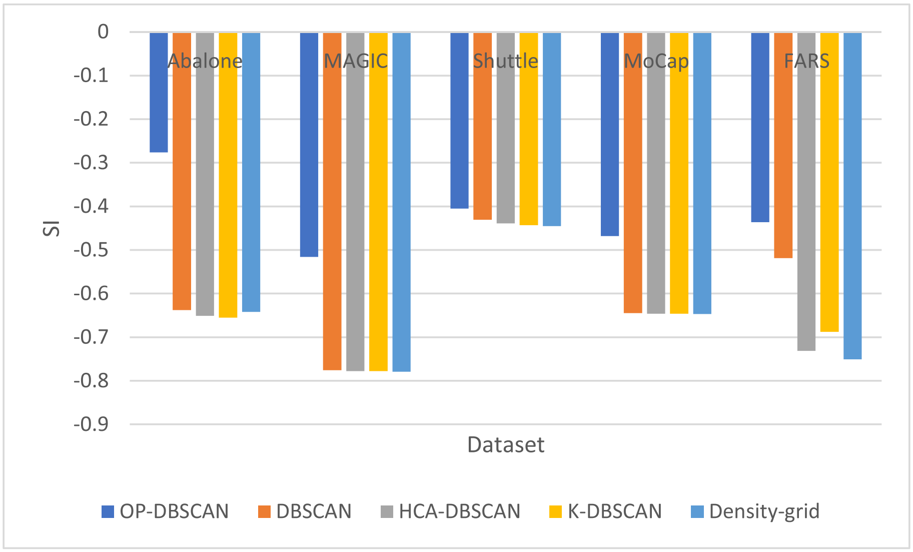

Fig. 9. Changes of SI for the 5 algorithms on different datasets.

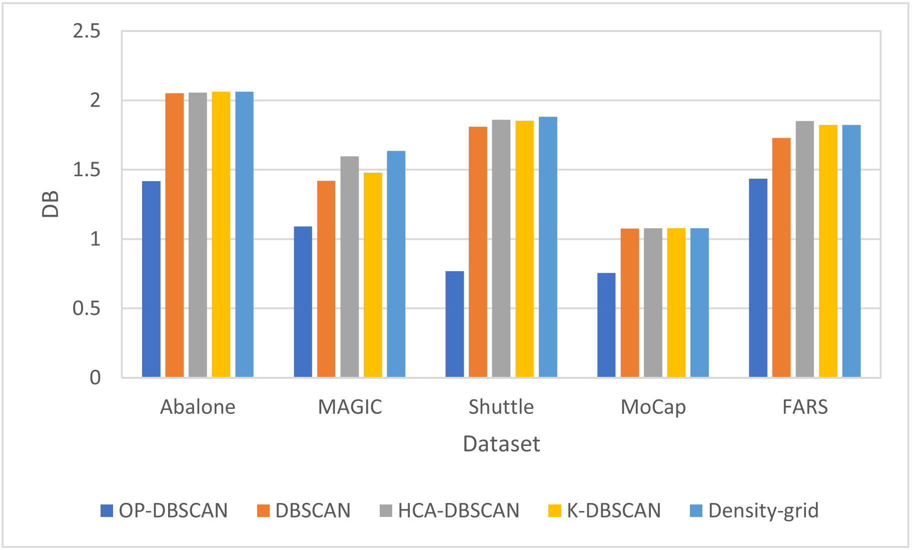

Fig. 10. Changes of DB for the 5 algorithms on different datasets.

interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Ananthi, V., Balasubramaniam, P., & Kalaiselvi, T. (2015). A new fuzzy clustering algorithm for the segmentation of brain tumor. *Soft Computing*, 4859–4879.

Baranidharan, B., & Santhi, B. (2016). DUCF: Distributed load balancing Unequal Clustering in wireless. *Applied Soft Computing*, 495–506.

Berkhin, P. (2006). A Survey of Clustering Data Mining Techniques. In I. J. Kogan, C. Nicholas, & M. Teboulle (Eds.), *Grouping Multidimensional Data* (pp. 25–71). Springer.

Brown, D., Japa, A., & Shi, Y. (2019). *A Fast Density-Grid Based Clustering Method*. Las Vegas, NV, USA: IEEE.

Chen, Y., Tang, S., Bouguila, N., Wang, C., Du, J., & Li, H. (2018). A Fast Clustering Algorithm based on pruning unnecessary distance computations in DBSCAN for High-Dimensional Data. *Pattern Recognition, 83*, 375–387.

Chen, Y., Zhou, L., Bouguila, N., Wang, C., Chen, Y., & Du, J. (2021). BLOCK-DBSCAN: Fast clustering for large scale data. *Pattern Recognition, 109*, Article 107624.

Davies, D., & Bouldin, D. (1979). A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1*, 224–227.

de Moura Ventorim, I., Luchi, D., Loureiros Rodrigues, A., & Miguel Varejão, F. (2021). BIRCHSCAN: A sampling method for applying DBSCAN to large datasets. *Expert Systems with Applications, 184*, Article 115518.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *KDD-96 Proceedings*.

Feyyad, U. (1996). Data mining and knowledge discovery: Making sense out of data. *IEEE Expert, 11*, 20–25.

Gholizadeh, N., Saadatfar, H., & Hanafi, N. (2020). K-DBSCAN: An improved DBSCAN algorithm for big data. *The Journal of, Supercomputing(77)*, 6214–6235.

Gunawan, A., & Berg, M. (2013). A faster algorithm for DBSCAN. In *Master's thesis*.

Hahsler, M., & Bolaos, M. (2016). Clustering Data Streams Based on Shared Density Between Micro-Clusters. *IEEE Transactions on Knowledge and Data Engineering, 28*, 1449–1461.

He, Y., Tan, H., Luo, W., Mao, H., Ma, D., Feng, S., & Fan, J. (2011). MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce. *2011 IEEE 17th International Conference on Parallel and Distributed Systems.* Tainan, Taiwan: IEEE.

Hou, J., Liu, W., E, X., & Cui, H. (2016). Towards parameter-independent data clustering and image segmentation. *Pattern Recognition, 60*, 25-36.

Kesavaraj, G., & Sukumaran, S. (2013). A study on classification techniques in data mining. *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. Tiruchengode, India: IEEE.

Kumar, K., & Reddy, A. M. (2016). A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method. *Pattern Recognition*, 39–48.

Li, S.-S. (2020). An Improved DBSCAN Algorithm Based on the Neighbor Similarity and Fast Nearest Neighbor Query. *IEEE Access*, 47468–47476.

Mahesh, B. (2020). Machine Learning Algorithms - A Review. *International Journal of Science and Research (IJSR), 9*(1).

Mai, S., Assent, I., & Storgaard, M. (2016). In *AnyDBC: An Efficient Anytime Density-based Clustering Algorithm for Very Large Complex Datasets* (pp. 1025–1034). ACM Press.

Mathur, V., Mehta, J., & Singh, S. (2019). HCA-DBSCAN: HyperCube Accelerated Density Based Spatial Clustering for Applications with Noise. *Sets and Partitions workshop at NeurIPS 2019*.

Qiao, S., Li, T., Li, H., Peng, J., & Chen, H. (2012). A new blockmodeling based hierarchical clustering algorithm for web social networks. *Engineering Applications of Artificial Intelligence, 25*, 640–647.

Rousseeuw, P. (1986). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 53–65.

Rubio, E., Castillo, O., Valdez, F., Melin, P., Gonzalez, I. C., & Martinez, G. (2017). An Extension of the Fuzzy Possibilistic Clustering Algorithm Using Type-2 Fuzzy Logic Technique. *Advances in Fuzzy Systems, 2017*, 1–23.

Sanchez, M., Castillo, O., Castro, J., & Melin, P. (2014). Fuzzy granular gravitational clustering algorithm for multivariate data. *Information Sciences, 279*, 498–511.

Shirkhorshidi, A., Aghabozorgi, S., Wah, T., & Herawan, T. (2014). In *Big Data Clustering: A Review* (pp. 707–720). Springer International Publishing.

Sinha, A., & Jana, P. (2016). A Novel K-Means based Clustering Algorithm for Big Data. *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. Jaipur, India: IEEE.

Weng, S., Gou, J., & Fan, Z. (2021). h -DBSCAN: A simple fast DBSCAN algorithm for big data. In *Proceedings of The 13th Asian Conference on Machine Learning* (pp. 81–96).

Zerhari, B., Lahcen, A., & Mouline, S. (2015). Big Data Clustering: Algorithms and Challenges. *International Conference on Big Data, Cloud and Applications.* Tetuan, Morocco.