Parallel DBSCAN-Martingale estimation of the number of concepts for automatic satellite image clustering

Ilias Gialampoukidis[1][0000−0002−5234−9795], Stelios Andreadis[1][0000−0002−5519−1962], Nick Pantelidis[1], Sameed Hayat[2], Li Zhong[2], Marios Bakratsas[1], Dennis Hoppe[2], Stefanos Vrochidis[1][0000−0002−2505−9178] and Ioannis Kompatsiaris[1][0000−0001−6447−9020]

[1] Information Technologies Institute, Centre for Research & Technology Hellas, Thessaloniki, Greece
{heliasgj,andreadisst,pantelidisnikos,mbakratsas,stefanos,ikom}@iti.gr
[2] University of Stuttgart – High Performance Computing Center Stuttgart, Stuttgart, Germany
{hpcshaya,li.zhong,hpcdhopp}@hlrs.de

Abstract. The necessity of organising big streams of Earth Observation (EO) data induces the efficient clustering of image patches, deriving from satellite imagery, into groups. Since the different concepts of the satellite image patches are not known a priori, DBSCAN-Martingale can be applied to estimate the number of the desired clusters. In this paper we provide a parallel version of the DBSCAN-Martingale algorithm and a framework for clustering EO data in an unsupervised way. The approach is evaluated on a benchmark dataset of Sentinel-2 images with ground-truth annotation and is also implemented on High Power Computing (HPC) infrastructure to demonstrate its scalability. Finally, a cost-benefit analysis is conducted to find the optimal selection of reserved nodes for running the proposed algorithm, in relation to execution time and cost.

Keywords: Density-based clustering · Image clustering · High Power Computing.

1 Introduction

Recent years have witnessed an increasing availability of frequent and free-of-cost Earth Observation (EO) data, which promotes the involvement of satellite imagery analysis in several domains, from agriculture to disaster management, and from security and defence to blue economy. To support applications in this wide range of domains, various satellite-based solutions are proposed by the scientific community, with a particular focus on artificial intelligence methods and machine-learning approaches [36,26].

Nonetheless, the large streams of time series of remotely sensed images need to be organised in a contextual manner. The multimodal dimension of multispectral satellite images, such as the Sentinel-2 images, require effective and efficient management of the associated metadata, so as to group them into clusters. The lack of training data in the EO domain results, though, to the inability of the EO downstream sector to apply supervised machine learning techniques for pattern recognition in satellite images. The problem is proved challenging when the number of clusters, that satellite image patches can be grouped in, is not known a priori and the labels are not known or not annotated. To that end, density-based algorithms are suitable [24], since they do not require the number of clusters as input, contrary to other clustering approaches, such as k-means. Density-based algorithms require as input two other parameters: the minimum number of points required to form a cluster, namely $minPts$, and the neighbourhood radius ϵ (density level). Nevertheless, their estimation is hard to be made and often requires several executions and combination of outputs from multiple density levels.

Over twenty years ago, [11] introduced the now popular DBSCAN, a density-based algorithm for discovering clusters in large spatial databases with noise. DBSCAN is still used both in research and real-world applications, while it has inspired numerous extensions. In particular, DBSCAN-Martingale [14] estimates the number of clusters by optimizing a probabilistic process, which involves randomness in the selection of density parameter. In this work we propose a parallel version of the DBSCAN-Martingale, which is validated in clustering large data assets of satellite image patches from Sentinel-2 imagery. Furthermore, the implementation of the proposed algorithm is transferred to High Power Computing (HPC) infrastructure so as to show how computational limitations can be overcome, to prove scalability and to achieve high efficiency.

The remainder of the paper is organised as follows. First, in Section 2 we discuss related publications that concern the optimisation of DBSCAN, either by optimising its parameters or its performance time. Section 3 presents the proposed algorithm, along with the necessary background, and describes the HPC infrastructure where the algorithm is transferred. Section 4 continues with the description of the dataset used for the evaluation of the algorithm, the results of the experiments, and a cost-benefit analysis to examine how time and cost range in relation to the number of used nodes. Finally, Section 5 concludes with a summary of the presented work and future steps.

2 Related Work

Due to its high popularity, the scientific community has focused on the improvement of DBSCAN, mainly in two directions: the optimisation of DBSCAN's parameters, i.e. $minPts$ and ϵ, and the performance time, proposing faster versions of the algorithm.

Regarding the parameters optimisation, the algorithm in [9] divides the data set into multiple data regions, sets the appropriate parameters for each data region for local clustering, and finally merges the data regions. [23] focus on minimizing the additional computation required to determine the parameters by using the approximate adaptive ϵ-distance for each density while finding the clusters with varying densities that original DBSCAN cannot find. [33] proposes a self-adaption grey DBSCAN algorithm that automatically selects its parameters, whereas [29] calculate the parameters $minPts$ and ϵ based on the optimal k-value, selected after multiple iterations of k-clustering. Furthermore, in the works of [17] and [28], the problem of proper parameterisation is solved by applying the density peak algorithm and the natural neighbour algorithm respectively.

As far as it concerns the performance, several recent works present revisions of DBSCAN, aiming to reduce the computation time. [37] target the time spent in the input/output (reading/writing data), while [12] inspects the neighbourhoods of only a subset of the objects in the dataset, similarly to the modification of DBSCAN presented in [21] that requires computing only the densities for a chosen subset of points. Likewise, the approaches of [40] and [7] run on selected core points, the first based on locality sensitive hashing and the latter on k-nearest neighbours.

In addition, [10] apply a novel randomized k-centre clustering idea to decrease the complexity of range query, which is a time-consuming step in DBSCAN. [5] improve the efficiency with Ada-DBSCAN, an extension that consists of a data block splitter and a data block merger, coordinated by local clustering and global clustering, and [27] with

ADBSCAN, which identifies local high-density samples utilizing the inherent properties of the nearest neighbour graph. Other extensions that present good performance are the 3W-DBSCAN [46], the Bisecting Min Max DBSCAN [22], and the method in [31] that only computes the distances between the object and its nearby neighbours.

Towards decreasing the complexity of DBSCAN, [2] avoid spending time on every edge in the neighbourhood graph by working with box graphs, while [19] exploit the Warshall algorithm to mitigate its complexity. In [4], problem complexity reduces by using a single parameter (choice of k nearest neighbours) and by handling large variations in cluster density (heterogeneous density). Moreover, [35] use a combination of distance-based aggregation by overlaying the data with customized grids and [3] utilize bitmap indexing to support efficient neighbour grid queries.

The upsurge of distributed and high-performance computing technology has motivated scientists to propose various parallel implementations of DBSCAN. [45] present a framework that divides data into partitions, calculates local DBSCAN results, and merges local results based on a merging graph. [8] enable parallel computing by a divide-and-conquer method that includes a simplified k-mean partitioning process and a reachable partition index, and [15] present a new approach that supports real-time clustering of data based on continuous cluster checkpointing. Additional parallelised versions of DBSCAN are found in the works of [25], [18] applying MapReduce, [47] with a parallel grid clustering algorithm, and [38], who developed a scalable distributed implementation of their own μDBSCAN.

[44] and [20] manage to parallelise DBSCAN by using Quadtree data structure. In the latter, the solution distributes the dataset into smaller chunks and then utilizes the parallel programming frameworks such as Map-Reduce to provide an infrastructure to store and process these small chunks of data. [30] utilizes Cover Tree to retrieve neighbours for each point in parallel and the triangle inequality to filter many unnecessary distance computations. Alternatively, AnyDBC by [34], instead of performing range queries for all objects, iteratively learns the current cluster structure of the data and selects a few most promising objects for refining clusters at each iteration.

Finally, a further step to distributed implementations of DBSCAN is the optimisation of the involved stages. [6] propose improvements both in data partitioning stage and in merging stage. [16] adopt a partitioning strategy based on kdtree, similarly to [39], and a new merging technique by mapping the relationship between the local points and their bordering neighbours. [43] use a Hilbert curve to identify the centres for initial partitioning, [32] optimise it with distance matrix and R-Tree based methods, and [41] suggest a cell-based data partitioning scheme, which randomly distributes small cells rather than the points themselves.

3 Background and Methodology

3.1 Background and notation

In this section we provide the notation that we deem necessary before presenting the proposed algorithm. As already mentioned, DBSCAN [11] has two parameters: ϵ and $minPts$. The parameter ϵ defines the radius of neighborhood around a point x (density

level), while the parameter $minPts$ is the minimum number of neighbors within ϵ radius. $minPts$ is usually predefined based on the minimum size of the desired clusters. On the other hand, the density level ϵ is hard to be estimated and, even so, the algorithm is not able to output all clusters using one single density level. The cluster structure is visualised using the OPTICS [1] plot of reachability distances, where the dents represent clusters and it is also possible to observe the density level at which the desired clusters are extracted. For each density level ϵ, the output of DBSCAN is one clustering vector and is denoted by $C_{DBSCAN_{(\epsilon)}}$. In detail, given a dataset of n-instances, DBSCAN provides as output a clustering vector C with values the cluster IDs $C[j]$ for each instance $j = 1, 2, ..., n$, assigning each element j to a cluster. In case the j-th element is marked as noise, then the cluster ID is zero ($C[j] = 0$). For further definitions of the DBSCAN-Martingale process and its extension for providing the estimation of the number of clusters $\hat{k}$, the reader is referred to [13,14].

3.2 The proposed Parallel-DBSCAN-Martingale

The proposed parallel version of the DBSCAN-Martingale algorithm, which requires R realisations and T iterations of the DBSCAN algorithm, each one allocated in different nodes and cores, so that the algorithm can scale up using an HPC infrastructure, is presented step-by-step in Algorithm 1 and the complete framework for its implementation is illustrated in Fig. 1.

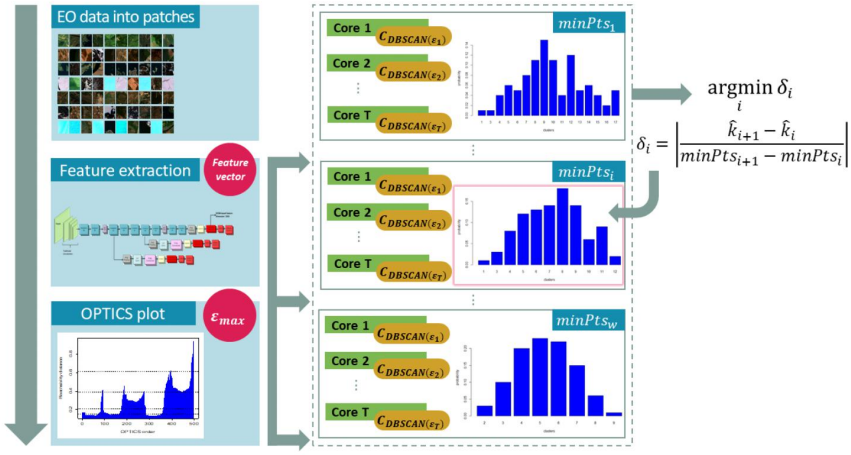

Fig. 1: The Parallel-DBSCAN-Martingale framework

Given a dataset of satellite images, which are cropped into patches, any feature extraction technique can be applied and the resulted feature vectors can be used for

clustering. As mentioned in section 3.1, the OPTICS plot is used for selecting the maximum ϵ and then DBSCAN-Martingale is executed on different cores in parallel, for multiple values of $minPts$, in order to detect the most probable number of clusters, which is the final output of the framework.

Algorithm 1 Parallel-DBSCAN-Martingale(N, T, R, w) return $\hat{k}$

Allocate N nodes and T cores
Set the number w of different values of $minPts$
Extract feature vectors per satellite image patch using any feature extraction method
Find ϵ_{max} using the maximum reachability distance from an OPTICS reachability plot
for $minPts \in \{minPts_1, minPts_2, ..., minPts_w\}$ **do**
    $clusters = \emptyset$, $k = 0$
    for $r = 1$ to R **do**
        Generate a random sample of T values in [0, ϵ_{max}]
        Sort the generated sample $\epsilon_t, t = 1, 2, ..., T$
        for $t = 1$ to T **do**
            compute $C_{DBSCAN(\epsilon_t)}$ distributed in each one of the T cores
            compute $C^{(t)}$ distributed in each one of the T cores
            update the cluster IDs
            update the vector C
            update $k = \max_j C[j]$
        end for
        $clusters$ =AppendTo($clusters, k$)
    end for
    $\hat{k}$ =mode($clusters$)
end for
compute $\delta_i = |\frac{\hat{k}_{i+1} - \hat{k}_i}{minPts_{i+1} - minPts_i}|, i = 1, 2, ..., w - 1$
compute $minPts$ such as: $minPts = \arg\min_i \delta_i$
find index i such that $minPts = minPts_i$
return $\hat{k}_i$

3.3 HPC Infrastructure

In order to demonstrate the scalability of the proposed algorithm and achieve the best performance of clustering, Parallel-DBSCAN-Martingale is also run on HPC infrastructure. On these clusters, the environments are setup so that machine learning and deep learning algorithms can be processed effectively. In particular, the NEC Cluster platform (Vulcan) is selected to accelerate the computation, shorten the running time and improve the performance. It is a heterogeneous cluster with currently 761 nodes of different types (Memory, CPU, Disk, Accelerators). For the experiments of Section 4, CascadeLake nodes (clx-25) have been used, which include Infiniband interconnect for high-speed transmission and high memory 384GB. Especially, near real-time data analytics are enabled to be performed with the high amount of RAM. The high performance data analytics (HPDA) system naturally enables different kinds of machine

learning and data analysis tasks, including the clustering task. The technical details of the NEC Cluster (Vulcan) are summarised in Table 1.

Table 1: Technical specification of the NEC Cluster (Vulcan).

	Vulcan CascadeLake nodes (clx-25)
Total number of nodes	84 nodes
Processors per node	2x Intel Xeon Gold 6248, 40 cores total @ 2.50GHz
1 RAM per node	384 GB
Disk storage per node	No local storage
External parallel file system	∼500TByte (shared), throughput of 6 GB/s
Operating system	CentOS 7
Available Standard Software	PBSPro, Apache Hadoop, Apache Spark, etc.
Main Programming Languages / Tools	C++, MPI, Python, etc.

4 Evaluation

4.1 Dataset Description

For the evaluation of the proposed algorithm, we have selected the *BigEarthNet*[1] dataset [42], which contains ground-truth annotation about Sentinel-2 satellite images and counts 590,326 patches of size 120x120 pixels. Each patch may contain one or more of the following labels: *water*, *rice*, *urban*, *vineyards*, *forest*, *bare rock*, and *snow*.

Table 2: Number of satellite image patches per label

Labels	**Single labeled dataset**	**Subset 1**	**Subset 2**	**Subset 3**
Water	71,549	100	1,000	30,000
Rice	1,122	100	1,000	-
Urban	33,411	100	1,000	30,000
Vineyards	2,749	100	1,000	-
Forest	220,406	100	1,000	30,000
Bare rock	52	-	-	-
Snow	61,707	100	1,000	30,000
Total	390,996	600	6,000	120,000

By definition clustering cannot handle a multi-labelling problem, since each patch will be assigned to a single cluster. Therefore, we have excluded the patches whose ground truth is more than one label and produced a modified version of the dataset

[1] http://bigearth.net/

that can be seen in the column "Single labeled dataset" of Table 2. Due to the high imbalance in the number of patches for each class (e.g., 220k forest, 1k rice, 52 rock) and in order to evaluate the scalability of the algorithm, we have produced three subsets: one with 100 patches per label, one with 1,000 patches per label and one with 30,000; classes with less than 100/1,000/30,000 instances were omitted in the preparation of the respective subset. All subsets can be seen in detail in Table 2.

It should be also noted here that in the following experiments the feature extraction stage generated one vector per satellite image patch using a Deep Convolutional Neural Network layer, but as stated in 3.2, the vector representation could be of any type, such as color histograms or other feature vectors.

4.2 Results

For our experiments, T was set to 5, ϵ_{max} to 10, and w to 12, with $minPts$ varying from 5 to 16. For each $minPts = 5, 6, ..., 16$ the proposed Parallel-DBSCAN-Martingale estimated the probabilities of the number of clusters (for reasons of space, only the plot for $minPts = 11$ can be seen in Fig. 2a) and generated the final result in an unsupervised way, searching for the most "stable" number of clusters as an optimal solution (Fig. 2b), i.e. 6 for Subset 1.

Estimating the number of clusters with the highest probability for every $minPts$ requires significant computational effort. For Subset 1 (600 patches) and 100 realisations, it was possible to run Parallel-DBSCAN-Martingale in a personal computer with 4 cores and achieve an execution time of 12 seconds. However, it did not manage to run for Subset 2 (6,000 patches) or Subset 3 (120,000), indicating that it needs to be distributed in multiple processing nodes for scalability. Four runs were successful on the HPC infrastructure, which has been presented in 3.3, for different parameters, i.e. the size of the dataset, the number of realisations and the available cores. All the details of the runs (parameters and execution time) can be found in Table 3. It should be highlighted here that with an HPC infrastructure, the proposed algorithm is able to cluster even within 40 minutes (2,400 seconds) satellite image patches that cover more than 172k square kilometers (Run 3).

Table 3: Execution time of Parallel-DBSCAN-Martingale for different parameters

		PC	**HPC**			
		Run 1	**Run 1**	**Run 2**	**Run 3**	**Run 4**
dataset	**number of patches**	600	6,000	6,000	120,000	120,000
	size of patches (MB)	50	501	501	10,020	10,020
	square kilometers	864	8,640	8,640	172,800	172,800
T		5	5	5	5	5
R		100	10	1,000	10	1,000
available cores		4	216	108	1,044	80
time (seconds)		12	12	160	2,400	12,400

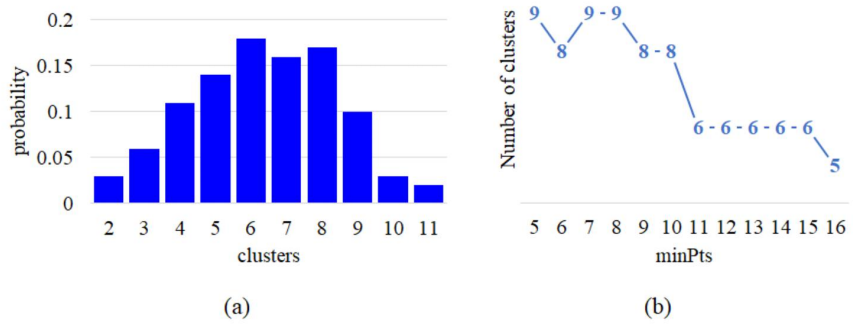

Fig. 2: (a) Probability of the number of clusters for $minPts = 11$ (b) Most probable number of clusters per $minPts$

4.3 Cost-Benefit Analysis

After achieving the parallel execution of DBSCAN-Martingale on HPC, a further analysis has been conducted in order to investigate how time and cost range in relation to the number of reserved nodes. Regarding prices, we refer to HLRS fee schedule for 2020[2], where the computing cost for Cascade-lake 384GB nodes is €1.31 per node*hour.

Table 4: Time & cost analysis of multi-node execution of Parallel-DBSCAN-Martingale

Reserved Nodes	Execution Time (hours)	Execution Cost (€)
1 (40 cores)	1.672	2.62
2 (80 cores)	0.822	2.62
3 (120 cores)	0.562	3.93
4 (160 cores)	0.533	5.24
5 (200 cores)	0.527	6.55
6 (240 cores)	0.273	7.86

The fluctuation of execution time (in hours) and execution cost (in euros) per number of nodes when running Parallel-DBSCAN-Martingale ($T = 5$, $R = 60$) is reported in Table 4 and also displayed as a line chart in Figure 3.

Moreover, we define marginal cost m_i as:

$$m_i = \frac{v_{n_i} - v_{n_{i-1}}}{t_{n_i} - t_{n_{i-1}}}$$

[2] https://www.hlrs.de/solutions-services/academic-users/legal-requirements/

Fig. 3: Execution Time (hours) and Execution Cost (€) vs Nodes

where v_c is the price value, t_c is the processing time and n is the number of nodes. The fluctuation of marginal cost m_i is illustrated in Figure 4 and shows that switching from 1 to 2 nodes is advantageous, since execution time is reduced by half with the exact same cost, while switching from 4 to 5 nodes is the least profitable option, considering that the cost increases, but the execution time remains almost the same.

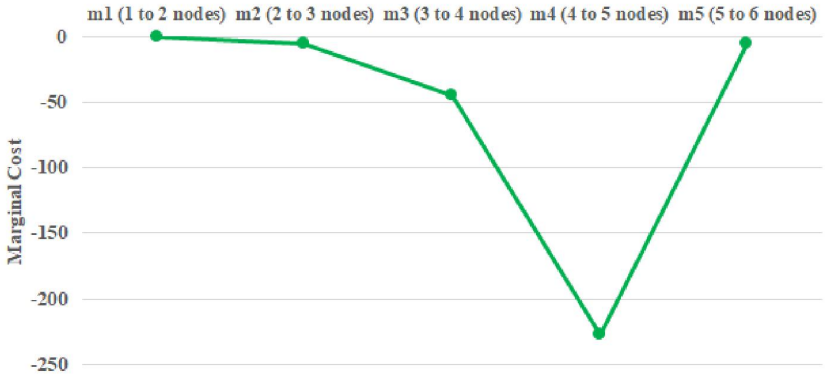

Fig. 4: Marginal cost while increasing number of nodes

5 Conclusion and Future Work

Clustering satellite image patches allows a fast grouping of Earth Observation data into clusters of similar semantic content, i.e. concepts such as water, snow, forest, rice,

etc. This process is unsupervised and does not require any label information or the number of clusters known a priori. In an operational level, once a cluster of satellite imagery patches is obtained and one of the patches is assigned a label (e.g., "this is a water surface"), then this label is propagated to all other members (patches) of the same cluster.

In this work we presented a parallel version of a state-of-the-art clustering algorithm, namely DBSCAN-Martingale, which estimates the number of clusters in an automatic way. The use of HPC allows us to distribute the processing task into several processing nodes, thus offering a scalable solution to the EO Big Data community. The parameters of the proposed density-based approach are also learned in an unsupervised way and a stable solution is searched with many executions and realisations of the process, to get the optimal value for the number of clusters. Once the optimal number of clusters is obtained, then any traditional clustering algorithm (e.g., k-means) can be applied, having as input the estimated number of clusters and the vector representation of the satellite image patches. Parallel-DBSCAN-Martingale was evaluated in a subset of the BigEarthNet dataset and achieved to estimate the correct number of clusters, i.e. labels of the patches. It was also run successfully on HPC infrastructure, proving its scalability and increasing its efficiency. Finally, a cost-benefit analysis was conducted to detect the optimal selection of nodes for running this particular algorithm on this particular infrastructure.

Future work will focus on further experiments on HPC to discover the optimal usage of multiple cores, as well as on the parallelisation of other parts of DBSCAN-Martingale, such as the realisations.

References

1. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure. ACM Sigmod record **28**(2), 49–60 (1999)
2. de Berg, M., Gunawan, A., Roeloffzen, M.: Faster dbscan and hdbscan in low-dimensional euclidean spaces. International Journal of Computational Geometry & Applications **29**(01), 21–47 (2019)
3. Boonchoo, T., Ao, X., Liu, Y., Zhao, W., Zhuang, F., He, Q.: Grid-based dbscan: Indexing and inference. Pattern Recognition **90**, 271–284 (2019)
4. Bryant, A., Cios, K.: Rnn-dbscan: A density-based clustering algorithm using reverse nearest neighbor density estimates. IEEE Transactions on Knowledge and Data Engineering **30**(6), 1109–1121 (2017)
5. Cai, Z., Wang, J., He, K.: Adaptive density-based spatial clustering for massive data analysis. IEEE Access **8**, 23346–23358 (2020)
6. Chen, G., Cheng, Y., Jing, W.: Dbscan-psm: an improvement method of dbscan algorithm on spark. International Journal of High Performance Computing and Networking **13**(4), 417–426 (2019)
7. Chen, Y., Zhou, L., Pei, S., Yu, Z., Chen, Y., Liu, X., Du, J., Xiong, N.: Knn-block dbscan: Fast clustering for large-scale data. IEEE Transactions on Systems, Man, and Cybernetics: Systems (2019)
8. Deng, C., Song, J., Cai, S., Sun, R., Shi, Y., Hao, S.: K-dbscan: an efficient density-based clustering algorithm supports parallel computing. International Journal of Simulation and Process Modelling **13**(5), 496–505 (2018)

9. Diao, K., Liang, Y., Fan, J.: An improved dbscan algorithm using local parameters. In: International CCF Conference on Artificial Intelligence. pp. 3–12. Springer (2018)
10. Ding, H., Yang, F.: On metric dbscan with low doubling dimension. arXiv preprint arXiv:2002.11933 (2020)
11. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: kdd. vol. 96, pp. 226–231 (1996)
12. Galán, S.F.: Comparative evaluation of region query strategies for dbscan clustering. Information Sciences **502**, 76–90 (2019)
13. Gialampoukidis, I., Vrochidis, S., Kompatsiaris, I.: A hybrid framework for news clustering based on the dbscan-martingale and lda. In: International Conference on Machine Learning and Data Mining in Pattern Recognition. pp. 170–184. Springer (2016)
14. Gialampoukidis, I., Vrochidis, S., Kompatsiaris, I., Antoniou, I.: Probabilistic density-based estimation of the number of clusters using the dbscan-martingale process. Pattern Recognition Letters **123**, 23–30 (2019)
15. Gong, Y., Sinnott, R.O., Rimba, P.: Rt-dbscan: Real-time parallel clustering of spatio-temporal data using spark-streaming. In: International Conference on Computational Science. pp. 524–539. Springer (2018)
16. Han, D., Agrawal, A., Liao, W.k., Choudhary, A.: Parallel dbscan algorithm using a data partitioning strategy with spark implementation. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 305–312. IEEE (2018)
17. Hou, J., Lv, C., Zhang, A., Xu, E.: Merging dbscan and density peak for robust clustering. In: International Conference on Artificial Neural Networks. pp. 595–610. Springer (2019)
18. Hu, X., Liu, L., Qiu, N., Yang, D., Li, M.: A mapreduce-based improvement algorithm for dbscan. Journal of Algorithms & Computational Technology **12**(1), 53–61 (2018)
19. Huang, M., Yan, Y., Xu, L., Ye, L.: Using warshall to solve the density-linked density clustering algorithm. American Journal of Applied Mathematics **8**(1), 11–16 (2020)
20. Ibrahim, R., Shafiq, M.O.: Towards a new approach for empowering the mr-dbscan clustering for massive data using quadtree. In: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). pp. 91–98. IEEE (2018)
21. Jang, J., Jiang, H.: Dbscan++: Towards fast and scalable density clustering. In: International Conference on Machine Learning. pp. 3019–3029. PMLR (2019)
22. Johnson, T., Prabhu, K., Parvatkar, S., Naik, A., Temkar, P.: The bisecting min max dbscan algorithm
23. Kim, J.H., Choi, J.H., Yoo, K.H., Nasridinov, A.: Aa-dbscan: an approximate adaptive dbscan for finding clusters with varying densities. Journal of Supercomputing **75**(1) (2019)
24. Kriegel, H.P., Kröger, P., Sander, J., Zimek, A.: Density-based clustering. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **1**(3), 231–240 (2011)
25. Kumari, A., Shrivastava, V., Pandey, A.: Reduction of dbscan time complexity for data mining using parallel computing techniques.
26. Lary, D.J., Alavi, A.H., Gandomi, A.H., Walker, A.L.: Machine learning in geosciences and remote sensing. Geoscience Frontiers **7**(1), 3–10 (2016)
27. Li, H., Liu, X., Li, T., Gan, R.: A novel density-based clustering algorithm using nearest neighbor graph. Pattern Recognition **102**, 107206 (2020)
28. Li, J., Chen, Y.: Improved dbscan algorithm based on natural neighbors. Modern Computer **13** (2018)
29. Li, J., Han, X., Jiang, J., Hu, Y., Liu, L.: An efficient clustering method for dbscan geographic spatio-temporal large data with improved parameter optimization. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences **42**, 581–584 (2020)

30. Li, S.S.: An improved dbscan algorithm based on the neighbor similarity and fast nearest neighbor query. Ieee Access **8**, 47468–47476 (2020)
31. Lin, P., Hong, Z., Feng, W., Li, Y., Wu, L.: Design and implementation of an improved dbscan algorithm. In: 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC). pp. 1834–1839. IEEE (2019)
32. Liyang, L., Hongzhen, S., Shen, W., Jinyu, L.: Parallel implementation of dbscan algorithm based on spark
33. Lu, S.: Self-adaption grey dbscan clustering. arXiv preprint arXiv:1912.11477 (2019)
34. Mai, S.T., Assent, I., Jacobsen, J., Dieu, M.S.: Anytime parallel density-based clustering. Data mining and knowledge discovery **32**(4), 1121–1176 (2018)
35. Mathur, V., Mehta, J., Singh, S.: Hca-dbscan: Hypercube accelerated density based spatial clustering for applications with noise. arXiv preprint arXiv:1912.00323 (2019)
36. Maxwell, A.E., Warner, T.A., Fang, F.: Implementation of machine-learning classification in remote sensing: An applied review. International Journal of Remote Sensing **39**(9), 2784–2817 (2018)
37. Pandey, S., Samal, M., Mohanty, S.K.: An snn-dbscan based clustering algorithm for big data. Advanced Computing and Intelligent Engineering pp. 127–137 (2020)
38. Sarma, A., Goyal, P., Kumari, S., Wani, A., Challa, J.S., Islam, S., Goyal, N.: μdbscan: an exact scalable dbscan algorithm for big data exploiting spatial locality. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER). pp. 1–11. IEEE (2019)
39. Shibla, T., Kumar, K.S.: Improving efficiency of dbscan by parallelizing kd-tree using spark. In: 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS). pp. 1197–1203. IEEE (2018)
40. Shiqiu, Y., Qingsheng, Z.: Dbscan clustering algorithm based on locality sensitive hashing. In: Journal of Physics: Conference Series. vol. 1314, p. 012177. IOP Publishing (2019)
41. Song, H., Lee, J.G.: Rp-dbscan: A superfast parallel dbscan algorithm based on random partitioning. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1173–1187 (2018)
42. Sumbul, G., de Wall, A., Kreuziger, T., Marcelino, F., Costa, H., Benevides, P., Caetano, M., Demir, B., Markl, V.: Bigearthnet-mm: A large scale multi-modal multi-label benchmark archive for remote sensing image classification and retrieval. arXiv preprint arXiv:2105.07921 (2021)
43. Tyercha, E.R., Kazmaier, G.S., Gildhoff, H., Pekel, I., Volker, L., Grouisborn, T.: Hilbert curve partitioning for parallelization of dbscan (Jun 11 2019), uS Patent 10,318,557
44. Wang, Y., Gu, Y., Shun, J.: Theoretically-efficient and practical parallel dbscan. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. pp. 2555–2571 (2020)
45. Yang, K., Gao, Y., Ma, R., Chen, L., Wu, S., Chen, G.: Dbscan-ms: Distributed density-based clustering in metric spaces. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE). pp. 1346–1357. IEEE (2019)
46. Yu, H., Chen, L., Yao, J., Wang, X.: A three-way clustering method based on an improved dbscan algorithm. Physica A: Statistical Mechanics and its Applications **535**, 122289 (2019)
47. ZHOU, G.j.: Research on parallel design of dbscan clustering algorithm in spatial data mining. DEStech Transactions on Engineering and Technology Research (ecar) (2018)