2024 3rd International conference on Power Electronics and IoT Applications in Renewable Energy and its Control (PARC) | 979-8-3503-0940-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/PARC59193.2024.10486339

A Survey of Advancements in DBSCAN Clustering Algorithms for Big Data

Omkaresh Kulkarni
Deparment of Computer Science and Engineering (AIML)
VIIT
Kondhwa-BK, Pune, India
omkaresh.kulkarni@viit.ac.in

Adnan Burhanpurwala
Department of Artificial Intelligence and Data Science
VIIT
Kondhwa-BK, Pune, India
adnan.22010210@viit.ac.in

Abstract—Cluster analysis is an unsupervised machine learning job of grouping objects based on some similarity measure. Among clustering algorithms, DBSCAN (Density Based Spatial Clustering of Application with Noise) contributes to unsupervised machine learning by enabling the clustering of datasets with varying densities, shapes, and sizes. DBSCAN does not require the predefinition of the number of clusters and is able to recognize noiseless arbitrary clusters by using two parameters, minPts and eps. This paper reviews the different DBSCAN algorithms for big data clustering and provides a detailed comparison among the algorithms.

Keywords—clustering algorithm, Density based clustering, DBSCAN, big data, data mining.

I. Introduction

As the use of computers and technology continues to rise, the quantity of data generated by humans also increases. Large companies like Walmart and Facebook store nearly 2.5 and 30 petabytes of data, respectively. Such massive quantities of data have prompted scientists to coin a term to describe it, namely “Big Data”. Big data is distinguished from ordinary data by four main characteristics. The Four V’s of big data are velocity, veracity, variety, and volume [13].

Various sources, such as GPS trajectories from taxis in metropolitan areas, generate substantial data, which data centers analyze rapidly. This analysis reveals insights into passenger movement patterns, taxi habits, and enables suggestions for optimal locations to locate passengers, illustrating the need for swift big data classification to transform raw data into meaningful information [13].

Data Mining is the process of extracting patterns and knowledge from large datasets [2]. Clustering algorithms have proven to be very beneficial in data mining and machine learning. Presently, there are many types of clustering algorithms that have been proposed and implemented like, partitional clustering, density-based clustering, grid-based clustering, and others [17].

Among density-based clustering, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is the most used algorithm [6]. It is a valuable tool for discovering clusters and outliers within datasets, aiding in the extraction of meaningful insights from complex data structures. DBSCAN contributes to data mining by efficiently grouping spatial data points based on density.

There have been numerous papers in recent times that have proposed various modifications and improvements to DBSCAN to make it better suited for big data clustering. However, there is a lack of a comprehensive survey that compares all the DBSCAN algorithms and provides an overview of the advantages and disadvantages of the algorithms. The objective of this paper is to summarize the variations in DBSCAN that have been proposed keeping in mind the task of clustering big data. The contribution of this paper is to identify the existing work done and the future enhancements that can be made to the algorithms to implement them on the latest frameworks.

In the next section, the DBSCAN algorithm is explained. Section III provides a literature survey with tabular analysis. In Section IV, the algorithms have been evaluated over some performance metrics. Lastly, Sections V and VI contain future enhancements and conclusions, respectively.

II. Density Based Spatial Clustering Of Applications With Noise For Big Data

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [6] stands as a widely utilized density-based spatial clustering method employed across various applications. It is capable of efficiently recognizing clusters of arbitrary shapes and separating noise. The input of the number of clusters is not needed [6]. In DBSCAN, clustering is the process of identifying regions with higher density from regions with low density in a given feature space [6]. DBSCAN runs in O(n2) for the worst case and O(nlogn) for the average case.

The general workings of density-based clustering algorithms can be broken down into two major steps. First is figuring out a procedure for estimating the density of a point and identifying objects that lie in dense regions. The second is a procedure to identify groups of observations that are reachable from some core observations [13]. In DBSCAN, clusters are constructed according to two main parameters: minPts and eps [13], which are defined as follows:

- minPts: Smallest count of points required within an object's eps vicinity to categorize it as a core point.
- eps: Radius of region surrounding an object.

The density of a point is determined by the count of neighboring points situated within a specified radius of the cluster. DBSCAN seeks out objects (those not considered core points) that can be reached in terms of density from these core objects. This process might encompass combining certain clusters whose density is reachable. Finally, when it becomes impossible to include additional points in any of the clusters, the leftover points are designated as outliers or noise.

However, DBSCAN is unsuitable for clustering large datasets due to its huge time requirements [7]. The computation time for high-dimensional data is also extremely high [7]. DBSCAN lacks the capability to detect clusters characterized by differing densities and partially intersecting boundaries. Most importantly, the performance and quality of the clusters generated by DBSCAN largely depend on its parameters. Without proper estimation of the parameters, the most accurate clusters may not be obtained.

III. Literature Survey

There are several variants of DBSCAN that are proposed to make it better suited for big data clustering. They are divided into two categories based on their design. Some more general algorithms are reviewed in the third sub-section. Table. 1 shows the comparison of the DBSCAN algorithms along with their advantages and limitations.

A. Fast-Approximate based DBSCAN Algorithms:

Existing exact DBSCAN can grow computationally expensive when the data size (n) increases to billions of points [3]. In this context, approximate versions of algorithms are of high interest since they tend to perform equally well as compared to exact algorithms but much faster. There is a trade-off between speed and accuracy that needs to be dealt with by most of the approximate algorithms.

KNN-BLOCK DBSCAN [17] employs a fast approximate kNN approach known as FLANN. The priority search k-means tree is integrated with FLANN to perform the nearest neighbor search. Subsequently, with the assistance of FLANN, it detects clusters of similar point types. In conclusion, the core-blocks are merged, and non-core points are allocated to their corresponding clusters.

BLOCK-DBSCAN [4] is another algorithm that combines the fast approximate technique with the use of cover tree data structures. In this algorithm, a cover tree, which is one of the fastest nearest-neighbor algorithms, is used to identify inner-core blocks, outer-core points, and the border points. A fast and approximate algorithm is employed to forecast whether there is density-based reachability between two inner core blocks. As part of this process, clusters will be formed by combining inner-core blocks and outer-core points, while border points will be allocated to their respective clusters.

PARDICLE [12] is a fast approximate and parallel algorithm that leverages dynamic programming for achieving load distribution and spatial proximity. The approach relies on approximate neighbor calculations, wherein, within a specified search radius (eps), it is unnecessary to retrieve all points within that range to identify accurate clusters. PARDICLE-MULTICORE [12] is the parallel version of PARDICLE. And PARDICLE-MULTINODE [12] is the distributed approximate version of PARDICLE-MULTICORE.

B. Soft-Clustering Based DBSCAN Algorithms:

A few soft clustering approaches are also proposed, like Fuzzy-DBSCAN [11] and E-DBSCAN [2]. These utilize soft constraints instead of exact values to represent approximate values of the input parameters. While crisp DBSCAN algorithms detect clusters with sharp boundaries, soft clustering algorithms generate clusters with fuzzy, overlapping boundaries. The advantage of fuzzy clusters is that in a single run of clustering, several distinct runs of the original approach can be summarized.

Fuzzy-DBSCAN [11] results from the fusion of Fuzzy Core DBSCAN and Fuzzy Border DBSCAN. The Fuzzy DBSCAN or FDBScan models fuzziness on core points and border points using the two soft constraints to replace minPts and eps. This enables the definition of a fuzzy local density and the fuzzy size of the local neighborhood for points.

E-DBSCAN [2] or Evidential DBSCAN is another soft clustering technique that integrates the principles of DBSCAN with the concepts of belief function theory to produce clusters with overlapping boundaries. The algorithm is equipped to manage the uncertainty of object cluster membership degrees. The E-DBSCAN method is characterized by defining an estimated radius value instead of a precise numeric parameter like eps.

C. Some more variants of DBSCAN for big data:

Apache Spark, known for distributed computing, can enhance the efficiency of the DBSCAN clustering algorithm when dealing with large spatial datasets. Despite this very few DBSCAN algorithms are implemented in the Spark framework. Partly, this is due to Spark being relatively new and requiring programmers to invest substantial time in creating improved algorithms to enhance the effectiveness of parallelization.

SEED-DBSCAN [8] is one of the few algorithms that have been implemented in the Apache Spark framework. However, the implementation leads to some practical issues. SEED-DBSCAN uses Spark features like executors to compute the partial clusters. And later these are merged into the driver. However, the task of identifying which partial clusters need to be merged in a single cluster is a challenge. This challenge is tackled by avoiding communication among different drivers to prevent overlap of partial cluster computation. And the uses of SEEDs that act as markers to help the drivers identify the partial clusters that need to be merged.

TABLE I. Comparison of Density-Based Clustering Algorithms for Big Data

No.	Algorithm	Author Name	Description	Advantages	Limitations
1.	RNN DBSCAN [3]	A. Bryant and K. Cios	-Uses reverse nearest neighbor (RNN) to determine density. -Only necessitates a singular parameter *k* (where *k* is the number of nearest neighbors).	-The parameter k remains unaffected by the dataset size when dealing with significant values of *n*. -Performs better than prior RNN approaches	-Performance drops when run on real-world datasets. -Parameter selection of k is based on trial and error.
2.	KNN-BLOCK DBSCAN [4]	Y. Chen, L. Zhou, S. Pei, Z. Yu, Y. Chen, X. Liu, J. Du and N. Xiong	-Uses a fast approximate kNN for identifying the type of each block (CB, NCB, NOB). -Data is processed by blocks instead of grids.	-Works well on large scale datasets. -Diminishes the extensive count of repetitive distance calculations. -Executes faster than pure kNN-based algorithm.	-Not suitable for high dimensional data. -With a high *minPts* value and a small *eps* value, the runtime increases.

3.	µDBSCAN-D [14]	A. Sarma, P. Goyal, S. Kumari, A. Wani, J. Sesh Challa, S. Islam and N. Goyal	-Parallel version of micro cluster-based µDBSCAN. -Comprises of three stages: 1) Partitioning of Spatial Data. 2) Local µDBSCAN Execution. 3) Integration of Local Clusters.	-Suitable for high dimensional data. -Identifies core points without having to perform neighborhood-queries.	-Requires higher memory since it uses *µR-tree*.
4.	K-DBSCAN [7]	Gholizadeh, N., Saadatfar, H. and Hanafi, N.	-Grouping is first applied using K-means++. -Then clustering is performed in each group separately using DBSCAN.	-Improves the execution speed of DBSCAN. -Requires only modest hardware resources and can be run on a solitary machine.	-The algorithm ignores noise points.
5.	MR-DBSCAN [9]	Y. He, H. Tan, W. Luo, S. Feng and J. Fan	-Uses cost-based spatial partitioning (CBP) which partitions data based on the estimated computation costs. -Uses MapReduce to ensure scalability of algorithm.	-Offers an approach for splitting an extensive dataset into multiple partitions according to the dimensions of the data. -Works well with skewed spatial big data.	-Increase in *eps* value, increases CBP partitioning time.
6.	BLOCK-DBSCAN [5]	Y. Chen, L. Zho, N. Bouguila, C. Wang, Y. Cuhen and J. Du	-Uses cover tree to compute density and identify type of each block. -Fast approximate algorithm is used to find density reachability between two inner core blocks.	-Suitable for high dimensional data. -Highly efficient and precise.	-Worst case time complexity is $O(n^2)$
7.	SEED-DBSCAN [8]	D. Han, A. Agrawal, W. -K. Liao and A. Choudhary	-Constructs *kd-tree* to reduce search time. - SEED's (points not assigned to the current partition) are employed to combine partial clusters into global clusters.	-One of the few algorithms to be implemented in Spark framework -Better scalable performance. -Outperforms similar implementation on MapReduce framework.	-Data points are not partitioned based on the neighborhood relationship which may cause workload to be unbalanced.
8.	E-DBSCAN [2]	M. Bessrour, Z. Elouedi and E. Lefevre	-An approximate value of radius (*eps*) is specified instead of crips numeric parameter *eps*. -Fuzzy clustering technique grounded in the principles of belief function theory.	-Faster than other soft methods like ECM and EK-NN cluster.	-More time consuming if dataset contains high overlapping areas between clusters.
9.	DBSCAN-GM [15]	A. Smiti and Z. Elouedi	-First runs Gaussian Means and estimates parameters for DBSCAN. -Next, it executes DBSCAN to manage noise and unveil clusters of diverse sizes and shapes.	-Suitable for high-dimensional data and capable of handling noise. -Does not require any prior information and automatically generates the parameters.	-The time needed to construct the clusters is approximately three times the runtime of DBSCAN.

While K-means++ is a popular clustering method initself, in K-DBSCAN [7], it is not used to cluster the data but rather to divide the data. Initially, the algorithm employs the K-means++ technique to group the data, followed by the independent application of DBSCAN to each group. Lastly, the clusters produced within distinct groups are combined.

h-DBSCAN [16] is an algorithm, that proposes to decrease the execution time in two ways. The first is the reduction in the number of points presented to DBSCAN, and the second is the application of the HNSW (Hierarchical Navigable Small World) technique to provide an approximate solution to the problem of K-nearest neighbor search. µDBSCAN [14] is a micro cluster-based algorithm that uses a two-level R-tree, µR-tree for nearest neighborhood query. A parallelized and scalable version of µDBSCAN, µDBSCAN-D is proposed, which exploits distributed memory architecture and can effectively cluster large data sets.

MR-DBSCAN [9] completely parallelizes all critical sub-procedures. Also, it introduces the utilization of a data partitioning technique grounded in computational cost estimation, incorporating point density as a factor. When parallel implementation of MR-DBSCAN [8] takes place via MapReduce, this partitioning method guarantees that the size of each partition remains sufficiently small to fit into memory, thus allowing the application of any sequential DBSCAN algorithm for clustering purposes.

DBSCAN requires the user to determine the parameters that can influence the clustering results negatively; DBSCAN-GM [15] automatically generates the appropriate parameters. GriT-DBSCAN [10] is a DBSCAN variant based on a grid structure, using a grid tree to arrange non-empty grids. The algorithm utilizes the spatial relationships between points to determine if the merging of two core grids is feasible. WOA-DBSCAN [18] uses whale optimization algorithm to determine a range of values for the DBSCAN parameters based on the dataset distribution. Silhouette coefficient is used as an objective function. By using density peaks, WOA-DBSCAN addresses the sensitivity issue associated with input parameters.

STRP-DBSCAN [1] employs spatial-temporal random partitioning to evenly distribute the workload across various computing nodes. Data with comparable spatial and temporal attributes are allocated to the same partition, minimizing communication in parallel processing and enhancing overall efficiency. Through data preprocessing, the algorithm mitigates noise points in the initial dataset, thereby boosting the efficiency of parallel DBSCAN.

IV. Results And Discussions

The algorithms are compared based on a common evaluation metric (ARI). As well as their execution times, are compared. Both comparisons are done on the Iris dataset. The runtime comparison is depicted in Figure 1. And the ARI comparison of these algorithms is shown in Figure 2.

When the runtimes of DBSCAN, h-DBSCAN [16], RNN-DBSCAN [3], K-DBSCAN [7], and E-DBSCAN [2] are compared on the iris dataset, h-DBSCAN is found to be the fastest. K-DBSCAN is the second-fastest algorithm with a low execution time. And all of them were found to be faster than DBSCAN.

The ARI of DBSCAN, h-DBSCAN [16], RNN-DBSCAN [3], K-DBSCAN [7], and E-DBSCAN [2] is compared. E-DBSCAN is found to have the best ARI since it is able to identify clusters correctly in all cases.

It uses a fuzzy value of eps to generate clusters with fuzzy borders. RNN-DBSCAN [3] requires only one parameter, k. k is used to determine the number of nearest neighbors and to identify dense observations. This makes it excellent for applying it to large datasets. Table. 2 shows the various performance metrics, time complexity, and input parameters used by the algorithms.

RNN-DBSCAN performs moderately well when tested on artificial datasets. In a distributed framework like MapReduce or Spark, the task of finding the data point which has a given point as its nearest neighbor can be computed across multiple nodes to distribute the workload. This will help in making RNN-DBSCAN more scalable with big data. If RNN-DBSCAN is implemented on parallel framework, Apache Spark is the best choice since it offers in-built fault tolerance and load balancing. In Spark, RNN-DBSCAN will also require communicating between different nodes to exchange information about the reverse nearest neighbor.

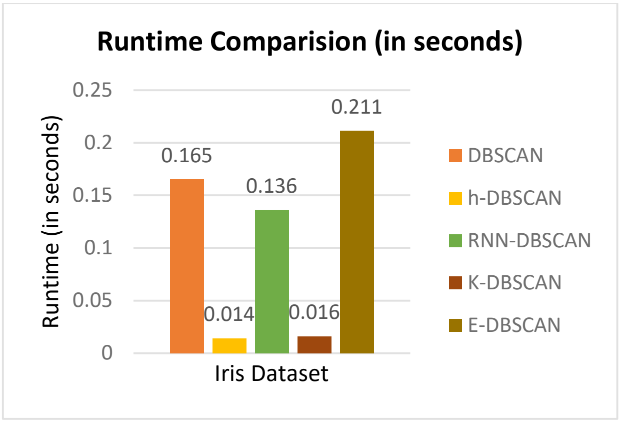

Fig. 1. Execution time comparison of algorithms. Comparison is based on Iris dataset.

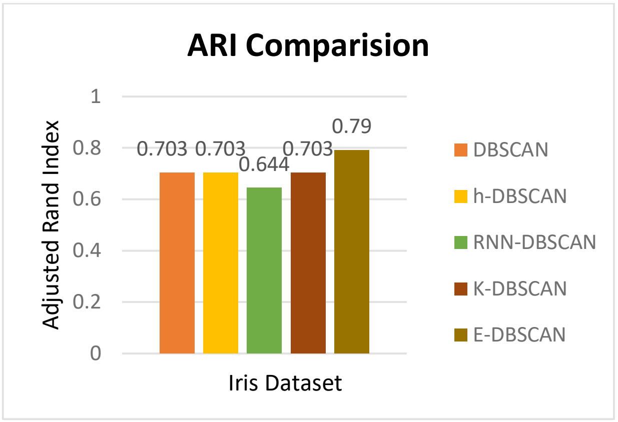

Fig. 2. Quality comparison of algorithms. Quality Metric: ARI. Comparison is based on Iris dataset.

TABLE II. Comparison of the Major Algorithm

Sr. No.	Algorithm	Time Complexity	Input Parameters	Performance Metrics	Type of Dataset
1	RNN-DBSCAN [3]	$O(k.n^2)$ *k*: Nearest-Neighbor *n*: Number of data points	*k* *k*: Nearest-Neighbor	ARI, NMI	Artificial and Real World
2	KNN-BLOCK DBSCAN [4]	$O(n^2)$ *n*: Number of data points	*eps, minPts*	Omega-Index, NMI, precision, recall, F1-score	Real World
3	µDBSCAN-D [14]	$O((n/p)\ log(m/p)\ +\ (n/p)\ log(r/p))\ +\ O(\delta \cdot n/p)$ *m*: Number of micro-clusters *r*: Average number of points in a micro-cluster *p*: Number of processing elements *δ*: Fraction of n/p points lying in the boundaries	*eps, minPts*	No performance metric used	Real World
4	K-DBSCAN [7]	$O(Iter.k.n)\ +\ ((n/k)^2 + (k(k-1)/2)m^2)$ *n*: Number of data points *k*:Number of center points *Iter*: Number of iterated computations *m*: *n*/k	*eps, minPts, k, MaxIter* *k*: Number of center points	Davies-Bouldin Metric, Silhouette Metric	Real World

			5MaxIter: Maximum n6umber of iterated computations		
5	MR-DBSCAN [9]	Not Discussed	*eps, minPts*	No performance metric used	Real World
6	BLOCK-DBSCAN [5]	*O(n.logn)* *n*: Number of data points	*eps, minPts*	Accuracy	Real World
7	E-DBSCAN [2]	Not Discussed	ε_{Max}, ε_{Min}, *minPts*	ARI, NMI	Artificial and Real World
8	DBSCAN-GM [15]	Not Discussed (Runtime is 3 times that of DBSCAN)	No input parameters r	PCC, Validity	Real World
9	SEED-DBSCAN [8]	*O(n+k.m)* *n*: Number of data points *k*: Max. size of partial clusters *m*: number of partial clusters	*eps, minPts*	No performance metric used	Artificial

KNN-BLOCK DBSCAN [4] and BLOCK DBSCAN [5] are fast approximate algorithms that are suitable for high-dimensional data. When tested on real-world datasets, they achieve high precision and produce almost identical clusters to DBSCAN. SEED-DBSCAN [8] and MR-DBSCAN [9] show good scalable performance when tested on large-scale datasets. PARDICLE [12] exhibit decent accuracy without making a compromise on the execution times.

The main challenge of making algorithms scalable is the requirement of having all the crucial subtasks parallelized. There are other challenges like data partitioning so that desirable load balancing can be achieved. MR-DBSCAN [9] uses the MapReduce framework to efficiently parallelize all the subtasks and a cost based partitioning system to ensure load balancing. This makes the algorithm scalable and it performs well even on datasets containing 1.2 billion data points.

V. Future Enhancements

From the survey, the following future enhancements to address the limitations of the current algorithms have been identified:

- It is observed that soft clustering algorithms, like Fuzzy DBSCAN [11] and E-DBSCAN [2], have higher runtimes when dealing with large datasets. In the future, they can be optimized to reduce their execution times.
- Algorithms with fast execution times, like h-DBSCAN [16] and K-DBSCAN [7], suffer from lower cluster accuracy. In the future, work can be done to improve their cluster quality.
- Proposing an effective density-based algorithm to apply in real-world areas like satellite image segmentation and spatio-temporal analysis remains to be pursued in the future.
- The application of KNN-BLOCK DBSCAN [4] and BLOCK-DBSCAN [5] on large-scale real-world datasets can be undertaken in the future.
- The application of DBSCAN-GM [15] to medical diagnosis data and the upkeep of case bases in case-based reasoning systems remain to be done.
- MR-DBSCAN [9] can be improved to support high-dimensional data.

Overall, from the comparisons of the above algorithms, none of the algorithms offer the ability to tackle all the issues. There is a need to propose an effective algorithm that offers good clustering quality along with low execution speeds for the purpose of big data clustering.

VI. Conclusion

The clustering of big data leads to many challenges like increased execution times, dealing with noise and high-dimensional data. The use of density-based clustering for big data has proven to be effective and promising. This paper presents an assessment of the current advancements in DBSCAN algorithms for clustering large-scale datasets. The paper compares the diverse qualities and evaluation metrics of the algorithms. Finally, it identifies potential research gaps and outlines future work that needs to be undertaken.

References

[1] An, Xiaoya, et al. "Strp-dbscan: A parallel dbscan algorithm based on spatial-temporal random partitioning for clustering trajectory data." Applied Sciences 13.20 (2023): 11122.

[2] Bessrour, Malek, Zied Elouedi, and Eric Lefevre. "E-DBSCAN: An evidential version of the DBSCAN method." 2020 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2020.

[3] Bryant, Avory, and Krzysztof Cios. "RNN-DBSCAN: A density-based clustering algorithm using reverse nearest neighbor density estimates." IEEE Transactions on Knowledge and Data Engineering 30.6 (2017): 1109-1121.

[4] Chen, Yewang, et al. "KNN-BLOCK DBSCAN: Fast clustering for large-scale data." IEEE transactions on systems, man, and cybernetics: systems 51.6 (2019): 3939-3953.

[5] Chen, Yewang, et al. "BLOCK-DBSCAN: Fast clustering for large scale data." Pattern Recognition 109 (2021): 107624.

[6] Ester, Martin, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." kdd. Vol. 96. No. 34. 1996.

[7] Gholizadeh, Nahid, Hamid Saadatfar, and Nooshin Hanafi. "K-DBSCAN: An improved DBSCAN algorithm for big data." The Journal of supercomputing 77 (2021): 6214-6235.

[8] Han, Dianwei, et al. "A novel scalable DBSCAN algorithm with Spark." 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2016.

[9] He, Yaobin, et al. "MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data." Frontiers of Computer Science 8 (2014): 83-99.

[10] Huang, Xiaogang, et al. "GriT-DBSCAN: A spatial clustering algorithm for very large databases." Pattern Recognition 142 (2023): 109658.

[11] Ienco, Dino, and Gloria Bordogna. "Fuzzy extensions of the DBScan clustering algorithm." Soft Computing 22.5 (2018): 1719-1730

[12] Patwary, Md Mostofa Ali, et al. "Pardicle: Parallel approximate density-based clustering." SC'14: Proceedings of the International

Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2014.

[13] Reddy, K. Shyam Sunder, and C. Shoba Bindu. "A review on density-based clustering algorithms for big data analysis." 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC). IEEE, 2017.

[14] Sarma, Aditya, et al. "μDBSCAN: an exact scalable DBSCAN algorithm for big data exploiting spatial locality." 2019 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2019.

[15] [Smiti, Abir, and Zied Elouedi. "Dbscan-gm: An improved clustering method based on gaussian means and dbscan techniques." 2012 IEEE 16th international conference on intelligent engineering systems (INES). IEEE, 2012.

[16] Weng, Shaoyuan, Jin Gou, and Zongwen Fan. "$ h $-DBSCAN: A simple fast DBSCAN algorithm for big data." Asian Conference on Machine Learning. PMLR, 2021.

[17] Zerhari, Btissam, Ayoub Ait Lahcen, and Salma Mouline. "Big data clustering: Algorithms and challenges." Proc. of Int. Conf. on Big Data, Cloud and Applications (BDCA'15). 2015

[18] Zhang, Xinliang, and Shibo Zhou. "WOA-DBSCAN: Application of Whale Optimization Algorithm in DBSCAN Parameter Adaption." IEEE Access (2023).