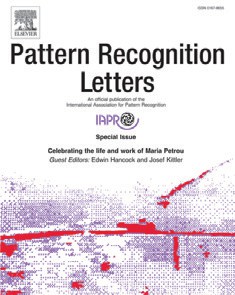
[Pattern Recognition Letters 117 (2019) 90–96](https://doi.org/10.1016/j.patrec.2018.12.010)



Contents lists available at [ScienceDirect](http://www.ScienceDirect.com/)

Pattern Recognition Letters

journal homepage: [www.elsevier.com/locate/patrec](http://www.elsevier.com/locate/patrec)



Sampling approaches for applying DBSCAN to large datasets [✩](#_bookmark2)

Diego Luchi [a,](#_bookmark0)[∗,](#_bookmark3) Alexandre Loureiros Rodrigues [b,](#_bookmark1) Flávio Miguel Varejão [a](#_bookmark0)

a *Computer Science Department Federal University of the state of Espírito Santo, Vitória, Brazil*

b *Statistical Department Federal University of the state of Espírito Santo, Vitória, Brazil*

a r t i c l e i n f o a b s t r a c t

*Article history:*

Received 22 January 2018

Available online 11 December 2018

*MSC:*

41A05

41A10

65D05

65D17

*Keywords:* Clustering Sampling DBSCAN

DBSCAN is a classic clustering method for identifying clusters of different shapes and isolate noisy pat- terns. Despite these qualities, many articles in the literature address the scalability problem of DBSCAN. This work presents two methods to generate a good sample for the DBSCAN algorithm. The execution time decreases due to the reduction in the number of patterns presented to DBSCAN. One method is an improvement of the Rough-DBSCAN and presented consistently better results. The second is a new heuristic called I-DBSCAN capable of adapting and generating good results for all datasets without the need of any additional parameter.

© 2018 Elsevier B.V. All rights reserved.

# Introduction

The DBSCAN (Density-based spatial clustering of applications with noise), proposed by [[1]](#_bookmark18), is an effective clustering technique and one of the most studied algorithms in the ﬁeld, as it can be seen in [[2]](#_bookmark19) and [[3,4]](#_bookmark20). The DBSCAN algorithm is known for its ability to ﬁnd groups with arbitrary shapes and its capacity to isolate noise from the rest of the data in the clustering process [[5]](#_bookmark21). For comparison, the Single-Linkage clustering (SLINK) [[6]](#_bookmark22) and the k-means algorithm [[7]](#_bookmark23), two of the most traditional clustering methods in the literature, are not robust or ﬂexible enough to be considered in many clustering applications. The former, although known for its capability to detect clusters with arbitrary shapes, is very sensitive to noise. The latter has a severe limitation of detect- ing only compact and spherical shaped clusters [[8]](#_bookmark24). In this context, DBSCAN appears as an attractive solution to many clustering prob- lems. However, like several other clustering algorithms, it does not scale well for large datasets due to its high time complexity [[9]](#_bookmark25).

Several ways to overcome this high computational cost have been proposed. A survey of some approaches can be found in

[[10]](#_bookmark26) and [[11]](#_bookmark27). Many of these techniques aim to solve the near- est neighbor query with data structures, such as R∗-Tree [[12]](#_bookmark28), M- Tree [[13]](#_bookmark29), X-Tree [[14]](#_bookmark30), or by putting a lattice over the data domain

[[15]](#_bookmark31). Those structures help to perform the nearest neighbor queries

✩ **Conﬂict of interest.** The authors declare no conﬂict of interest.

∗ Corresponding author.

*E-mail address:* [dluchi@inf.ufes.br](mailto:dluchi@inf.ufes.br) (D. Luchi).

more eﬃciently for low-dimensional datasets, but if the number of dimensions is high the beneﬁt of such data structures quickly vanishes. Such degradation of eﬃciency for many data structure frameworks is a well-documented effect [[16,17]](#_bookmark32).

For high-dimensional datasets, two viable alternatives are typ- ically used to reduce the DBSCAN runtime. The ﬁrst approach fo- cus on solving the query of the nearest neighbors more eﬃciently

[[18]](#_bookmark33) or in an *approximate* way, for example, to use hash meth- ods such as *locality-sensitive hashing* (LSH) [[19]](#_bookmark34). In [[18]](#_bookmark33) the authors proposed a clever way of avoiding unnecessary distance calcula- tions for the nearest neighbor queries by taking advantage of tri- angular inequality. This proposal scales well for high-dimensional datasets unlike the aforementioned data structures used for spa- tial indexing. Therefore, it is an effective way to accelerate the ex- ecution of DBSCAN. However, this proposal has a greater neigh- borhood search complexity than approximate approaches such as LSH. On the other hand, by adopting hash schemes the density estimation for each element is susceptible to errors due to colli- sions in the hash table [[20]](#_bookmark35). The second approach uses sampling techniques to improve the computational performance of DBSCAN to ﬁnd an approximate solution even for large datasets. In those situations a small set of instances is chosen to represent the en- tire dataset. This size reduction greatly reduces the DBSCAN run- ning time. While some approaches randomly generate the sample for the DBSCAN [[21]](#_bookmark36), others generate a sample from the output of clustering algorithms that have lower computational complex- ity, such as CLARANS in [[22]](#_bookmark38) and Leader in [[23]](#_bookmark39).

<https://doi.org/10.1016/j.patrec.2018.12.010>

0167-8655/© 2018 Elsevier B.V. All rights reserved.

This work follows the latter approach and is motivated by the work in [[23]](#_bookmark39), which proposes a way to reduce the size of the dataset by applying the single pass Leader clustering algorithm

[[24]](#_bookmark40) on the data. After the clustering step, the authors propose a method to decide whether or not the leaders of each cluster will compose the sample. This decision is made based on the den- sity of each leader. However, with the output of the Leader al- gorithm alone it is very diﬃcult to calculate their densities cor- rectly. The authors present a clever way to approximate the den- sities, named *rough cardinality*. After running DBSCAN in the sam- ple, the ﬁnal partition is returned by assigning the remaining el- ements to their respective leader’s cluster. Viswanath and Babu

[[23]](#_bookmark39) call this method Rough-DBSCAN. Inspired by this work, this article presents two sampling methods to reduce the size of the dataset to carry out the DBSCAN algorithm. The ﬁrst proposal, called Rough∗-DBSCAN, is similar to Rough-DBSCAN, however the original Leader algorithm is modiﬁed to provide a trivial way to

estimate the density of each leader. In the second method, called I-DBSCAN, a heuristic technique to draw a sample from the lead- ers and the elements contained in the intersections of the clusters found by the Leader algorithm is proposed.

Experiments measured the performance of the proposed meth- ods by comparing their results with the one obtained by running the DBSCAN algorithm carried out over the entire dataset. While the ﬁrst proposal achieved a better result approximation than the original method, the second proposal was the fastest method still getting good results.

The remainder of this paper is organized as follows: [Section 2](#_bookmark5) brieﬂy introduces the DBSCAN, Leader and Rough- DBSCAN clustering algorithms; [Section 3](#_bookmark8) shows the Rough∗-

DBSCAN and the I-DBSCAN algorithms; [Section 4](#_bookmark14) describes experi-

ments where both methods proposed and the Rough-DBSCAN are applied on public domain datasets from the LIBSVM repository. Finally, [Section 5](#_bookmark37) draws some conclusions.

# DBSCAN, leader and rough-DBSCAN

This section shows a quick overview of DBSCAN, Leader, and Rough-DBSCAN algorithms.

* 1. *DBSCAN*

The DBSCAN algorithm, presented in [[1]](#_bookmark18), is one of the most suc- cessful clustering algorithms in the literature. Among its advan- tages, one can highlight its ability to identify clusters of various shapes and handle noisy data. DBSCAN ([Algorithm 1](#_bookmark4)) scans the entire dataset *D* (line 2) checking if each element *d* ∈ *D* has a den- sity above a certain threshold *minPts*. The calculation of the density of any element is done by counting how many elements are within a distance less than a certain value *E*. If the density is greater than *minPts* the element is considered a dense pattern (line 5) other- wise it will be temporarily classiﬁed as noise (line 6). If the ele- ment is dense, it will be assigned to a new cluster *C* (line 9) and a breadth-ﬁrst search (BFS) in the dense neighboring elements of *d* that have not yet been examined begins (lines 16–18). After all iterations of the inner loop (line 11), all elements without cluster, or temporary classiﬁed as noise, discovered by the BFS, is assigned to the cluster *C*.

Despite the advantages described in the in the preceding sec- tion, DBSCAN has a very bad scalability due to the functions per- formed in lines 4 and 16 of the [Algorithm 1](#_bookmark4). This function stands for nearest neighbors query (called NN-QUERY for short). It ﬁnds all elements of the dataset at a given distance from a given ele- ment. This operation is necessary to determine if the element is in a dense region or is a noisy pattern.

**Algorithm 1:** DBSCAN [[25]](#_bookmark41).

**Data**: Dataset *D*, distance *E*, minimum elements *minPts*

**1** *C* ← 0;

**2 for** *each d* ∈ *D* **do**

**3 if** *labeld is undeﬁned* **then**

**4** Neighbors *N* ←NN-QUERY*(D, d, E)*;

**5 if** |*N*| *< minPts* **then**

**6 ** label*d* ← noise;

**7 else**

**8** *C* ← *C* + 1;

**9** label*d* ← *C*;

**10** *S* ← *N* \ {*d*};

**11 for** *each q* ∈ *S* **do**

**12 if** *labelq is noise* **then**

**13 ** label*q* ← *C*;

**14 if** *labelq is undeﬁned* **then**

**15** label*q* ← *C*;

**16** Neighbors *N* ←NN-QUERY*(D, q, E)*;

**17 if** |*N*| ≥ *minPts* **then**

**18 ** *S* ← *S* ∪ *N*;

**19 return** *labeli (i* = 1*, . . . ,* |*D*|*)*

Several papers focus on reducing the computational cost of NN- QUERY, while others focus on ﬁnding an approximate solution to NN-QUERY with lower computational burden. Another approach for reducing the cost of DBSCAN is to reduce the number of pat- terns in the dataset, either by random sampling or by using clus- tering algorithms to generate samples composed of prototypes. The Rough-DBSCAN and the two proposals presented in this paper use the Leader algorithm to search for prototypes. Therefore, in the next subsection the Leader algorithm is described.

* 1. *Leader clustering algorithm*

The Leader [[24]](#_bookmark40) is an incremental clustering algorithm which performs a *single scan* on the dataset. Unlike k-means, the Leader does not implement any optimization step. Therefore, it has a much shorter execution time.

The canonical Leader algorithm ([Algorithm 2](#_bookmark6)) starts making the ﬁrst element *D*0 a leader and scans the remaining of the dataset

**Algorithm 2:** Leader.

**Data**: Dataset *D*, Threshold distance *τ*

**1** *L* ← *D*0 ;

**2** *F*0 ← *D*0 ;

**3 for** *each d* ∈ *D* \ {*D*0} **do**

**4** leader ← true;

**5 for** *each l* ∈ *L* **do**

**6 if** ||*l* − *d*|| ≤ *τ* **then**

**7** *Fl* ← *Fl* ∪ *d*;

**8** leader ← false;

**9 break**;

**10 if** *leader* **then**

**11** *L* ← *L* ∪ *d*;

**12** *Fd* ← *d*;

**13 return** *{L, F}*

*2.4. Rough-DBSCAN*

**Fig. 1.** Output example of the Leader algorithm, the highlighted red stars are the leaders and the followers are all the black dots inside the leader’s region. (For in- terpretation of the references to color in this ﬁgure legend, the reader is referred to the web version of this article.)

verifying whether the current element is below a certain distance *τ* of any of the existing leaders. If an element is not close to any leader, it becomes a leader. Otherwise, it will be a follower of the ﬁrst leader found within a distance less than *τ* . It is important to note that as *τ* → 0 all elements of the dataset will be leaders, *L* → *D*.

In Rough-DBSCAN, proposed by Viswanath and Babu [[23]](#_bookmark39), the Leader algorithm is used to generate a new dataset with the proto- types and their respective densities. These densities are calculated based on the number of elements in each cluster returned by the algorithm.

The Rough-DBSCAN combines the methods that have been pre- sented so far. The ﬁrst step is to generate a sample using the Leader clustering algorithm. The second step is to evaluate the density of each leader using the rough-cardinality method. This new dataset only consists of leaders who have an estimated den- sity above the DBSCAN threshold. Thus, the sample is composed only by the leaders with cardinality greater than *minPts*. The DB- SCAN algorithm runs on this sample and identiﬁes the clusters. The result of DBSCAN is a grouping of leaders. To construct the ﬁnal partition, each leader must be replaced by all his followers found during the execution of the Leader algorithm, and every follower will belong to the same cluster of the leader.

# Rough∗-DBSCAN and I-DBSCAN

It is easy to see that Leader is very sensitive to the order in which the data is presented to the algorithm. Elements at the be- ginning of the dataset have a higher chance of becoming leaders than the last ones. In addition, looking at lines 6–9 of [Algorithm 2](#_bookmark6), one can note that the ﬁrst leaders to be discovered will always have preference over others when deciding which leader the cur- rent element will follow ([Fig. 1](#_bookmark7)). If only the number of followers of each leader are considered to estimate their density, it will of- ten underestimate the density of some clusters due to this prior-

ity in deciding which leader the follower will be associated. There will almost always be elements within a distance less than *τ* from

the leader, but not following it. Therefore, calculating the den- sity from the output of the Leader algorithm alone is not a trivial task.

*2.3. Rough-cardinality*

Viswanath and Babu [[23]](#_bookmark39) proposed a way to estimate the den- sities of the leaders from the output of the Leader algorithm, called rough-cardinality. The cardinality of a leader can be deﬁned as the amount of elements existing within a given distance from the leader.

The rough-cardinality also uses the parameter *E* (the parame-

ter in DBSCAN used to determine whether or not a point is dense) to estimate the cardinality of each leader. The rough-cardinality is deﬁned for each leader as the *sum of the cardinality of all lead- ers* within a radius *E*. This procedure has a worst case complex- ity of *O(k*2 *)* where *k* represents the number of leaders found in the previous step. Although it has a quadratic complexity, normally *k* is much smaller than the number of patterns in the dataset. Even though the Leader algorithm guarantees that no leader will be closer than *τ* from another, note that if *τ* ≤ *E* there can still be leaders with a less than *E* distance from each other.

This section presents two new adaptations of the DBSCAN algo- rithm to perform clustering tasks in huge datasets. The two pro- posals are based on a modiﬁed version of the Leader algorithm, called Leader∗. Therefore, this section starts describing the Leader∗

and follows describing the Rough∗-DBSCAN and the I-DBSCAN.

* 1. *Leader*∗ *algorithm*

One of the major weaknesses of Rough-DBSCAN is the density estimation based on the result of the Leader algorithm. In order to overcome this weakness, an element may be associated with more than just one leader. With this modiﬁcation a more precise and straightforward density estimation can be performed.

The modiﬁed version of the Leader algorithm can be seen in [Algorithm 3](#_bookmark10). Even though it is necessary to do a second scan on

**Algorithm 3:** Leader∗.

**Data**: Dataset *D*, Threshold distance *τ* , Distance *E*

**1** *L* ← *D*0 ;

**2 for** *each d* ∈ *D* \ {*D*0} **do**

**3** leader ← true;

**4 for** *each l* ∈ *L* **do**

**5 if** ||*l* − *d*|| ≤ *τ* **then**

**6** leader ← false;

**7 break**;

**8 if** *leader* **then**

**9 ** *L* ← *L* ∪ *d*;

**10 for** *each d* ∈ *D* **do**

**11 for** *each l* ∈ *L* **do**

**12 if** ||*l* − *d*|| ≤ *E* **then**

There are two types of errors in this estimation. First, leaders **13**

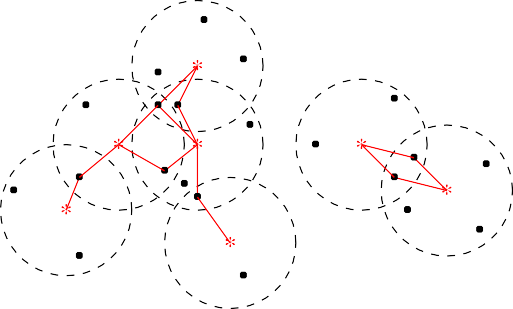
who are more than *E* distance yet have priority over the leader

*Fl* ← *Fl* ∪ *d*;

which the procedure is trying to estimate the density, this effect will decrease the density of this leader. Secondly, leaders who are a distance above *τ* and less than *E* with elements at a distance greater than *E* of the current leader increases the density estima- tion of the latter. The authors show that on average the two effects tend to cancel each other and this method provides a good estima- tion for the leader’s densities [[23]](#_bookmark39). While the results obtained by the authors were satisfactory, it is possible to improve the results by using a mechanism to correct the density estimation.

**14 return** *{L, F}*

the dataset (lines 10–13), in this modiﬁcation the preference ef- fect of the Leader algorithm is eliminated. The density is estimated based on the number of elements associated with each leader. [Fig. 2](#_bookmark11) illustrates the result of this modiﬁed version. The small in- crease in the computational cost leads to much better results due to the correct density estimation.



**Fig. 2.** Output example of the modiﬁed Leader algorithm, the highlighted red stars are the leaders and the followers are all the elements inside the leaderâs region (in- cluding intersections). The red lines connecting some elements to the leaders show followers who follow simultaneously more than one leader. (For interpretation of the references to color in this ﬁgure legend, the reader is referred to the web ver- sion of this article.)

* 1. *Rough*∗*-DBSCAN*

The Rough∗-DBSCAN is very similar to Rough-DBSCAN, the dif- ference is in the Leader algorithm itself. Instead of executing the canonical Leader algorithm and estimating the density of each leader through the rough-cardinality, the Rough∗-DBSCAN uses the modiﬁed version of the Leader to obtain the densities of the lead- ers to carry out the DBSCAN.

Given the DBSCAN parameters, *E* and *minPts*, and the *τ* pa-

rameter the Rough∗-DBSCAN performs a modiﬁed version of the Leader clustering algorithm, presented in this section, and only se- lects the leaders with more than *minPts* followers in a sphere of ra- dio *E* to compose the sample. All leaders with less than *minPts* fol- lowers are discarded. Since all elements in the sample have more than *minPts* elements within a *E* radius, they are above the den-

idea is to compose a sample of leaders and some elements belong- ing to more than one leader, i.e. elements on the intersection of the leaders’ spheres. For that reason it was called I-DBSCAN, where I stands for intersection.

Unlike Rough-DBSCAN and Rough∗-DBSCAN, the I-DBSCAN does not have the *τ* parameter. Therefore, it does not need to be tuned. This heuristic only needs parameters *E* and *minPts*, the parameters

required by the DBSCAN algorithm.

Due to the absence of the *τ* parameter, the I-DBSCAN algorithm runs with *τ* = *E,* and for that reason, can not execute using only the leaders as a sample. Since the leaders will be at a distance greater than or equal *E* from each other, if DBSCAN were to run in that sample with *minPts >* 1 all elements would be marked as noise. After all, there will be no other element at a distance less than *E*, or in case *minPts* = 1*,* each element would be assigned to a different cluster.

For ensuring the dense leader in the original dataset is also dense in the sample one must add more elements to the sam- ple than just the set of leaders. Adding the elements at the in- tersection of leaders is a good idea for two reasons. Firstly, ele- ments belonging to the intersections contribute to the density of more than one leader and, therefore, helps to keep the sample size small. Secondly, the elements at the intersections allow a path by

which NN-QUERY can reach one leader from another. These ele- ments are closer than *E* from the leaders which ensures that all leaders who are linked through these elements will belong to the

same cluster. The only exception happens when one of the two leaders is classiﬁed as noise by the DBSCAN. In this case the ele- ment at the intersection will be assigned to the cluster of the non- noisy leader.

The I-DBSCAN starts executing the modiﬁed version of the Leader clustering algorithm with *τ* = *E,* as can be seen in the ﬁrst line of the [Algorithm 4](#_bookmark12). After the execution of the Leader algo-

sity required by DBSCAN to not be marked as noise, so the *minPts*

parameter is from now on irrelevant and it is possible to assign *minPts* = 1. The next step of the algorithm is run the DBSCAN algo- rithm with parameters *E* and *minPts* = 1 on the sample. The ﬁnal solution is returned by taking every leader on the sample and re- placing by all the followers of that leader, without duplicates. Due to the intersections it is possible that the algorithm tries to add an element more than once. In this situation the element will be added in an arbitrarily chosen leader. The remaining elements are marked as noise.

It’s is important to note when *τ* → 0 the result of Rough∗-

DBSCAN approximates the result of DBSCAN in the whole dataset.

In this scenario, the output of the Leader algorithm will be com- posed of each element of the dataset *D* as a leader, forming |*D*| clusters without followers. The density of every leader will be es-

timated and input for the DBSCAN algorithm will be composed of all the dense elements of the dataset. The DBSCAN algorithm will group the leaders into clusters and every element assigned to the DBSCAN will be replaced by itself in the same cluster and the re- maining of the elements left over the sample by the density cri- teria are marked as noise. The *τ* parameter acts as an approxi- mation factor of the DBSCAN algorithm in the complete dataset. When *τ* = 0*,* the algorithm returns the same result as the DBSCAN, whereas when *τ* increases the results become further from that one obtained by DBSCAN in the complete dataset, but with a re- duction in the runtime.

* 1. *I-DBSCAN*

In addition to the Rough∗-DBSCAN, this work proposes a heuris- tic to draw a sample for the DBSCAN that also uses the modiﬁ- cation of the Leader algorithm, proposed in [Section 3.1](#_bookmark9). The main

**Algorithm 4:** I-DBSCAN.

**Data**: Dataset *D*, *E*, *minPts*

**1** {*L, F*} ← Leader\*(*D, E, E*) ; /\* Modified version \*/

**2** *S* ← *L*;

**3 for** *each l* ∈ *L* **do**

**4** *s* ← Find all followers of *l* in any intersection;

**5 if** |*Fl* | *> minPts* **then**

**6 if** |*s*| *> minPts* **then**

**7 ** *s* ← FFT-SAMPLING(*s, minPts*);

**8 else**

**9 ** *s* ← *s* ∪ SAMPLE(*Fl , minPts* − |*s*|);

**10** *S* ← *S* ∪ *s*;

**11 return** *S*;

rithm, the set of all leaders are inserted into the sample (line 2). The remainder of the sampling algorithm basically examines each leader (line 3) to add new elements to the sample. If a leader con- tains less than *minPts* followers, all the followers of that leader who follow more than one leader are added to the sample (line 10). Otherwise, if the leader has more than *minPts* followers (line 5), two scenarios should be considered. The ﬁrst happens when the leader has less than *minPts* elements at intersections. In this case, all elements in the intersection are added to the sample and also some elements chosen by a uniform random procedure SAMPLE (line 9). Those extra elements are just for ensuring that a leader who is dense in the original dataset remains dense in the sam- ple. These random elements are inserted only to prevent the leader being marked as noise by the DBSCAN algorithm. The second sce-

**Table 1**

Datasets and DBSCAN parameters.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset | #elements | #features | *E* | *minPts* |
| Abalone (Scale) | 4177 | 8 | 0.2 | 3 |
| Mushrooms | 8124 | 112 | 2.5 | 4 |
| Pendigits | 10,992 | 10 | 40 | 4 |
| Letter | 20,000 | 26 | 0.5 | 8 |
| Cadata | 20,640 | 8 | 200 | 8 |
| Shuttle | 58,000 | 7 | 0.03 | 20 |
| Sensorless (Scale) | 58,509 | 48 | 0.3 | 20 |
| SensIT (acoustic) | 98,528 | 50 | 0.5 | 5 |
| SensIT (seismic) | 98,528 | 50 | 0.4 | 5 |
| Skin nonskin | 245,057 | 3 | 60 | 10 |
| Poker | 1,025,010 | 10 | 4 | 10 |

nario occurs when the leader has more than *minPts* elements in the intersections. It is not needed to add all of them to ensure the density of the leader. For keeping the sample as small as possi- ble, it is only necessary to add *minPts* elements. The elements are chosen using the Farthest-First Traversal sampling (FFT-SAMPLING) algorithm (line 7). Farthest-First Traversal sampling initially selects an arbitrary element and then successively selects the farthest el- ement from the set of previously selected elements. This assures the selected elements are well spread out, probably covering more than one intersection.

After the sample is assembled, the next steps of the algorithm

are similar to the others methods. The DBSCAN algorithm runs in the sample, with the *minPts* and *E* parameters, and like Rough- DBSCAN and Rough∗-DBSCAN the ﬁnal partition is classiﬁed based on the leaders’ clusters. Every leader on the sample is replaced by all the followers found by the Leader clustering algorithm. Note that all non-leader elements added in the sample are discarded in this ﬁnal classiﬁcation, they are added just to assure the minimum density of the leaders and help DBSCAN ﬁnd the correct grouping for the leaders.

# Experiments

The comparison of methods was performed on eleven datasets ([Table 1](#_bookmark13)). Two of them, Pendigits and Letter, were chosen for be- ing present in the original work of Rough-DBSCAN. The others are public domain datasets, from different domains avaliable on the LIBSVM repository. They were chosen for verifying if the behav- ior of the algorithms remains the same in different scenarios. The experiments try to identify which method most closely matches the DBSCAN results and still signiﬁcantly reduces its running. The performance is measured by the Adjusted Rand Index (ARI) [[26]](#_bookmark42), a metric that measures the level of agreement between two cluster- ing results of a dataset. The Adjusted Rand Index has a maximum value of 1 indicating that the clusterings are exactly the same. In the experiments, the ARI was measured comparing the clusters ob- tained by each method, with the partitioning obtained by the DB- SCAN method running in the complete dataset.

The two largest datasets, Poker and Skin nonskin, were in- cluded to demonstrate the ability of these methods to run in larger datasets. For those datasets the DBSCAN was not executed due to the high computational cost. The original DBSCAN was inter- rupted for these datasets when the running time exceeded 24 h. For that reason in these datasets each method had its results mea- sured against the results obtained by the other two sampled meth- ods.

The parameters *τ* , *E* and *minPts* used on the experiments per-

formed on the Pendigits and Letter datasets have the same values used on the experiments performed in [[23]](#_bookmark39). For the other datasets, the parameters were chosen arbitrarily. [Table 1](#_bookmark13) shows the values

of the parameters *E* and *minPts* for each dataset in addition to the number of elements and features.

The *τ* parameter affects the sample size of the Rough-DBSCAN

and Rough∗-DBSCAN. For each dataset four values were chosen for *τ* to generate from small to large samples. Due to the lack of a random component in the methods, each one was performed only once for each parameter setting.

[Table 2](#_bookmark15) shows the experimental results for all datasets with exception of Poker and Skin nonskin datasets. The best ARI and time results are presented in bold. [Fig. 3](#_bookmark16) summarizes the results of [Table 2](#_bookmark15) showing the averaged ranking for ARI and running time

for all methods considering all datasets and each *τ* value setting.

The ﬁrst value on the x-axis summarizes the results for the high- est *τ* value for all datasets, the next value for the second largest *τ* value, and so on. When *τ* is large, yielding a small sample size for the Rough-DBSCAN and Rough∗-DBSCAN, one may see that the I-DBSCAN gets the best average ranking for the ARI and the Rough- DBSCAN is the fastest method. The sample sizes rises as *τ* dimin- ishes and the Rough∗-DBSCAN becomes the closest method to the DBSCAN results, going from the worst ranking to the best. How- ever, it is the slowest method in all cases. Although I-DBSCAN has a ﬁxed running time it becomes the fastest since the other two methods require larger computational time as a result of the increase of the sample size. It is important to note that Rough- DBSCAN presented the worst rankings for the ARI (except when *τ* was at the maximum) and it is always worse than the rank ob- tained by I-DBSCAN.

Once DBSCAN and I-DBSCAN do not have the *τ* parameter,

[Tables 2](#_bookmark15) and [3](#_bookmark17) display the same value of running time of these methods for each dataset.

* 1. *Running time*

The runtime for all datasets ([Tables 2](#_bookmark15) and [3](#_bookmark17)) shows the proxim- ity of the Rough-DBSCAN and the Rough∗-DBSCAN. Rough-DBSCAN has a small advantage in time when compared against Rough∗- DBSCAN in all datasets due to the modiﬁed version of the Leader algorithm, which requires two scans in the dataset. However, as it can be seen, this additional step in the Leader does not impact too much the overall execution time.

The results for the Abalone, Pendigits and Letter datasets

([Table 2](#_bookmark15)) show that the Rough-DBSCAN and Rough∗-DBSCAN are slower than DBSCAN itself when the *τ* value is small. However, for larger datasets, even the slowest method (Rough∗-DBSCAN) with the smallest *τ* parameter value runs faster than DBSCAN. These facts indicate that both methods have a relatively high computa- tional time, but they have a better scalability than the DBSCAN algorithm. Therefore, for larger datasets, all methods offer a sig- niﬁcant execution time reduction.

The I-DBSCAN had smaller overall execution times for all datasets. The only exception is seen when *τ* was close to *E*. These cases coincide exactly with the smallest sample size and with a

poor approximation in relation to the results obtained by the DB- SCAN. In these situations Rough-DBSCAN and Rough∗-DBSCAN al- gorithms achieved a better runtime performance.

The methods were run on larger datasets for testing their scal- ability. [Table 3](#_bookmark17) shows the results for Skin nonskin and Poker datasets. In the Skin nonskin dataset the patterns seemed very dense and the runtime of both Rough-DBSCAN and Rough∗- DBSCAN was low due to the small sample size required. Although fast, the procedure for selecting the elements at the intersections in the I-DBSCAN had a major impact on the overall execution time. The results for the Poker dataset repeats the same results pre- sented for the smaller datasets (as shown by [Fig. 3](#_bookmark16)).





**Fig. 3.** Ranking comparison between Rough-DBSCAN, Rough∗-DBSCAN, I-DBSCAN for all results presented in [Table 2](#_bookmark15). For each dataset the three methods were ranked based on their performances and average across the datasets were calculated for each method.

**Table 2**

Experimental results.

DBSCAN Rough-DBSCAN Rough∗-DBSCAN I-DBSCAN

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Dataset | *τ* | Time(s) |  | ARI | Time (s) |  | ARI | Time (s) |  | ARI | Time (s) |  |
| Abalone (Scale) | 0.15 | 2.0 |  | 0.89498 | **0.3** |  | 0.82760 | 0.4 |  | **0.98212** | **0.3** |  |
|  | 0.1 | 2.0 |  | **0.99542** | 0.8 |  | 0.97705 | 0.9 |  | 0.98212 | **0.3** |  |
|  | 0.075 | 2.0 |  | 0.99834 | 1.5 |  | **0.99878** | 1.7 |  | 0.98212 | **0.3** |  |
|  | 0.05 | 2.0 |  | 0.99953 | 2.7 |  | **1.0** | 2.8 |  | 0.98212 | **0.3** |  |
| Mushrooms | 2.25 | 97.8 |  | 0.90744 | 1.9 |  | 0.90723 | 4.3 |  | **0.91018** | **1.8** |  |
|  | 2 | 97.8 |  | 0.90744 | 2.1 |  | 0.90967 | 29.5 |  | **0.91018** | **1.8** |  |
|  | 1.75 | 97.8 |  | **0.91018** | 16.8 |  | 0.90967 | 29.5 |  | **0.91018** | **1.8** |  |
|  | 1.5 | 97.8 |  | **0.91018** | 16.6 |  | 0.90967 | 29.4 |  | **0.91018** | **1.8** |  |
| Pendigits | 30 | 25.6 |  | 0.90735 | 5.9 |  | 0.87531 | 8.2 |  | **0.97226** | **5.5** |  |
|  | 25 | 25.6 |  | 0.97434 | 12.6 |  | **0.98108** | 16.4 |  | 0.97226 | **5.5** |  |
|  | 20 | 25.6 |  | 0.97997 | 25.5 |  | **0.99382** | 30.0 |  | 0.97226 | **5.5** |  |
|  | 15 | 25.6 |  | 0.98396 | 43.8 |  | **0.99873** | 45.0 |  | 0.97226 | **5.5** |  |
| Letter | 0.4 | 83.2 |  | 0.48805 | **25.9** |  | 0.51722 | 35.9 |  | **0.78703** | 36.5 |  |
|  | 0.35 | 83.2 |  | 0.60939 | 47.6 |  | 0.74672 | 62.1 |  | **0.78703** | **36.5** |  |
|  | 0.3 | 83.2 |  | 0.65552 | 63.2 |  | **0.82271** | 77.5 |  | 0.78703 | **36.5** |  |
|  | 0.25 | 83.2 |  | 0.71589 | 104.2 |  | **0.92820** | 114.6 |  | 0.78703 | **36.5** |  |
| Cadata | 190 | 105.5 |  | 0.00176 | **2.6** |  | 0.00086 | 4.0 |  | **0.95008** | 5.3 |  |
|  | 170 | 105.5 |  | 0.32304 | **3.0** |  | 0.23303 | 4.8 |  | **0.95008** | 5.3 |  |
|  | 150 | 105.5 |  | 0.80000 | **3.8** |  | 0.75335 | 5.8 |  | **0.95008** | 5.3 |  |
|  | 120 | 105.5 |  | 0.88185 | 5.4 |  | 0.93695 | 8.0 |  | **0.95008** | **5.3** |  |
| Shuttle | 0.025 | 791.6 |  | 0.20430 | **4.2** |  | 0.17485 | 5.9 |  | **0.82221** | 8.5 |  |
|  | 0.02 | 791.6 |  | 0.75313 | **6.8** |  | 0.74034 | 9.2 |  | **0.82221** | 8.5 |  |
|  | 0.015 | 791.6 |  | 0.85545 | 9.3 |  | **0.85470** | 13.0 |  | 0.82221 | **8.5** |  |
|  | 0.01 | 791.6 |  | 0.99861 | 15.3 |  | **0.99992** | 21.4 |  | 0.82221 | **8.5** |  |
| Sensorless (Scale) | 0.2 | 7379.3 |  | 0.79940 | **32.6** |  | 0.81089 | 104.7 |  | **0.99891** | 34.0 |  |
|  | 0.175 | 7379.3 |  | 0.76622 | 55.4 |  | 0.77719 | 146.7 |  | **0.99891** | **34.0** |  |
|  | 0.15 | 7379.3 |  | 0.80119 | 107.2 |  | 0.81217 | 264.8 |  | **0.99891** | **34.0** |  |
|  | 0.125 | 7379.3 |  | 0.93949 | 208.5 |  | 0.95797 | 484.5 |  | **0.99891** | **34.0** |  |
| SensIT (acoustic) | 0.4 | 31640.9 |  | 0.77247 | 1265.0 |  | 0.86352 | 1775.5 |  | **0.93737** | **1033.4** |  |
|  | 0.35 | 31640.9 |  | 0.79622 | 1879.4 |  | **0.95174** | 2530.8 |  | 0.93737 | **1033.4** |  |
|  | 0.3 | 31640.9 |  | 0.79910 | 2734.2 |  | **0.98590** | 3461.8 |  | 0.93737 | **1033.4** |  |
|  | 0.25 | 31640.9 |  | 0.79992 | 3953.5 |  | **0.99487** | 5155.7 |  | 0.93737 | **1033.4** |  |
| SensIT (seismic) | 0.35 | 10626.4 |  | 0.70192 | **2405.2** |  | 0.56338 | 3296.8 |  | **0.96184** | 3507.2 |  |
|  | 0.3 | 10626.4 |  | 0.77549 | 4232.6 |  | 0.93424 | 4970.6 |  | **0.96184** | **3507.2** |  |
|  | 0.25 | 10626.4 |  | 0.79154 | 5788.6 |  | **0.98769** | 6986.2 |  | 0.96184 | **3507.2** |  |
|  | 0.2 | 10626.4 |  | 0.79323 | 7440.9 |  | **0.99821** | 7766.6 |  | 0.96184 | **3507.2** |  |

**Table 3**

Experimental results for the Skin nonskin and Poker data sets.

Rough-DBSCAN Rough∗-DBSCAN I-DBSCAN

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Dataset | *τ* | Time (s) | ARI (Rough∗-DBSCAN) | ARI (I-DBSCAN) |  | Time (s) | ARI (I-DBSCAN) | Time (s) |
| Skin nonskin | 28 | **2.5** | 0.98217 | 0.97781 |  | 5.8 | 0.99014 | 6.3 |
|  | 26 | **2.6** | 0.99450 | 0.99233 |  | 6.3 | 0.99015 | 6.3 |
|  | 24 | **3.0** | 0.99395 | 0.99179 |  | 7.3 | 0.99124 | 6.3 |
|  | 22 | **3.7** | 0.88711 | 0.88175 |  | 8.6 | 0.99124 | 6.3 |
| Poker | 3.8 | **1844.0** | 0.91414 | 0.98236 |  | 2371.0 | 0.98236 | 2340.9 |
|  | 3.7 | 2599.6 | 0.99309 | 0.99874 |  | 3112.5 | 0.99874 | **2340.9** |
|  | 3.6 | 3145.5 | 0.99866 | 0.99985 |  | 4088.8 | 0.99985 | **2340.9** |
|  | 3.5 | 3355.1 | 0.99866 | 0.99985 |  | 4163.3 | 0.99985 | **2340.9** |

* 1. *Adjusted Rand Index*

All the methods obtained a good level of agreement with the results of the DBSCAN, considering a suitable choice of the *τ* pa- rameter value. The I-DBSCAN does not need this extra parameter and still achieved a good level of approximation for all datasets. The worst results for the I-DBSCAN achieved an Adjusted Rand In- dex of 0.78 and 0.82 in the Letter and Shuttle datasets respectively ([Table 2](#_bookmark15)) and on the others datasets the ARI was always above 0.91. Except from Abalone, Pendigits, and Shuttle datasets, the I-DBSCAN obtained better results than Rough-DBSCAN for all values of *τ* with a shorter overall runtime.

The Rough-DBSCAN obtained the best results only in 3 cases. For most cases the Rough∗-DBSCAN outperforms the Rough-

DBSCAN. Indeed, the Rough∗-DBSCAN with the two lowest setting of *τ* shows the best Adjusted Rand Index for the majority of the

datasets.

To verify if the results have any statistical signiﬁcance, the Wilcoxon signed-rank test was applied in the results presented in [Table 2](#_bookmark15). As in [Fig. 3](#_bookmark16), the results were grouped into four groups considering the parameter *τ* in order to avoid the dependence be- tween the results of the same dataset. The ﬁrst group is composed of all results of the highest *τ* values, second group composed of all results of the second largest value, and so on. The I-DBSCAN presented the best results for the highest values of *τ* (0.0039 *p*- value for Rough-DBSCAN and Rough∗-DBSCAN) while the results between Rough-DBSCAN and Rough∗-DBSCAN had no statistical difference on average. In descending order of *τ* values, the next test showed that there are statistical differences only between the I-DBSCAN and Rough-DBSCAN (0.0273 *p*-value). For this scenario I-DBSCAN is still achieved the best performance (alongside with Rough∗-DBSCAN). In the next test, due to the increase of the sam- ple size, Rough-DBSCAN was able to match the I-DBSCAN results, so no method showed any statistical difference. For the ﬁnal test, with the lowest *τ* value, the I-DBSCAN and the Rough-DBSCAN still did not make a signiﬁcant difference while the Rough∗-DBSCAN stood out from the two with the best results against Rough- DBSCAN (0.0117 *p*-value) and with no statistical difference in re- lation to I-DBSCAN.

[Table 3](#_bookmark17) shows the results for the two largest datasets, however, since ARI is a symmetric function, some existing results displayed in the previous columns may be omitted. In both datasets the agreement level between the methods was above 0.99 with the ex-

ception of the results for the Skin nonskin dataset when *τ* = 22. In

this case, Rough-DBSCAN found a different result, while I-DBSCAN and Rough∗-DBSCAN presented a high level of agreement. The al- gorithms in fact allowed the execution in large datasets with a

high degree of concordance.

# Conclusion

This work proposed two methods for sampling large datasets in order to execute the DBSCAN algorithm. After obtaining the clus- tering result on the sample, the rest of the elements are quickly distributed in the clusters.

One of the methods presented is a variation of the Rough- DBSCAN, called, Rough∗-DBSCAN, which obtained a better approxi- mation for the DBSCAN than the original method with a marginally

higher computational cost. The other method proposed, I-DBSCAN, was the fastest method without requiring any further parameter to

be tuned. On average, it achieved approximately 92.5% agreement with the result obtained by DBSCAN.

Both Rough-DBSCAN and Rough∗-DBSCAN need the *τ* parame- ter and a wrong choice for the *τ* value greatly impacts the quality

of the results obtained by both methods. Until today there is no established method to determine a value for this parameter. This problem will be studied in future works.

# References

1. [M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: Kdd, 96, 1996,](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0001)

[pp. 226–231.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0001)

1. SIGKDD, SIGKDD test of time award, 2014.
2. [R. Xu, D. Wunsch, Survey of clustering algorithms, IEEE Trans. Neural Networks 16 (3) (2005) 645–678.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0002)
3. [V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, ACM Comput. Surv. (CSUR) 41 (3) (2009) 15.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0003)
4. [A. Kassambara, Practical Guide to Cluster Analysis in R: Unsupervised Machine Learning, 1, STHDA, 2017.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0004)
5. [R. Sibson, Slink: an optimally eﬃcient algorithm for the single-link cluster method, Comput. J. 16 (1) (1973) 30–34.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0005)
6. [J. MacQueen, et al., Some methods for classiﬁcation and analysis of multivari- ate observations, in: Proceedings of the ﬁfth Berkeley symposium on mathe- matical statistics and probability, 1, Oakland, CA, USA, 1967, pp. 281–297.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0006)
7. [A.K. Jain, Data clustering: 50 years beyond k-means, Pattern Recognit. Lett. 31](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0007)

[(8) (2010) 651–666.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0007)

1. [J. Gan, Y. Tao, Dbscan revisited: mis-claim, un-ﬁxability, and approximation, in: Proceedings of the 2015 ACM SIGMOD International Conference on Man- agement of Data, ACM, 2015, pp. 519–530.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0008)
2. [T. Ali, S. Asghar, N.A. Sajid, Critical analysis of dbscan variations, in: Infor- mation and Emerging Technologies (ICIET), 2010 International Conference on, IEEE, 2010, pp. 1–6.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0009)
3. [K. Khan, S.U. Rehman, K. Aziz, S. Fong, S. Sarasvady, Dbscan: past, present and future, in: Applications of Digital Information and Web Technolo- gies (ICADIWT), 2014 Fifth International Conference on the, IEEE, 2014,](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0010)

[pp. 232–238.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0010)

1. [N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The r∗-tree: an eﬃcient and robust access method for points and rectangles, in: Acm Sigmod Record, 19, ACM, 1990, pp. 322–331.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0011)
2. [P. Ciaccia, M. Patella, F. Rabitti, P. Zezula, Indexing metric spaces with m-tree, in: SEBD, 97, 1997, pp. 67–86.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0012)
3. [S. Berchtold, D.A. Keim, H.-P. Kriegel, The x-tree: an index structure for high-](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0013)

[-dimensional data, in: Proceedings of the 22th International Conference on Very Large Data Bases, in: VLDB ’96, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996, pp. 28–39.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0013)

1. [S. Mahran, K. Mahar, Using grid for accelerating density-based clustering, in: Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on, IEEE, 2008, pp. 35–40.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0014)
2. [E. Chávez, G. Navarro, R. Baeza-Yates, J.L. Marroquín, Searching in metric spaces, ACM Comput. Surv. (CSUR) 33 (3) (2001) 273–321.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0015)
3. [D. Arlia, M. Coppola, Experiments in parallel clustering with dbscan, in: Euro- pean Conference on Parallel Processing, Springer, 2001, pp. 326–331.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0016)
4. [Y. Chen, S. Tang, N. Bouguila, C. Wang, J. Du, H. Li, A fast clustering algorithm based on pruning unnecessary distance computations in dbscan for high-di- mensional data, Pattern Recognit. (2018).](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0017)
5. [Y.-P. Wu, J.-J. Guo, X.-J. Zhang, A linear dbscan algorithm based on lsh, in: Machine Learning and Cybernetics, 2007 International Conference on, 5, IEEE, 2007, pp. 2608–2614.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0018)
6. [M. Datar, N. Immorlica, P. Indyk, V.S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: Proceedings of the twentieth an- nual symposium on Computational geometry, ACM, 2004, pp. 253–262.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0019)
7. [X. Wang, H.J. Hamilton, Dbrs: a density-based spatial clustering method with random sampling, in: Paciﬁc-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2003, pp. 563–575.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0020)
8. [Y. El-Sonbaty, M.A. Ismail, M. Farouk, An eﬃcient density based clustering al- gorithm for large databases, in: Tools with Artiﬁcial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on, IEEE, 2004, pp. 673–677.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0021)
9. [P. Viswanath, V.S. Babu, Rough-dbscan: a fast hybrid density based clustering method for large data sets, Pattern Recognit. Lett. 30 (16) (2009) 1477–1488.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0022)
10. [J.A. Hartigan, Clustering Algorithms, 99th, John Wiley & Sons, Inc., New York, NY, USA, 1975.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0023)
11. [E. Schubert, J. Sander, M. Ester, H.P. Kriegel, X. Xu, Dbscan revisited, revisited: why and how you should (still) use dbscan, ACM Trans. Database Syst. (TODS) 42 (3) (2017) 19.](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0024)
12. [L. Hubert](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0025), [P. Arabie, Comparing partitions, J. Classif. 2 (1) (1985) 193–218](http://refhub.elsevier.com/S0167-8655(18)30920-6/sbref0025).