Detection of natural clusters via S-DBSCAN a Self-tuning version of DBSCAN

Frédéric Ros, Serge Guillaume, Rabia Riad, Mohamed El Hajji

Version of Record: https://www.sciencedirect.com/science/article/pii/S0950705122000946
Manuscript_aa3cbc5ad6b3d92a57b2f49bc857b6d8

Detection of natural clusters via *S-DBSCAN* a Self-tuning version of *DBSCAN*

Frédéric Ros[a,*], Serge Guillaume[b], Rabia Riad[c], Mohamed El Hajji[d]

[a] *Laboratory PRISME, Orléans university, France*
[b] *ITAP, Univ Montpellier, INRAE, Montpellier SupAgro, Montpellier, France*
[c] *ERMAM - FPO, Ibn Zohr university, Morocco*
[d] *IRF-SIC, Ibn Zohr university, Morocco*

Abstract

Density-based clustering algorithms have made a large impact on a wide range of application fields application. As more data are available with rising size and various internal organizations, non-parametric unsupervised procedures are becoming ever more important in understanding datasets. In this paper a new clustering algorithm *S-DBSCAN* [1] is proposed in the context of knowledge discovery. *S-DBSCAN* belongs to the connectivity-based family such as *DBSCAN* but with noticeable differences and advantages as working in a differential mode. It is formalized via a very simple hierarchical process that hybridizes distance, k-nearest and Density peaks concepts. It aims at partitioning existing data into clusters until no more clustering can be done. The information delivered allows the user to intuitively deduce different sets of natural partitions in clusters at different scales. *S-DBSCAN* scans the database in a ordered way by applying its algorithm core (*S-DBSCANCORE*) with judicious input parameters. Given a set of data patterns in some space, *S-DBSCANCORE* groups together data patterns that are closely packed together with respect to the differential density. Data patterns whose nearest neighbors have too different densities are

*Corresponding author
Email addresses: frederic.ros@univ-orleans.fr (Frédéric Ros), serge.guillaume@inrae.fr (Serge Guillaume), r.riad@uiz.ma.ac (Rabia Riad), m.elhajji@uiz.ac.ma (Mohamed El Hajji)

[1] A sample code is (will be when ready for publication) available at: http://frederic.rosresearch.free.fr/mydata/homepage/

detected and marked as borders while the others are not visited. *S-DBSCAN* embeds some intelligence that makes it self-tuning (almost fully automatic) and not dependent on a global density threshold as many existing algorithms. Tests were carried out using 2-dimensional benchmark datasets of various shapes and densities. They showed that *S-DBSCAN* was highly effective. It also proved efficient in high dimension space when natural clusters exist and much easier to use than competitive algorithms.

Keywords: Clustering, Natural cluster, Distance, Density, Neighbors

---

1. Introduction

Clustering [1] is probably the most powerful unsupervised tool to find a structure in a collection of unlabeled data. It can be formally defined as the process of organizing the data into clusters whose members are similar in some way while data in different clusters are dissimilar. Three basic notions of what a cluster is lead to three main types of algorithms. If a cluster is defined by its center and a basin of attraction then distance is the central concept. It is also possible to define a cluster as a dense area separated from another cluster by a sparsely populated zone; in this case, density is the key idea. Finally, a third definition is based on a set of connected points, in which case neighborhood is of prime concern.

Many methods were developed in the past and the topic has been continously investigated [2, 3]. When the data are easy to cluster, meaning that the groups are well separated, most of the existing algorithms are likely to yield a good result, but clustering algorithms have to deal with more complex situations such as different types of attributes, various shapes and densities, and must include outlier and noise management. In addition to the recurrent challenges, the expectations for new proposals can be grouped in two categories. The first one concerns the dimension and volume of 21^{st} century databases. It involves the curse of dimensionality and algorithm scalability. This issue is not novel [4, 5] but more challenging today [6, 7] in the era of big data. The second

one is discovering natural clusters without any a priori information, which is still a challenging problem in cluster analysis. There is firstly a related issue concerning the cluster definition itself. The task of clustering is inherently prone to subjectivity, the optimal solution can be extremely costly to discover and sometimes even unreachable or nonexistent. Moreover, although conceptually the cluster notions are well-shared by the community, clustering algorithms nevertheless differ in how these notions are interpreted and formalized. Some algorithms are based on interpretations that are too restrictive to deal with real world situations or are too specific. Most of clustering algorithms are driven by the number of clusters as input. These approaches are however not very appropriate for a discovery process where the number of clusters is unknown. With them, the discovery is classically handled via internal indexes that can help while lacking in genericity and precision.

The real challenge for new algorithms is to be sufficiently self-tuning and simple. They also have to be adaptive for the result to be acceptable whatever the input parameters, or put differently, for the same set of parameters to handle various datasets. In that sense, the input parameters only differentiate between acceptable solutions and allow the user to select a more or less detailed representation of the data. This question of scale is essential. Several natural partitions can exist and need to be discovered.

In the private sector, pioneer *kmeans* and *DBSCAN* are rather satisfying in many cases, understandable by the investigator and especially easy to use. Since *DBSCAN* relies on a global density threshold to find core (high-density) points, it has issues in detecting clusters with varied densities. Different from *DBSCAN* using a Euclidean distance function, Shared Nearest Neighbours (*SNN*) [8] clustering uses SNN similarity instead of the distance measure in *DBSCAN*. The *SNN* similarity of any two points is the number of neighbours they share in their lists of nearest neighbours. Because points usually have lower *SNN* similarity in transition regions between different clusters, *SNN* can discover clusters with relatively uniform regions of different density values. *SNN* uses the same neighbourhood density estimator and the same cluster definition as *DBSCAN*. There

are many variants of these algorithms which have been proposed to overcome this weakness such as *Recon-DBSCAN* [9], [10], [11], *Rnn-DBSCAN* [12] and *SNN-radius* [13].

There are however not yet popular in the private sector as there is not a clear perception of the added value in terms of both efficiency and simplicity. Other variants also focus on the computational aspect and scalability addressing the big data challenge. They are based on approximate [14] or exact [15, 16] *DBSCAN* clustering but they do not address the discovery challenge presented above.

Our proposed algorithm "a Self-tuning version of *DBSCAN*" (*S-DBSCAN*) deals with the discovery process and aims at facing the multi-challenge of simplicity and efficiency while not addressing the big data challenge. It can however be combined with so called "sampling algorithms" to handle larger databases.

Its core is conceptually close to *DBSCAN* type algorithms while it has properties similar to those of partitioning and hierarchical algorithms. As in a divisive clustering, it keeps dividing existing data into clusters using increasing radii until no more clustering can be done. *S-DBSCAN* assumes that a cluster is characterized by the distribution of its neighboring patterns: quite homogeneous in the core of the cluster, with a possible decreasing level of homogeneity when moving away from the core. This definition covering many cluster issues is formalized in *S-DBSCAN* that differs from *DBSCAN* in its process. *DBSCAN* is based on driving parameters in absolute terms (Npts, ε).

These parameters are not easy to tune and *DBSCAN* encounters problems with density variation (cf. Fig.1). Our algorithm runs in a differential mode, which allows it to better manage density variation. It only needs one volume parameter.

The main contribution of this paper can be summarized as the proposal of a simple yet effective method *S-DBSCAN*.

The rest of the paper is organized as follows: Section 2 reviews some algorithms that are recent or close to our work. Section 3 recalls some definitions and notations. Section 4 presents the algorithm and includes all the pseudo

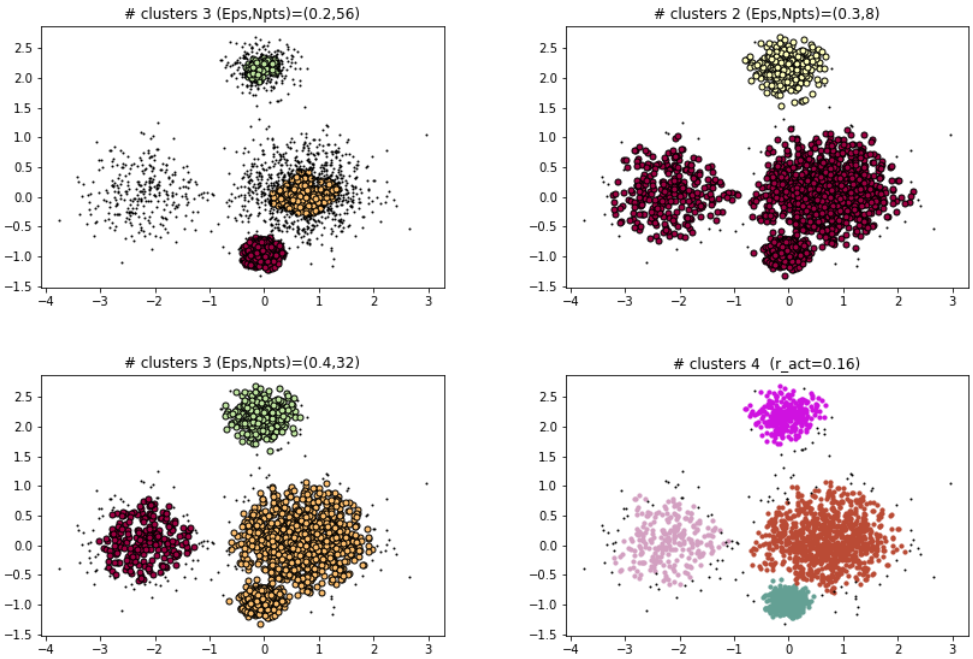

Figure 1: A simple data set with varied cluster densities in 2D space where *DBSCAN* encounters problems to discover a partition in 4 clusters whereas *S-DBSCAN* can do it (bottom-right Figure) without any tuning.

codes. Numerical experiments are carried out in Section 7. The final remarks and open perspectives are stated in Section 8.

2. Related work

The topic of clustering is mature and has been widely studied while still remaining challenging today. One of the recurrent challenges is related to the search of solutions for managing cluster complexity, which has been the subject of recent developments: non-linear distances with the kernel k-means, neural-networks, Bregman distances [17] or graph-based algorithms [18], hierarchical representation with agglomerative algorithms, based on density with *DBSCAN* [19], *Recon-DBSCAN* [9] and *DENCLUE* [20], the mean shift algorithm, *SC-DOT* [21], *Munec* [3] or *Kdmutual* [22].

Among recent algorithms the density peaks clustering algorithm, *DP*, was proposed in 2014 [23] and has become popular. It is based on the idea that cluster centers are characterized by a higher density than their neighbors and by a large distance from items with a higher density. The pioneering algorithm however suffered from some drawbacks due to its simplistic partition strategy: the cutoff distance is not really adapted to the density variability and once a high density point is mishandled its neighbors with lower density are more likely to be misclassified. In addition, when the data sizes of large clusters are much greater than those of small clusters, the information of small clusters is easily overwhelmed, resulting in subjectivity in determining the number of clusters. Improvements were recently proposed [21, 24, 2, 25, 26, 27, 12]. In [28], the threshold distance d_c is now automatically set using the potential entropy of the data field from the original dataset. The local density is estimated using a Gaussian function instead of the classical nearest neighbor count.

In [25], the Comparative Density Peaks algorithm (*Comp. Dp*) consists in considering the parameter $\theta_i = \delta_i - \tau_i$ instead of δ_i, τ_i being the distance between the point i and its nearest neighbor of lower density. θ embodies the relative magnitude of δ by comparing with τ and thus helps to identify the potential cluster centers. The automatic selection keeps points with the largest product distance by density.

In the *SCDOT* algorithm [21] cluster centers are also assumed to be density peaks that have a relatively large distance from higher density peaks. Local density and distance are estimated in the same way as in *DP*. A neighboring graph is constructed, as in *Chameleon* clustering [29], but with an additional constraint to yield a tree. A node is connected to only one other node, *i.e.* its nearest neighbor of higher density. The edge valuation is the same as in [23]. Cluster centers are recognized as points for which the edge value is larger than the typical nearest neighbor distance.

The *HierOpt* algorithm was detailed in [30]. Its basis is a hierarchical clustering algorithm using the single linkage criterion. Two improvements were proposed to deal with noise. First, the single linkage criterion takes into ac-

count the local density to ensure that the distance involves core points of each group. Second, the hierarchical algorithm forbids the merging of representative clusters, higher than a minimum size, once identified.

The *KdMutual* algorithm [22] is based on the assumption that working with cluster cores rather than considering frontiers makes the clustering process easier. It combines the best characteristics of density peaks and connectivity-based approaches. In *KdMutual* potential core clusters are first identified to allow clusters to be selected using a specific ranking criterion close to [23].

The *DBSCAN* algorithm is very popular without requiring the cluster number as input. It does not cope with varying density clusters subject of recent improvements [9, 12, 31]. For example, while *DBSCAN* defines reachable points using two parameters, *Recon-DBSCAN* [9] considers two radii, ϵ and θ with $\theta \geq \epsilon$. The reachability is based on the density ratio $N_{pts}(\epsilon)/N_{pts}(\theta)$ compared to the τ threshold. In the *Rnn-DBSCAN* algorithm [12], only one parameter should be specified which presents a great advantage. The algorithm adopts the same principle as *DBSCAN* to define the reachability of points in a data set but based on a reverse k nearest neighbors model. Its core idea is based on the union of k-nearest neighbor and reverse nearest neighbor to expand the cluster. Conceptually, with *Rnn-DBSCAN* a cluster of non-core objects with different densities cannot however be well-managed. The process of cluster expansion requires heavy memory that makes it computationally inefficient.

The Shared Nearest Neighbor algorithm, *SNN* [8], as well as its variants [13], is a density based clustering algorithm working similarly to *DBSCAN* albeit less popular. The main difference is that the volume is not defined by the radius but is induced by the nearest neighbors. The volume can be optionally limited by a radius. It encounters the same issue as *DBSCAN* as it relies on a global density threshold to find core (high-density) points.

In [32], the time complexity of *SNN* is reduced to $O(n \log(n))$ instead of $O(n^2)$ by using a kd-tree implementation.

In the *Mdca* algorithm [31], clusters of arbitrary shape can be easily located based on the selection of initial data objects. However, it needs to determine

the density threshold before clustering. For this reason, it is not suitable when the density of data sets varies largely or the overall density is basically the same.

The *Munec* algorithm [3] is based on an iterative process that merges mutual nearest neighbors. It does not require the number of clusters as input. The first merging steps are only controlled by the number of sub-clusters, to yield a skeleton of the data structure. Then, two distinct stages are proposed and three heuristic conditions help to discriminate between more nuanced situations and provide the final partition. The algorithm is driven by a single parameter that controls the level of density differentiation. The heuristics behind the algorithms remain complex to understand for the investigator; they also appear to be more appropriate to work in low dimensional space.

Several other density-based methods derived from *DBSCAN* have been recently proposed to deal with large scale data [33, 32, 16, 15, 34, 6, 14] or accelerate the processing. They do not address our challenge directly but are worth to be cited as complementary. Among them, the *Mr-DBSCAN* [34] that can achieve an ideal load balance in a severely skewed data environment. The latter was extended to *Isb-DBSCAN* [6] by focusing on clustering non-core objects, which is undetermined when two core objects are equidistant from a non-core object. The *Block-DBSCAN* [14] is a recent algorithm that consists in an approximate and grid-based clustering approach. Its complexity is $O(n \log(n)$. Two versions L_2 and L_∞ for relatively high and high dimensional data respectively were proposed. This idea presents two strategies using a computation reduction method. The first strategy uses a $\frac{\epsilon}{2}$-norm ball to detect the inner core blocks in which all points are the core points. The second strategy is a fast approximate algorithm that predicts whether two inner core blocks are density-reachable. Along with a cover tree to speed up density computations for unvisited points. A benefit of this technique is that it can cluster high-dimensional data relatively fast with high efficiency. At the opposite, even smart it remains based on the concept of approximate grid approaches that are their own shortcomings.

A few algorithms such as the *Block-DBSCAN* [14] aim at improving the run-time performance of *DBSCAN*. However, they suggest solutions based on

approximate clustering. Others such as μ-*DBSCAN* [16] or *G-DBSCAN* [15] produce exact *DBSCAN* clustering under the general idea to accelerate neighbor searching. The μ-*DBSCAN* algorithm is also amenable to parallelization. The idea behind this algorithm consists in reducing the number of neighborhood queries significantly. The algorithm first scans the data points, forms micro-clusters, builds a hierarchical structure known as Rtree, and then processes the micro-clusters to get preliminary clusters. In synthesis, it novelty lies in smart identification of core points and it allows to reduce the average time complexity of *DBSCAN* to $\mathrm{O}(n\log(m) + n\log(r))$, where n, m and r denote number of data points, micro-clusters and average number of points in one micro-cluster respectively.

In synthesis, many algorithms have been proposed to face the clustering challenges. Several are working without requiring the number of clusters as input and are more appropriate for a discovery process. The most popular such as *DBSCAN*, *SNN*, *Chameleon*, *DP* are based on global settings and cannot cover all the varied cluster situations and more constraining are not easily tunable. This shortcoming has boosted the generation of improved and more sophisticated variants from several years.

The latter have afforded interesting research findings (some of them are more difficult to tune) and have not turn into standards yet. Some others are focusing on the acceleration of known algorithms such as *DBSCAN* via approximate or exact clustering approaches. They do not however focus on the discovery improvement.

3. Definitions and Notation

3.1. Clustering problem

The clustering problem can be formally defined as follows: Given X, the clustering of X is the partitioning of X into C clusters $\{C_1, C_2, \ldots, C_C\}$ satisfying the following conditions: each pattern should be assigned to a cluster, *i.e.* $\cup_{l=1}^{C} C_l = X$, each cluster has at least one pattern assigned to it, *i.e.* $C_p \neq \emptyset$,

$p = 1, \ldots, C$ and for hard clustering, each pattern is assigned to one and only one cluster, *i.e.* $C_p \cap C_q = \emptyset$ for $p \neq q$.

3.2. Neighbors and neighboring

The neighborhood concept has been used in many clustering techniques. There are two main ways to define a neighborhood. The first one counts items that fall within a space, usually a hypersphere of radius r (influence zone of x) centered on the point x defined as $H_x(r)$.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n items (or patterns) in the p-dimensional feature space. The neighbors of x are the items located at a distance less than r:

$$N_r(x) = \{x_i \mid ||x_{(i)} - x|| < r\} \tag{1}$$

$$y \in H_r(x) \Leftrightarrow y \in N_r(x) \tag{2}$$

$H_r(x)$ defines the hypersphere centered in x and having a radius r.

The second one is the set of nearest neighbors. Let $\{x_{(1)}, x_{(2)}, \ldots, x_{(n-1)}\}$ be the permutation of the elements of $X \setminus \{x\}$, such that:

$$||x_{(1)} - x|| \leq ||x_{(2)} - x|| \leq \ldots ||x_{(n-1)} - x|| \tag{3}$$

The k-nearest neighbors of the x item are the set defined as:

$$N^k(x) = \{x_{(1)}, x_{(2)}, \ldots, x_{k)}\} \tag{4}$$

then, one can define the distance between x and its nearest neighbor number k as:

$$d^k(x) = d(x, x_{(k)}) \tag{5}$$

where d is the selected distance (Euclidean...).

The $n \times (k+1)$ nearest-neighbor matrix (NN) is set up as follows: the first entry in each row indicates the pattern under consideration, the second entry indicates the first nearest neighbour, the third entry indicates the second

nearest neighbour, and so on until the $(k+1)^{th}$ entry indicates the k^{th} nearest neighbour.

The $n \times n$ similarity matrix S is defined so that entry S_{ij} is the distance between pattern i and j.

3.3. Specific definitions for S-Dbscan

The similarity of densities between two data patterns x_1 and x_2 is expressed as:

$$Sim(x_1, x_2) = \frac{m}{M} \tag{6}$$

where $m = \min(dens_l(x_1), max(dens_l(x_2)$, $M = \max(dens_l(x_1), max(dens_l(x_2)$ and $dens_l(x))$ is the local density (section 5.2) at x. Let k be the considered nearest neighbors and r a hypersphere radius, Ω the corresponding neighborhood. One can define the local similarity maximum $Max_s\binom{k}{r}(x)$ and the average $\mu_s\binom{k}{r}(x)$ as follows:

$$Max_s\binom{k}{r}(x) = \max_{z \in \Omega}(Sim(x, z)) \tag{7}$$

$$\mu_s\binom{k}{r}(x) = \frac{\sum_{l=1 \ to \ k'} Sim(x, N^l(x))}{k'} \tag{8}$$

where k' ($k' \leq k$) is the number of nearest neighbors among k present in $H_r(x)$. When r and k are identified, the notation is abbreviated by $Max_s(x)$ and $\mu_s(x)$. The minimum distance between a pattern x and its nearest neighbors having a higher density than x is defined as:

$$d_{min}(x) = \underset{y \in N^k(x)}{argmin}\{d(x, y) \ / dens(x) \leq dens(y)\} \tag{9}$$

The average k nearest distance for the set X is defined as:

$$d_\mu[k] = \frac{\sum_{x \in X} d(x, N^k(x))}{n} \tag{10}$$

4. The proposed algorithm

4.1. Basic idea:

Our algorithm is articulated between the *S-DBSCAN* algorithm and its core algorithm named *S-DBSCANCORE*. *S-DBSCAN* consists in searching for the connectable elements using its core algorithm *S-DBSCANCORE* by scanning the patterns on the basis of an increasing series of influence zones $r_{inf}, \ldots r_{sup}$ that are automatically generated. The underlying idea is similar that the one of *DBSCAN* as *S-DBSCANCORE* groups together data patterns that are closely packed. With *S-DBSCAN* the connectivity criteria are based on the differential density and the scan is driven via selected entries (seeds). The seeds are determined by classifying (ordering) the patterns according to the principle of searching for density peaks proposed in [23]. Thus, the first entry point for *S-DBSCANCORE* is the 1^{st} seed, then the second is the 2^{nd} seed available (not already connected) until the entire database has been scanned. *S-DBSCAN* includes the mechanisms to provide all the required parameters for launching *S-DBSCANCORE* that is based on the following concept of connectivity:
Let A be a pattern of the base. A is connectable if there is a pattern B such that:

1. B is one of the k nearest neighbors of A: $B \in N^k(A)$
2. B is in the zone of influence $H_A(r)$(formalized by a hypersphere of radius r) of A: $B \in N_r(A)$
3. B must have a density comparable to that of A, measured by $max_s(A)$ and compared with the threshold Sim_{max}

Conditions 1 and 2 are basic neighborhood rules as condition (3) aims at prohibiting connections as soon as a discontinuity in density is detected. The algorithm is driven by three inputs (k, $H_A(r)$ and Sim_{max}) and that are automatically processed in *S-DBSCAN* and not tuned by the investigator. The idea widens the possibilities of connectivity as the zone of influence grows while remaining within the limit of the k nearest neighbors. The value of k corresponds

to the minimum nearest neighbors for which the patterns of the same natural cluster are mutually reachable. This value is central as it serves for estimating the local density (see section 5). In the case where clusters are separable, the patterns belonging to the same natural cluster are then mainly connectable. For a given r, there are then k' ($k' \leq k$) nearest neighbors that fall within the corresponding hypersphere.

4.2. Detailled description

The *S-DBSCAN* algorithm is shown in Algorithm 1. The pre-processing operations of *S-DBSCAN* concern the calculation of the features denoted I_p needed to drive the process: S, NN, k that will be used in the whole process, the determination of the local density $dens_l$ of all the patterns and Sim_{max}. These algorithms are detailed in section 5.

The input parameters for the user are the data set X, g_r. g_r serves to define the granularity of the scans for *S-DBSCAN* (line 8) in the interest range of distances. Then, the successive radii can be automatically calculated (line 11) and incremented. Sim_{max} is the main parameter that drives the algorithm by defining the level of continuity between pattern densities to be connectable. It is set up automatically (Algorithm 8) but it can be tuned by the user to modify the severity level.

S-DBSCAN includes internal generic parameters that are fixed. $SizeMin$ is the threshold above which a group can be considered as a cluster. The default value is set at $n/100$ that allows skipping outliers and noisy patterns while keeping small clusters, ε stands for the partition stagnation between two iterations (connex components without and with border patterns). Min_w is the threshold above which the found partition is considered as enough representative to be selected. The default value is set at $n/2$ meaning that at least half of patterns have to be included in the discovered partition to be qualified.

The core of the algorithm is concentrated in the central repeat loop (lines 8-20) and includes an internal loop (lines 10-13). At the beginning of the internal loop, all patterns are available and can be visited (line 9). The entry point for

Algorithm 1 *S-Dbscan* Clustering algorithm

1: Input: X, g_r {granularity}
2: Process I_p: [S, NN, k (Alg. 5), $dens_l$ (Alg. 7), Sim_{max} (Alg. 8)].
3: $\Delta_r = (d_\mu[k] - d_\mu[1])/gr$ {Equation 10}
4: $t = 0$, $r_{act} = d_\mu[1]$, $Min_w = 0.5$, $SizeMin = n/100$, $\varepsilon = 0.01$
5: **for** ($i = 1$ to n) **do**
6: $\delta[i] = dens_l[i] \times d_{min}[i]$ {$\delta[i]$: peak level for i, (Equation 9) }.
7: **end for**
8: **repeat**
9: $C_{all}[t] = \emptyset$, $Id = 0$, Visited[1..n]=false
10: **while** (seed=EntryPoint(δ, Visited) == true) **do**
11: $C_{Id} = S\text{-}DBSCANCORE(seed, r_{act}, Visited, Ip)$
12: $C_{all}[t] = C_{all}[t] \cup C_{Id}$, $Id = Id + 1$
13: **end while**
14: $C_{all[t]} = \{ \bigcup_{i \leq Id} C_i \mid |C_i| \geq SizeMin\}$, Id = $|C_{all[t]}|$
15: $N_{it}[t] = Id$, $R_{it}[t] = \dfrac{\sum_{i \leq Id} |C_i|}{|X|}$
16: **if** ($R_{it}[t] \geq Min_w$ & $N_{it}[t] \geq 1$) **then**
17: GrowingCore($C_{all[t]}$, Ip) {algorithm 9}
18: **end if**
19: $t = t + 1$, $r_{act} = r_{act} + \Delta_r$, $\Delta_c = R_{it}[t]$ - $R_{it}[t-1]$
20: **until** ($t > g_r$) & ($\Delta_c \leq \varepsilon$ || $N_{it}[t] == 1$)

S-DBSCANCORE is determined (line 10) following the principle of [23]. At each iteration, the entry point is the pattern having the best potential to be a density peak while being not visited or not marked as a border (cf. algorithm 2). From this entry point, all the patterns that are connectable are put in the same object using the core algorithm *S-DBSCANCORE*. All the connected patterns are marked when visited as well as the patterns that were visited but rejected (border patterns) as not belonging to the current object.

This subset of patterns is considered as a reliable connex component if it

represents a minimum number of patterns (line 14). In this case, the number of connex components is increased as well as the total number of patterns belonging to a connex component. The internal loop ends when there are no more potential seeds (line 10). One obtains both the number of clusters and the representativity (lines 14-15).

Algorithm 2 *EntryPoint* algorithm

1: Input: Visited, δ
2: Output: TRUE/FALSE, seed
3: $S = \{ \underset{x_i \in X}{\cup} | \ Visited[i] = false\}$
4: **if** $(S == \emptyset)$ **then**
5: return FALSE
6: **end if**
7: seed = $\underset{x_i \in S}{argmax}\{\delta[i]\}$ {the best not visited peak pattern}
8: return TRUE

$N_{it}[t]$ is the number of discovered clusters at time t while $R_{it}[t]$ expresses the consistency of a discovered partition (line 15). A minimum of consistency is needed to validate the partition. The case where $N_{it}[t] = 1$ may appear in the case of strong overlapping meaning the non presence of natural clusters.

By increasing the radii (line 19), the general tendency is to have the number of connex components increasing (groups having the minimum sizes are generated) and then decreasing (the partitions are formed with groups of larger sizes). The idea is then to stop the process in the decreasing phase when there is no point in testing a larger radius again. The algorithm stops after a minimum of g_r iterations (line 20) and when there is either one partition or a tiny evolution of the cumulative size of the connex components between two iterations (line 20).

A post processing stage is done under representative conditions (lines 16-17) (cf. algorithm 9) that consists in developing the discovered clusters using a neighbourhood mechanism. This algorithm is not central for the discovery topic but allows to refine the discovered clusters while avoiding the selection of

outlier data patterns.

4.3. S-DBSCANCORE algorithm

The key role of *S-DBSCANCORE* is to group together data patterns in subsets that are closely packed in terms of density. Each subset of patterns forms a connex component that represents a cluster. For this, it is necessary to examine if a candidate pattern can be connectable to the current group.

It is conceptually very close to *DBSCAN* while it is driven by a different strategy.

S-DBSCANCORE algorithm is shown in algorithm 3. It is recursive and driven by the Ip parameters given by *S-DBSCAN*. At the first call, the entry point is the available seed given in [23] and the cluster is empty. At the other calls, the entry point is a neighbor of a pattern already selected in this cluster. To be examined, each candidate pattern has to be not already visited (line 3). The visited patterns can belong to other clusters or marked as a border. In this case, the evaluation is done via the algorithm IsConnex (Algorithm. 4). Given a candidate A, its influence hypersphere $H_A(r_{act})$, and the similarities with all its neighbors included in $H_A(r_{act})$ (lines 5-6) are calculated (line 7). The maximum (line 9) and average (lines 8, 13) of similarities are determined and compared with Sim_{max}. If the condition is fulfilled A is connectable (line 13) then selected. If a pattern A is selected, it is added to the current cluster (line 10).

Then, *S-DBSCANCORE* is applied to all its neighbors in the influence region materialized by r_{act} to decide whether they are in the same component or not (lines 12). If A is rejected, its neighbors are simply not inspected. At the end of the algorithm, none of the visited patterns are available anymore and the list of connected patterns (from the entry point) is returned (line 8) as a novel cluster C. There are the selected and border patterns (visited but not selected).

4.4. Illustrative example and behavior

This dataset comprises 2000 patterns in $2d$ that were centered/normalized in each dimension. It includes 5 clusters with a small overlap and noisy patterns

Algorithm 3 *S-DBSCANCORE* algorithm

Input: Visited, Cand, C(cluster), r_{act}, I_p
Ouput: C
if (Visited[Cand] == TRUE) **then**
    return C;
end if
Visited[Cand] = TRUE
if (IsConnex(Cand, r_{act}, I_p) == FALSE) **then**
    return C {Alg. *IsConnex*}
end if
$C = C \cup$(Cand) {Candidate is included in C}
for ($i = 1$ to k') **do**
    C = *S-DBSCANCORE*(Visited, NN[Cand][i], C, r_{act}, I_p) {k'is the number of nearest neighbors in r_{act} with k'$\leq$ k}
end for

Algorithm 4 IsConnex algorithm

Input: Cand, r_{act} (radius), I_p
Output: TRUE/FALSE
$Max_s = 0$
for ($i = 1$ to k) **do**
    Neigh = NN[Cand][i], $d_{neigh} = d^{Cand}(Neighbor)$
    if ($d_{neigh} \leq r_{act}$) **then**
        $r = \dfrac{min(dens_l[Cand], dens_l[Neigh])}{max(dens_l[Cand], dens_l[Neigh])}$
        $Max_s = \max(Max_s, r)$
    end if
end for
if ($Max_s \geq Sim_{max}$) **then**
    return TRUE
else
    return FALSE
end if

(20%) randomly distributed. Their sizes are different and their shapes are rather compact for 3 of them and rather elongated for 2. Two of them are denser than the others. To run *S-DBSCAN*, only the granularity g_r was set up to its default value (10).

Fig. 2 illustrates the local densities (10 and 20 strata from left) of the patterns where the color depicts the relative local density between patterns: in red the densest (10%), in orange (between 10% and 20%) and in purple (from the highest to the lowest) the others. The relative densities are similar. The potential peak density can be seen in Fig. 3. The highest values of δ are in the patterns in red and in the large circle; then in yellow, grey, blue and the lowest values in black. The first radius $r_{act} = 0.049$ and $\Delta_r = 0.000705$.

S-DBSCAN needs 12 iterations to finish the process; the stopping radius was $r_{act} = 0.15$. In Fig. 4 from top to bottom and left to right, the connex components found by the algorithm can be seen. The growing procedure (*cf.* Algorithm 9 has not been activated to see the cluster cores. The number of connex components of size more than $n/100$ first increases and then decreases. The first two densest clusters are detected first and then the others (Fig. 4). For the latter, a higher radius is required to reach the pattern neighbors. Fig. 5 illustrates the process. The x axis presents the different radii (at the bottom) and the number of small subsets (at the top) considered as noise or outliers. At the beginning there are a lot of small subsets and this number decreases with the increase in radii. The y axis depicts the number of connex components (or clusters).

The result is reliable only from iteration 5 and beyond as the number of patterns grouping all the connex components is more than 50%. The vertical line in green shows this frontier. There is a plateau for 5 connex components that highlights the presence of 5 natural clusters in the database without any tuning. Partitions in 3 and 6 clusters are also possible but they present less stability. When the radius increases, *S-DBSCAN* tries to catch neighbors that respect the connectivity rule. As there is only one iteration to go from 5 to 3 and 3 to 1 cluster, it is likely that the clusters are closed to each other. The

cluster sizes from iterations 5 to 12 are given in Table 1. At each iteration (It), one can see the number of connex components (#C). The %" represent patterns without (%−") and with (%+") border patterns and the sizes of the discovered clusters. A border pattern is a pattern that has been reached via the recursive algorithm, but from which it is not possible to continue the process.

Fig. 5 shows the algorithm behavior when one threshold is automatic while the other varies in logical ranges. The results are comparable producing the same major plateau.

Table 1: cluster number

It	#C	%−	%+	$\|C1\|$	$\|C2\|$	$\|C3\|$	$\|C4\|$	$\|C5\|$	$\|C6\|$
5	5	54.6	55.4	483	480	48	36	32	X
6	6	64.5	65.9	504	500	110	87	54	30
7	5	69.8	71.9	550	509	155	95	93	X
8	5	73.4	75.78	561	515	172	122	107	X
9	5	76.0	78.91	568	521	187	146	116	X
10	5	78.5	81.68	573	524	215	156	124	X
11	3	79.9	83.22	796	690	136	X	X	X
12	1	80.8	84.24	1642	X	X	X	X	X

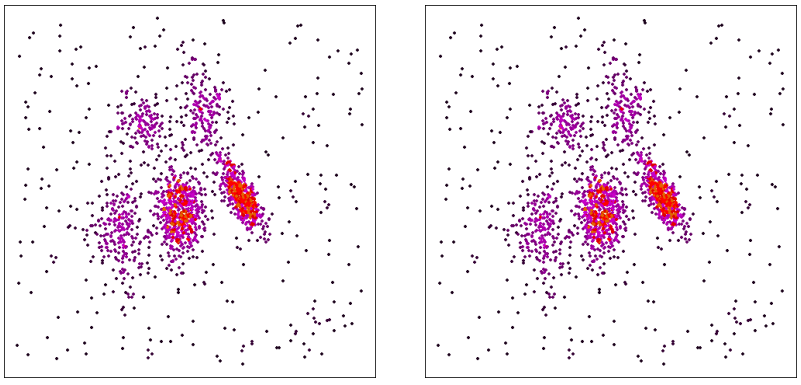

Figure 2: Local density illustration: with 10 strata at left and 20 at right

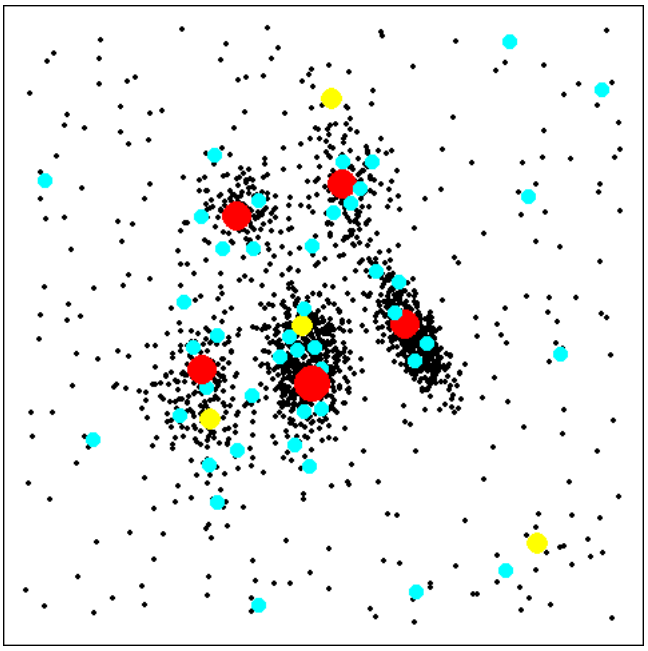

Figure 3: Detected Peaks by importance (red, yellow and cyan)

5. Peripherical algorithms

This section describes all the peripherical algorithms that are needed to run *S-DBSCAN* and produce inputs for *S-DBSCANCORE*.

5.1. Reachability via the k-nearest neighbors

Conventional k-nearest neighbor classification approaches have several limitations when dealing with some problems caused by the special datasets, such as the sparse and noise problem. In our approach, the local and global information of the query pattern are taken into account by hybridizing the notions of neighborhood based on the k nearest neighbors, and those of distance and density. In *S-DBSCAN* the reachability is defined as the number of minimum nearest neighbors k for which a pattern y can be reached from x via the subset of patterns $\Omega_k(x)$ generated by iterating the nearest neighbors function (Equation 12). Let $f^k(x)$ be the search function for the k nearest neighbors of x:

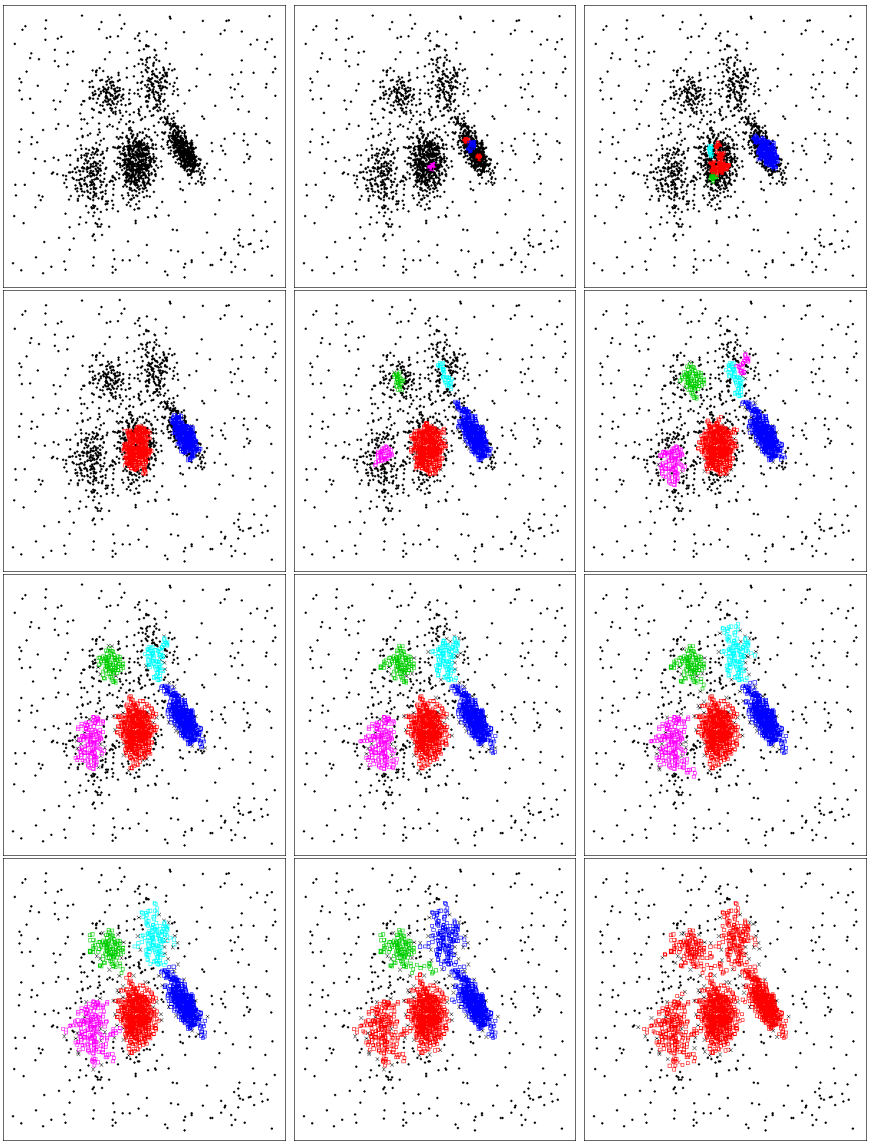

Figure 4: hierarchical process with clusters of varied densities: from left to right and top to bottom.

$$N^k(x) = f^k(x) \tag{11}$$

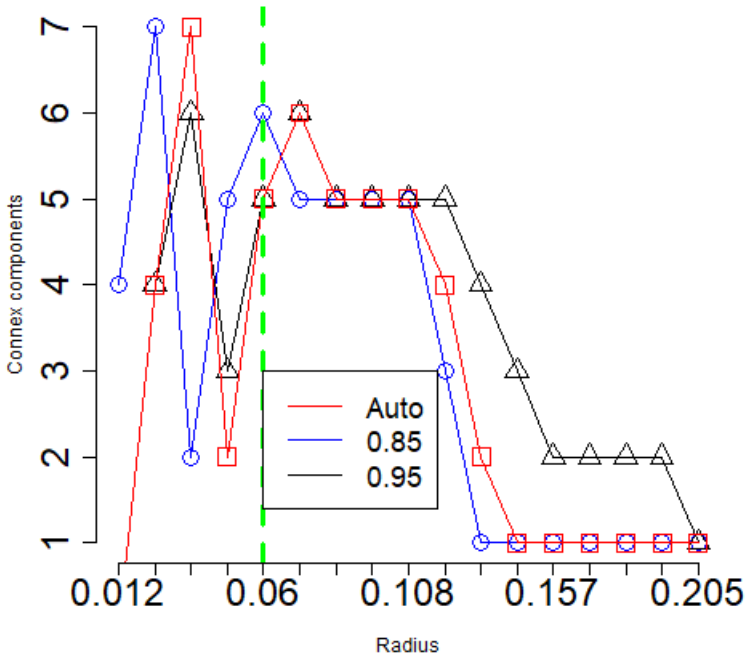

Figure 5: Effect of Sim_{max} : the same plateau is discovered

$$\Omega_k(x) = f^k \circ \ldots \circ f^k(x) \tag{12}$$

$f^k(x)$ takes as input a data pattern and produces k data patterns from which f is again applied. These patterns are marked as visited. To obtain $\Omega_k(x)$, the operation is repeated until there are no more unvisited patterns.

The number of iterations of $f^k(x)$ is such that $\forall y \in \Omega_k(x) \quad N^k(y) \subset \Omega_k(x)$. Then, y is reachable from x, if $y \in \ \Omega_k(x)$. Behind this idea, the goal is to have patterns belonging to same natural cluster that are mutually reachable. In machine learning, pre-defined k values for defining a neighbordhood are set to $\lambda \sqrt{n}$ with $\lambda \leq 1$. The λ coefficient acts on the noise sensitivity as well as the frontier accuracy when the purpose is classification. In any case, k depends on n and has no link with the data structure. For our goal, $k_{max} = \sqrt{n}$ is considered

as an upper bound and the objective is to find k so that the reachability is achieved.

Algorithm 5 Reachability algorithm

```
1: Input: NN, n
2: Output: k
3: k_max = √n, T = 0.95, N = min(1000, n) {fixed variables}
4: for (k = 1 to k_max) do
5:    count = 0
6:    for (i = 1 to N) do
7:       pattern = random(n)
8:       neighbor = NN[pattern][k_max]
9:       for (j = 1 to n) do
            Visited[j]=FALSE
10:      end for
11:      if (Isreachable(Visited, pattern, neighbor, k, NN) == TRUE) then
12:         count = count + 1
13:      end if
14:   end for
15:   if (count ≥ Tn) then
16:      return k
17:   end if
18: end for
19: return k_max
```

The *reachability* algorithm (see algorithm 5) formalizes this idea. It consists in testing a subset of $N = 1000$ patterns and for each pattern determines the number of neighbors needed to reach its k_{max} nearest neighbor, N being justified by [35]. This guarantees with a high probability that at least a minimum of points is taken per cluster via its sample, when C clusters are considered. The main loop of the algorithm (lines 4-18) consists of evaluating k from 1 to the upper bound for N patterns taken randomly. Given a pattern A (line 7), its

k_{max} neighbor is taken (line 8) for the NN matrix and its reachability for a k is tested (line 11).

The algorithm stops when there is a k such that most of the tested patterns the reachability is achieved (line 15). In any case, the maximum value of k is k_{max}. The *Isreachable* algorithm (see algorithm 6) is a recursive algorithm that tests whether for an entry point p_{seed} and a k value, p_{seed} can reach p_{target} by recursively propagating the connectable neighbor patterns of p_{seed}. The algorithm stops as soon as there is a p_{seed} that matches with p_{target} (line 5), or when all the successive neighbor patterns have been visited (line 12) without reaching p_{target}. p_{target} is fixed and p_{seed} is recursively replaced by its k nearest neighbors (line 9).

Algorithm 6 Isreachable algorithm

```
1: Input: Visited, p_seed, p_target, k, NN
2: Ouput: TRUE/FALSE
3: if (Visited[p_seed] == TRUE) then
4:    return FALSE
5: end if
6: if (p_seed == p_target) then
7:    return TRUE
8: end if
9: Visited[p_seed]=TRUE
10: for (i = 1 to k) do
11:    if (Isreachable(Visited, NN[p_seed][i], p_target, k, NN) == TRUE) then
12:       return TRUE {all the neighbors are tested}
13:    end if
14: end for
15: return FALSE
```

5.2. Local density algorithm

The reader is referred to [36] for a recent survey devoted to local density estimation. Whatever the method, how to define the neighbordhood remains a key question. Any efficient algorithm can be used in *S-DBSCAN*. Our local density algorithm is computed using both k nearest neighbors and hypervolume information. The novelty stems from the smart mechanism that is adapted to the reachability notion. To be run, the algorithm needs k, SS and NN. It starts by defining a series of increasing radii r_h. The lower bound (r_{low}) is determined so that 95% of the patterns have their 1_{nn} nearest distance less than r_{low} (line 3). The principle is the same for the upper bound while k is considered instead of 1 (line 4). The range is cut into N_r strata empirically set at 10 (default value). A coarser / thinner cut only slightly impacts the results. Given a pattern A, the main idea is to consider its k neighbors (line 1), select the ones contained in each hypersphere $H_A(r)$(line 17) and process the partial score (line 15) per hypersphere. This score is in the range $[0, 1]$, 1 when the target is similar to A and 0 when the target is located at the frontier (line 18). Then, having the partial score in each stratum, the local density is deduced by averaging the sum (line 22).

5.3. Automatic threshold

The driven parameter Sim_{max} has to be high [0.8-0.99] in accordance with our cluster definition. Except for outliers or some noisy patterns that can have specific local densities, most of the patterns considered in a given neighborhood have a relatively similar density. This is true if there is no cluster and in presence of many clusters. A majority of patterns are not noisy, not outliers and not at a cluster frontier. Clusters can have different densities from one to another and even in the same cluster. However, the density difference for close patterns in the same cluster is comparatively small, at to a lesser extent it is even true for noisy patterns that are more isolated and those that have tiny densities.

Under these assumptions, an automatic threshold can be easily estimated (see algorithm 8) via the distribution θ of Sim_{max} in the largest neighborhood

Algorithm 7 Local density estimation algorithm

Input: k, NN, dist, N_r {default value=10}
Output: $dens_l$ {local density vector}
$r_{low} = \underset{i=1,..,n}{sort}(d^1(i))[95\%n]$ {Quantile of the 1_{nn} distances}
$r_{up} = \underset{i=1,..,n}{sort}(d^k(i))[95\%n]$ {Quantile of the k_{nn} distances}
$\Delta = (r_{up} - r_{low})/N_r$
for ($\lambda = 1$ to N_r) **do**
  $r_h[\lambda] = r_{low} + (\lambda - 1)\Delta$ {radius vector}
end for
for ($i = 1$ to n) **do**
  $dens_l[i]$=0 {initialisation of the pattern density}
  for (λ= 1 to N_r) **do**
    $dens_r[\lambda]$=0 {estimation in a given radius}
  end for
  for ($j = 1$ to k) **do**
    target = NN$[i][j]$
    for ($\lambda = 1$ to N_r) **do**
      if ($dist[i][target] \leq r_h[\lambda]$) **then**
        $dens_r[\lambda] = dens_r[\lambda] + (1. - \frac{dist[i][target]}{r_h[\lambda]})$
      end if
    end for
  end for
  $dens_l[i] = \frac{1}{N_r} \sum_{\lambda=1}^{N_r} dens_r[\lambda]$
end for
return $dens_l$

represented by the number of k nearest neighbors (cf. algorithm *reachability*). By skipping the lowest (10%) values corresponding to the maximum neighborhood potential frontier patterns are rejected as marginal.

Algorithm 8 Automatic threshold determination for Sim_{max}

1: Inputs: n, k, NN, $dens_l$, T_{max}
2: Output: sim_{max}
3: **for** ($i = 1$ to n) **do**
4: sim_{max}=0
5: **for** ($j = 1$ to k) **do**
6: Neigh = NN[i][j]
7: $r = \frac{min(dens_l[i], dens_l[Neigh])}{max(dens_l[j], dens_l[Neigh])}$
8: $sim_{max} = max(sim_{max}, r)$
9: **end for**
10: $\theta_{max}[i] = sim_{max}$
11: **end for**
12: return sim_{max}=$Q(\theta_{\max}, T_{max}\%))$

5.4. Growing core algorithm

The idea of the algorithm (cf. algorithm 9) is to add the candidate patterns, i.e the ones not already included in a partition, in the nearest existing cluster (line 16) using the single link distance.

The process is iterative with k (line 5) allowing the cluster to grow at each step. A restriction (line 13) is added to avoid the inclusion of outlier/noise by comparing the nearest distance with the average of the 1_{nn} distance of the initial patterns included in the closest cluster (line 4). At the end, the remaining patterns are placed in a separate category.

6. Algorithm complexity

The *S-DBSCAN* algorithm requires minimum interaction with the investigator and the hierarchical process is generated with a small number of evaluations (cf. algorithm 1). However, *S-DBSCAN* itself can run only with small or medium datasets. In its current version, it has not been optimized to deal with large data sets as the scalability was not its primarily objective. Most of

Algorithm 9 *GrowingCore* algorithm

Input: k, X, C, NN, d_{euc}
Output: C, noise
Thresh = 0.5, novel = 0
Process μ_{1nn} {the average vector of the 1_{nn} distance for each cluster}
for ($u = 1$ to k) **do**
  taken=TRUE, N_k=0
  while (taken==TRUE) **do**
    taken=FALSE
    for ($i = 1$ to n) **do**
      target=NN[i][u]
      if ($X[i] \in C$) || !(target $\in C$) **then**
        continue {not in the same neighbourdhood}
      end if
      $r = \frac{min(\mu_{1nn}[Cl[target]], d_{euc}[i][target])}{max(\mu_{1nn}[Cl[target]], d_{euc}[i][target])}$
      if ($r <$ Thres) **then**
        taken=TRUE, novel=novel+1 N_k=N_k+1
        C[Cl[target]] = C[Cl[target]] $\cup$ $\{X[i]\}$ {$X[i]$ is added to the same cluster Cl[target] of target}
      end if
    end for
  end while
end for
if ($\|X\| == \|C\|$) **then**
  noise = FALSE
else
  noise = TRUE {the data not included in a cluster are regrouped in a noise category}
end if
return C

implied mechanisms in the algorithm are based on the availability of the SS and NN matrices. Having these matrices avoids distance and nearest neighbours re-

computations. It however needs $O(n^2)$ (exactly $(n^2-n)/2$) and $O(k\times n)$ memory respectively without the use of an accelerating index structure, whereas a non-matrix based implementation of *S-DBSCAN* only needs $O(n)$ memory while being computationally expensive. The number of iterations in the central loop *S-DBSCAN* is governed by g_r that allows to determine the radius increment Δ_r, then in reality g_r runs of the *S-DBSCANCORE* algorithm. *S-DBSCANCORE* visits each point of the database, possibly several times as candidates to different clusters. For practical considerations, however, the time complexity is mostly governed by the number of region query invocations. In the worst case, the run time complexity is however $O(n^2)$ as *S-DBSCAN* executes exactly one such query for each point. The central loop has a run time complexity of $O(n^2)$ in the worst case. The space and time complexities s are summarized in Table 2.

Table 2: Time and space complexity for *S-DBSCAN*

Time complexity	
Construction of S and NN	$O(n^2)$
Reachability algorithm	$O(N \times n)$ (N=1000 by default)
Local density estimation algorithm	$O(k \times N_r \times n)$ ($N_r \leq 20$ and k$\leq n^{0.5}$)
Automatic threshold algorithm	$O(k \times n)$
GrowingCore algorithm	$O(k \times n)$
S-DBSCANCORE processing	$O(n^2)$
S-DBSCAN processing	$O(g_r \times n^2)$
worst cost	$O(g_r \times n^2)$ (optimization: $O(g_r \times n \log(n))$
Space complexity	
S	$O(n^2)$ (exactly $(n^2 - n)/2$)
NN	$O(k \times n)$
dens	$O(n)$
worst cost	$O(n^2)$ (optimization $2 \times O(n \times k)+O(n)$)

In synthesis, the worst complexity case for time complexity remains $O(n^2)$

but can be reduced to $O(n \times \log(n))$ using a Kd-tree structure. For space complexity, the worst case is $O(n^2)$ but can be reduced to $2 \times O(k \times n) + O(n)$ as *S-DBSCAN* only needs for each point its k nearest neighbors, the corresponding distances and the local density of each data point.

In its current version, *S-DBSCAN* has to be combined with sampling and/or coreset techniques [37, 38, 39] to manage large datasets. These techniques aim at organizing, summarizing and finally understanding the data. As a recall, their goal is to select a sample that behaves like the whole, i.e. without losing any valuable information. The idea consists in quantifying the distortion of a given monotonic measure when computed on a sample instead of on the whole set. Then, solving the optimization problem or its approximation on the small coreset yields an approximate solution of the original dataset. The key idea is that the number of samples needed to perform clustering does not depend on the dataset size, it is related to the data structure.

7. Experimental study and discussion

This section aims at demonstrating that *S-DBSCAN* is efficient for a large variety of data structures. Its efficiency was evaluated at two levels: the ability to discover natural clusters and the accuracy of the discovered partition. In all experiments, *S-DBSCAN* was run with the same setting to check its self-tuning power. The parameter Sim_{max} is set up using the algorithm 8. When the ground truth is available accuracy is measured with two non symmetric indices, the *F-measure* (F_m) [40] and the *Mutual Index* (M_I) [41] that compare the expected partition with the one yielded by the algorithm. The scores were processed by skipping the noise both for the ground truth and the result. In real life the reference is unknown and clustering algorithms are used to analyze the data. In the case where there is no reference, only internal validation indices [42] can be considered even if their efficiency is questionable. In this study an extended version of the David-Bouldin index has been selected. Different kinds of experiments are proposed.

The first subsection deals with 2-dimensional datasets as they allow for a human assessment of what partitions are acceptable.

The second subsection illustrates the behavior of the proposal in various difficulties of higher dimensions involving Gaussian and non Gaussian data structures.

The third subsection deals with real data where the ground truth does not stand out. In the fourth subsection a comparison with competitive algorithms was carried out using a selected set of databases introduced in the first subsection. The preprocessing includes a $\{\mu, \sigma\}$ standardization step and, for the datasets with more than four thousand items, a sampling [38] algorithm is applied in order to store the distance matrix in memory and to complete the tests in a reasonable amount of time. This also allows all the competitive algorithms to be run.

S-DBSCAN can be combined with a sampling algorithm to handle large databases. A complementary study is done to compare the efficiency of this combination with two recent algorithms [14, 15] especially developed to handle very large data bases. These algorithms produce approximate and exact *DBSCAN* clustering while improving its run-time performance. The comparison focus on the run-time efficiency.

7.1. Benchmark datasets in two dimensions

A wide range of datasets (16) is used as benchmarks in this section. Some datasets are from the data clustering repository of the computing school of Eastern Finland University[2] while others come from the github repository [3] or were proposed in the published literature. They are usually considered for testing new clustering algorithms. To complete the diversity, homemade data have been added: clusters are different in size, shape, density, amount of noise and degree of separation. Their main characteristics are summarized in Table

[2] `https://cs.joensuu.fi/sipu/datasets/`

[3] `https://github.com/deric/clustering-benchmark/tree/master/src/`

4 and the acceptable partitions given via the ground truth are displayed in Fig. 6. The data may include some variations in the clusters.

Five categories have been established to describe these variations, and for each category there is a code (from 1 to 3) depicting the level of difficulty. As for example, F1 depicts the potential difficulty related to variation of cluster size and F5 the one related to the variation of cluster density.

The main sources of variation with their associated codes are given in Table 3. The datasets are classified as shown in Table 4.

Results are shown in Table 5. The expected number of clusters are displayed in column 2 and the discovered ones in column 3 (Found $\#C$).

The left number corresponds to the cluster number, the number in parentheses in column 3 is the plateau size, and in bold the number of clusters having the largest plateau size. As for example for $D15$, 12 valid clusters have been detected five times, each time corresponding to a radius value, having in mind that the range is set up automatically (cf. Algorithm 1).

D (column 4) indicates success or failure in the discovery process, F_m (column 5) the F-Measure coefficient and M_I the mutual index (column 6).

Table 3: The main sources of variation and their corresponding code from 1 to 3

	1	2	3
Size(F_1)	Similar	Variability	Irregular
Shape(F_2)	Compact	Long/thin	Irregular
Separation(F_3)	Well-separated	Low/very Low	small Overlap
Noise(F_4)	None	Small amount	Large amount
Density(F_5)	No variation	small	Large

7.2. Experiments with datasets of higher dimension

Three kinds of experiments were carried out to assess the behavior of the algorithm in higher-dimensional spaces under different configurations: density peak, pseudo uniform data, more or less separated, different densities. In the

Table 4: The sixteen datasets and their classification in five categories (F1 to F5)

Dataset	Size	#C	Name	Origin	F_1	F_2	F_3	F_4	F_5
D1	3000	4	A.set 1	[43]	2	3	1	1	2
D2	5250	2	A.set 2	[43]	1	2	1	1	1
D3	240	2	FLAME	[44]	1	2	1	1	1
D4	7500	2	A.set 3	[43]	2	3	2	1	2
D5	373	5	JAIN	[45]	2	2	2	1	1
D6	5000	15	S.sets 1	[46]	1	2	1	1	1
D7	5000	31	S.sets 1	[46]	1	1	3	1	1
D8	5401	15	Dim sets 1	Foot 2	1	2	2	1	1
D9	8000	6	Chameleon	[29]	2	3	2	3	1
D10	10000	2	Cluto-t7.10k	Foot 3	2	3	2	3	1
D11	2000	4	H1	Home	1	1	1	2	1
D12	3800	15	H2	Home	3	2	1	2	2
D13	2200	4	H3	Home	2	3	1	1	3
D14	2000	2	H4 (Spiral)	Home	1	2	2	2	1
D15	2500	12	H5	Home	2	3	2	2	2
D16	3500	6	H6	Home	3	2	2	2	3

first one, a series of high-dimensional datasets with Gaussian clusters are tested. The first sub series (from "dimsets low"[47]) is in moderate dimension and the data patterns are partitioned in 9 Gaussian clusters in $d = \{2, 5, 10, 15\}$. The second (from "dimset high" [48]) contains high-dimensional datasets with 16 Gaussian clusters in $d = \{32, 64, 128, 256\}$. Each cluster in a set has the same size that increases linearly as dimensionality increases. These sets were proposed in [48] and clusters are rather well separated. They are denominated $G1, \ldots, G8$. In the second one, 10 datasets, $d = \{3, 6, 9, 10, 18 \ldots, 58\}$, with 3 clusters are tested. They are denominated $S1, \ldots, S10$. They are also Gaussian based but more difficult to discriminate. The separation level is low, shapes and densities are different and some amount of noise is introduced. The formulas used for

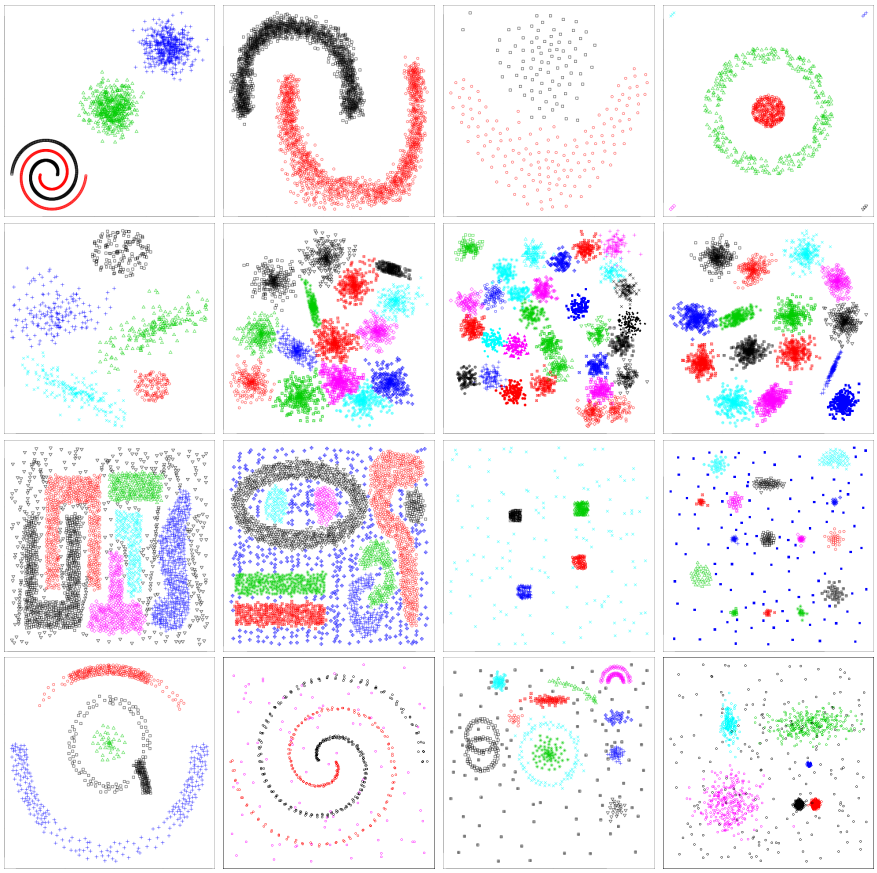

Figure 6: The sixteen datasets. The labels are the x and y coordinates.

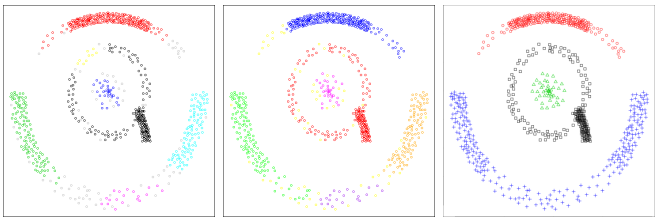

Figure 7: Effect of Sim_{max}=$\{auto, 0.7, 0.6\}$ (database 13)

cluster generation are as follows, i being the dimension:

- the first cluster (green in the Fig. 8) is non spherical: $\mu_i = -2$ if $i \neq 1$ $\mu_1 = 0$, $\sigma_i = 0.5$ if i is odd otherwise $\sigma_i = 0.1 + 2 \times rand(0.1)$;

Table 5: Results for synthetic data set: in bold the number of found clusters having the largest plateau size (in parenthesis).

Dataset	#C	Found #C	D	F_m	M_I
D1	2	5(2),**4(20)**,3(10)	Y	1.00	1.00
D2	2	6(2),5(2),3(2),**2(28)**	Y	1.00	1.00
D3	2	4(1),3(5),**2(20)**	Y	0.989	1.00
D4	2	10(4),8(3),5(2),**2(49)**	Y	0.978	0.984
D5	5	8(4),6(3),**5(6)**,4(8)	Y	0.997	0.998
D6	15	**15(6)**,13(4),12(3),8(1)	Y	0.978	0.992
D7	31	**31(3)**,30(3),16(6),12(2)	Y	0.989	0.991
D8	15	**15(35)**,14(13),13(14),12(2)	Y	0.999	0.998
D9	6	**6(4)**,5(14),4(6),3(5)	Y	1.00	1.00
D10	9	**9(10)**,8(4),5(4),4(4),2(8)	Y	1.00	1.00
D11	4	**4(31)**,5(4),3(10),2(5)	Y	1.00	1.00
D12	15	**15(27)**,9(3),6(5),5(7)	Y	1.00	1.00
D13	4	10(3),9(15),8(9),6(10)	N	0.885	0.866
D14	2	**2(13)**	Y	1.00	1.00
D15	12	**12(5)**	Y	0.967	0.963
D16	6	8(1),7(2),**6(8)**,5(12)	Y	0.978	0.999

- the second one (black) includes two Gaussian components with different densities in each dimension: $\mu_i^1 = 0$, $\sigma_i^1 = 0.4$ and $\mu_i^2 = 0.5$, $\sigma_i^2 = 0.1$;
- the last one (red) is spherical: $\mu_i = -1$, $\sigma_i = 0.3$.

d_r random features were added to the initial dimension space, d_f (see example in Fig. 8). The final dimension is $d = d_f + d_r$. d_r is computed according to Eq. (13).

$$d_r = \begin{cases} d_f/2 & if \quad d_f < 10 \\ d_f & otherwise \end{cases} \tag{13}$$

The third experiment is based on the *genRandomClust* R package. Clusters are now represented by pseudo uniform data instead of density peaks. Again,

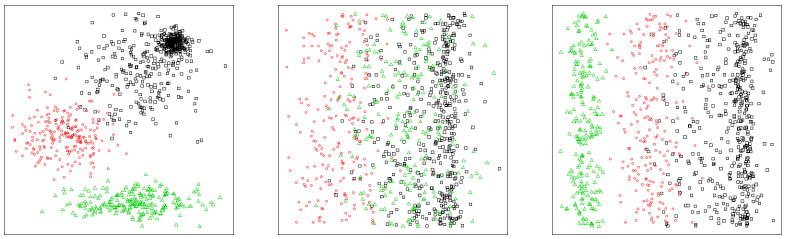

Figure 8: Views of the first dataset, respectively 1-2, 1-3 and 2-3 axis.

clusters have different sizes, shapes, densities and irregularities. This is an implementation of the method proposed in [49]. The degree of separation between any cluster and its nearest neighboring cluster can be set to a specified value regarding the separation index proposed in [50]. The package uses the basic parameters for cluster generation such as the number of clusters, the space dimension and their respective sizes but also allows for variability management. A ratio between the upper and the lower bound of the eigenvalues can be specified. The default value of 10 is used in all the experiments. The range of variances in the covariance matrix was set to the default value, $rangeVar = [1, 15]$. The only parameter used in this experiment is the value of the separation index between two neighboring clusters, $SepVal$. It ranges from -1 to 1. Four typical configurations in 2 dimensions can be seen in Fig. 9. To each dataset 20% of noise is added. The difficulty stems from the hybridization of sources of clustering issues at especially low separation levels (0.1 and 0.2): the variations in density, size (to a lesser extent, shape), a large amount of noise (20%). The number of clusters C was randomly chosen ($rangeK = [3, 6]$) at each configuration to provide diversity. The tests were carried out in d=$\{2, 3, 5, 7, 10, 20\}$ with 4 values of the separation degree: $SepVal = \{0.1, 0.2, 0.3, 0.4\}$. The number of samples per cluster is random in the range $[100 + 20d, 300 + 20d]$ increasing linearly with the dimension. These datasets are denominated $U1, \ldots, U28$. For low values of $SepVal$, the clusters become less and less separable while the space dimension increases due to the curse of dimensionality. As the shape is rather simple, the Silhouette index is processed on the ground truth partition to delineate this

situation. Synthetic results for the three series are given in Table 6. Some synthesis were done when the results were redundant. The average results are provided.

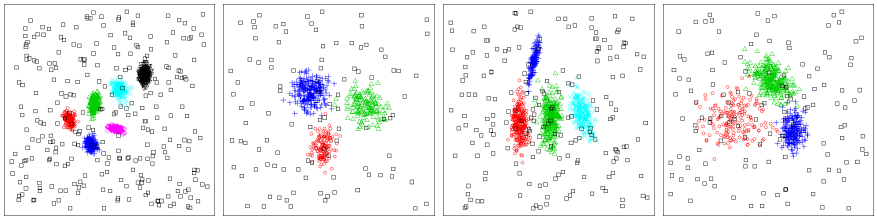

Figure 9: Four configurations of $SepVal$ for random cluster generation in $d = 2$ (axes x and y), from left to right: 0.4, 0.3, 0.2 and 0.1

7.3. Experiments with real data sets

In real life, the original labels of data are unknown and clustering algorithms are used to analyze the data. To determine the cluster quality by features such as similarity, compactness, and separation, an extended version of Davies-Bouldin has been used in order to better measure the efficiency when the discovery is related to more complex cluster shapes.

The idea is very simple: each discovered cluster is divided in sub-clusters via a prototype selection algorithm ([38]) in which the Davies-Bouldin index is applied. The idea behind is to reduce the whole inertia via the division and then obtain more "globular" shapes. Then, instead of applying the Davies-Bouldin index to each complete cluster it is applied to its sub-clusters considering the worse Davies-Bouldin index between sub-clusters. This way presents a fair analytic method and the division in sub-clusters puts the evaluation in a context where the use of a simple index has a sense.

Ten benchmarked data sets denoted $R1$ to $R10$ covering a wide span of applications were selected in order to satisfy a number of criteria such that the collection represents a vast array of applications. These data sets are widely used in the literature and come from popular repository databases as shown in Table 7. For example, The Iris dataset is well known to contain only two clusters

Table 6: Results for data sets in higher dimensions, #C is the expected cluster number (the plateau size being in parenthesis), D the finding result (Y or N), F_m and M_I the internal indexes. In bold the number of found clusters having the largest plateau size (in parenthesis)

Dataset	#C	found #C	D	F_m	M_I
G1-G4	9	**9 (all)**	Y	1.00	1
G5-G8	16	**16 (all)**	Y	1.00	1
S1-S5	3	8(1), 5(3), **3(6)**,2(3)	Y	0.968	0.959
S6-S10	3	**3(7)**,2(5)	Y	0.987	0.988
U1	4	6(1),5(3),**4(3)**,**3(1)**	Y	0.994	0.994
U2	3	4(2),**3(14)**,2(17)	Y	0.999	0.998
U3	6	**6(13)**,5(9),3(2),2(3)	Y	1.00	1.00
U4	4	5(1),**4(22)**,3(10),2(6)	Y	1.00	1.00
U5	4	**4(3)**,3(1),2(3)	Y	0.901	0.912
U6	3	**3(2)**,2(4),3(1),2(6)	Y	1.00	1.00
U7	5	**5(24)**,4(1),3(1),2(6)	Y	0.989	0.978
U8	4	**4(12)**,3(5),2(3)	Y	1.00	1.00
U9	3	3(all)	Y	0.978	0.969
U10	3	**3(2)**,2(1)	Y	1.00	1.00
U11	3	**3(all)**	Y	1.00	1.00
U12	4	**4(20)**	Y	1.00	1.00
U13	6	**1(all)**	N	X	X
U14	3	3 *(manual mode)*	Y	0.971	0.969
U15	3	**3(3)**,2(5)	Y	0.901	0.913
U16	4	**4(2)**,3(1)	Y	1.00	1.00
U17-U18	4,4	1	N	X	X
U19	3	2(4)	N	0.819	0.821
U20	4	**4(4)**,2(8)	Y	0.987	0.986
U21-U23	3,5	1	N	X	X
U24	3	2(3)	N	0.791	0.782

with a rather obvious separation: one of the groups contains the Iris setosa, while the other group contains both the Iris virginica and the Iris versicolor; the latter group cannot be separated without information on the species that Fisher used [4]. Without any ambiguity, *S-DBSCAN* discovers the two clusters (a plateau with 14 points) as shown in Table 7 for three groups, two species being not separable. Fig. 10 displays the discovery process for some databases where different plateaus are visible. The vertical axis shows the number of connex components, the horizontal one the radius and the top axis the number of patterns not selected.

Table 7: Real world datasets: description and results where (#C) is the number of discovered clusters with the largest plateau. In bold the number of found clusters having the largest plateau size (in parenthesis)

Dataset	Size	#d	Name	Or	clusters	D	#C
R1	857357	3	Trans90k	[51]	18(2),**5(9)**	Y	(4)
R2	34112	3	House8	[46]	6(3),**3(6)**,2(3)	Y	(3)
R3	245057	3	Skin	*UCI*	5(3),**3(5)**,2(2)	Y	(3)
R4	440	8	Wholesale	*UCI*	5(8),4(4),**2(10)**	Y	(5)
R5	150	4	Iris	*UCI*	3(1),**2(14)**	Y	(2)
R6	68040	9	Color moment	[51]	**2(14)**,3(1)	Y	(2)
R7	1473	9	Contraceptive	[51]	**6(8)**,9(1),16(2)	Y	(6)
R8	6876	13	Marketing	[51]	4(1),3(7),**2(8)**	Y	(2)
R9	7200	21	Tyroid	[51]	10(2),**11(8)**,12(2), 13(4)	Y	(11)
R10	58000	9	Shuttle	*UCI*	**10(5)**	Y	(5)

7.4. Comparisons with other algorithms

The proposal was compared to 13 other algorithms depicted in Table 8 with their parameters and referred to in section 2. Several competitive algorithms ($A1$, $A6-A9$, $A11-A12$) take the number of clusters as input. These algorithms

[4] `https://fr.wikipedia.org/wiki/Iris_de_Fisher`

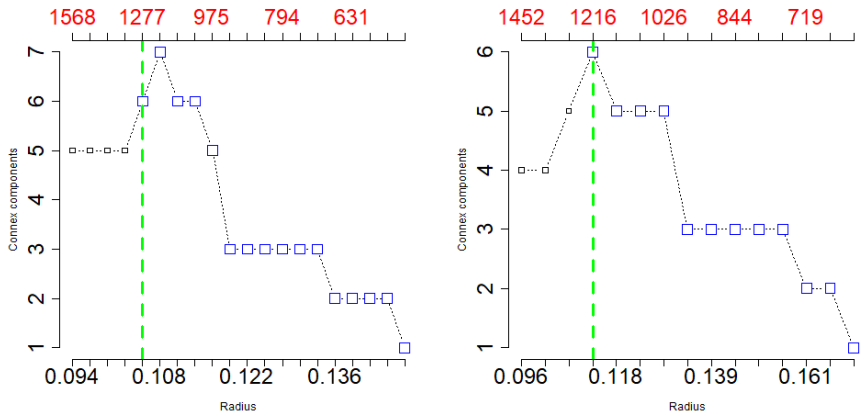

Figure 10: Results for R2 and R3 from left to right. top-bottom. In red, the number of patterns that are not assigned to a cluster (before the growing process driven by the algorithm 9) . The vertical green line presents the frontier from which the found partitions are eligible as enough representative.

were run by varying the number of clusters from 2 to 20. The second ones (in bold) do not require the number of clusters as input. They are more interesting for the discovery aspect. For all the methods, the configurations producing eligible partitions were retained to assess the discovery potential. A discovered cluster is considered if it contains more than 2% of the whole patterns. A partition is eligible if the set of eligible clusters present more than 80% of the whole patterns. Results are shown in Table 9; *S-DBSCAN* is denoted $A14$. For databases without ground truth, the extended David-Bouldin index has been considered to evaluate the eligible partitions.

7.5. Results and discussion

Synthetic data sets. Table 5 highlights the power of *S-DBSCAN* to deal with clusters in different 2D structures. Except for $D13$, the ground truth clusters are easily discovered by the algorithm with high efficiency ($F_m \sim 1$). The stronger the separation, the more the algorithm provides a very distinguishable plateau ($D2$, $D3$, $D8$, $[D11, ..., D15]$) while in other cases suggesting several partitions as for $D1$ 2, 3 and 4 partitions that are clustering compatible. For $D13$, *S-*

Table 8: The competitive algorithms

Algorithm	Acro.	Parameters	Range	Ref
$Kmeans^{++}$	A1	c	$[2, 20]$	[1]
DBSCAN	A2	ϵ, Minpts	$[0.05, 0.25]$, $\sqrt{n} \cdot [0.05, 0.25]$	[19]
R-DBSCAN	A3	ϵ, θ, τ	$\sqrt{d}[0.01, 0.25]$, $\epsilon[1.1, 5]$, $[0.2, 0.6]$	[9]
SNN	A4	k,$Share$	$\sqrt{n}[0.01, 0.3]$, $k[0.05, 0.9]$	[8]
SNN R(ε)	A5	k,$Share$,ε	$\sqrt{n}[0.01, 0.3]$, $k[0.05, 0.9]$, $\sqrt{d}[0.05, 0.3]$	[13]
DPeaks	A6	c, d_c	$[2, 35]$, $\sqrt{d}[0.01, 0.2]$	[23]
DPeaks DF	A7	c	$[2, 20]$	[28]
Comp. Dp	A8	c	$[2, 20]$	[25]
SCDOT	A9	c	$[2, 20]$	[21]
Munec	A10	u	$[0.02, 0.15]$	[3]
HierOpt	A11	c	$[2, 20]$	[30]
Kdmutual	A12	c	$[2, 20]$	[22]
Rnn-DBSCAN	A13	k	$\sqrt{n} \cdot [0.05, 0.5]$	[12]
S-DBSCAN	A14	g	[1, 10]	X

DBSCAN in its automatic mode discovers one more cluster than the ground truth that can be retrieved via a manual tuning ((Fig.7). The performances are promising in higher dimension (Table 6). The algorithm fails however in its discovery process with the U series for low $SepVal$ values when the dimension increases. Clusters are retrieved with a very high Sim_{max} for $U14$ but not for $U[17, 18]$, $U[21...24]$. The difference in the distance between pairs of items diminishes and discrimination is no longer possible. This is attested by the level of the Silhouette index (as example $\{U21, U23\}$ gives $\{0.09\}$) that is relevant in this case. The cluster shapes generated via the *genRandomClust* R package are globular/ellipsoidal.

As shown in Table 9 all the competitors succeed in discovering clusters when they are well-separated ($G5$) even in high dimension, and none succeed when there is a large overlap ($U21$) even in moderate dimensions. Between these two situations, the performances are less clear-cut and more varied. *S-DBSCAN* performs similarly to but better than connective approaches. The reason for this advance is that proposal inherits the strength concepts of *SNN* and *DBSCAN*

Table 9: Synthetic data bases: best F-score index for 9 representative datasets (D6..U21) and 14 competitive algorithms (A1 to A14) where X means no finding and X(C) means that the discovered partition in C clusters represents less than n/2 patterns

	D6	D9	D16	G5	S5	U5	U10	U14	U21
A1	0.91	X	X	0.89	X	0.96	0.97	0.98	X
A2	0.97	X	0.82	1.00	X	0.86(4)	X	X	X
A3	0.98	0.89	1.00	1.00	X	0.86(4)	X	X	X
A4	X	X	X(4)	1.00	X	X	X	X	X
A5	X	X	X(4)	1.00	X	X	X	X	X
A6	0.97	X	0.99	1.00	0.98	0.97	0.97	0.99	X
A7	0.97	X	0.99	1.00	0.98	0.97	0.98	0.99	X
A8	0.97	0.89	0.99	1.00	0.96	0.98	0.97	0.98	X
A9	X	X	0.98	1.00	0.89	X	0.89	X	X
A10	0.99	X(8)	0.99	1.00	0.96	0.98	0.98	X	X
A11	0.96	0.97	0.98	1.00	X	0.98	0.97	0.96	X
A12	0.96	0.98	0.99	1.00	0.96	0.94	0.97	0.97	X
A13	0.99	1.00	0.99	1.00	0.99	0.99	0.99	0.891	X
A14	0.98	1.00	0.98	1.00	0.96	0.99	0.98	0.97	X

while being self-tuning and more adaptative as driven in a differencial mode. The previous methods are static and thus less accurate. For $D16$, *DBSCAN* discovers 5 clusters, *SNN* only 4 while *S-DBSCAN* discovers 6 and 5. For $U5$, *DBSCAN* discovers 4 clusters instead of 5.

By nature, *SNN* algorithms can manage some differential in density but has other limitations. In presence of a small overlap/noisy patterns and with irregular shapes, fixing an unique sharing neighbor value is not enough: one wants to discriminate via this strong condition while in data structure the sharing configuration varies. The shared neighborhood is no longer controlled: small values are likely to produce many small clusters as larger ones can lead to only one partition in the worse case, whatever the sharing level. This issue is managed

with *S-DBSCAN* as the scan acts at different levels as in $D16$. *SNN* algorithms failed to discovering natural clusters with most of the selected datasets. *DBSCAN* approaches perform better than *SNN* ones especially at managing noisy patterns or small overlaps. They however face issues as soon as there is a density difference between clusters and a variability inside clusters, $S5$ being a typical example. With *DBSCAN*, *SNN* and their derived approaches, a tedious and computational multi-tuning is required without any guarantee of success even in moderate dimensions ($D9$ is a typical example).

On the other hand, *DP* families and $Kmeans^{++}$ are highly efficient when clusters match well with density peaks even in presence of noise or overlapping. They succeed with $U14$ whereas all the connective algorithms fail including *S-DBSCAN* in automatic mode. For these cases, there are no natural clusters but one peak per cluster does exist as the shapes are "simple". Up to a given dimension (failure for $U21$) these algorithms can successfully aggregate the corresponding patterns, which is no longer possible for connective algorithms at low separation level. On the contrary, they systematically failed in presence of irregular shapes, large size variation even if low dimensions ($D9$). *Kdmutual* reaches relevant F_m scores as good as *S-DBSCAN* but as for the previous algorithms it needs the cluster number as input.

HierOpt and *SCDOT* are very slow and perform less well than *S-DBSCAN*. *Munec* achieves more comparable results but needs the u parameter to be tuned which is not straightforward as it is based on complex heuristics. *S-DBSCAN* is computationally interesting (Table 12). It requires only one run and at worse few trials in the manual mode.

Real world data sets. Real world data sets include more different data organizations more difficult to handle. For each algorithm, the tuning produces a partition that is formalized by the number of clusters and the associated extended David-bouldin score. The main results are given in Tables 10 and 11: each value is the minimum extended David-bouldin score obtained for each database by varying the input parameter(s) of each algorithm. The last columns depicts

the percentage of discovery. Table 10 and 11 relate partitions with less or equal 5 (10) clusters. It can be seen that the results highlight significant variation between the competitive algorithms.

Table 10: Competitive results for real data bases (from R_1 to R_{10}) and 14 competitive algorithms (from A_1 to A_{14}) : min david-bouldin value for $\#C$ found ≤ 5 where X means no finding.

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	%
A_1	1.68	1.22	1.72	1.16	0.40	1.94	1.28	1.99	1.02	1.49	100.00 %
A_2	0.95	0.66	0.64	0.38	X	0.83	X	X	X	0.48	60.00 %
A_3	0.95	0.66	0.64	0.43	X	0.83	X	X	X	0.48	60.00 %
A_4	1.21	1.02	0.34	X	X	X	X	X	X	X	30.00 %
A_5	1.21	0.11	0.34	X	X	0.83	X	X	X	X	40.00 %
A_6	1.08	1.61	1.62	1.03	0.40	2.14	1.38	2.06	1.56	0.81	100.00 %
A_7	1.78	1.61	1.75	1.02	0.40	1.99	1.38	2.11	1.51	0.81	100.00 %
A_8	1.09	1.61	1.62	1.03	0.40	2.13	1.38	2.12	1.47	0.98	100.00 %
A_9	1.86	0.47	0.65	1.13	0.88	1.07	1.26	1.10	X	X	80.00 %
A_{10}	4.01	1.73	X	1.04	0.39	2.16	1.31	X	2.42	1.97	70.00 %
A_{11}	1.12	1.39	1.15	0.61	0.40	X	1.13	0.62	0.99	1.03	90.00 %
A_{12}	1.09	1.56	1.67	1.17	0.40	2.04	0.87	2.07	1.01	0.93	100.00 %
A_{13}	X	0.73	0.93	0.87	0.34	X	X	X	X	X	40.00 %
A_{14}	X	1.16	1.84	0.92	0.43	1.90	1.06	2.25	1.11	0.80	90.00 %

S-DBSCAN found out natural clusters for 6 databases among 8 that are correctly discriminated regarding the index. The algorithms working with the number of clusters find eligible partitions. They deliver good extended David-Bouldin scores. This has a certain logic as these algorithms work on the basis of prototype detection. Even in case of the absence of natural clusters they can determine partitions.

It is not the case for the other algorithms that work differently and can produce more complex shapes. Except for the *Munec* algorithm that proves to

discover eligible partitions in many cases, the other algorithms working without the number of clusters as input have real difficulties to discover eligible partitions. It is particularly true when the dimension space is not low. We think that an extensive tuning is likely to improve the results of the competitors but with the consequence to degrade the usability for the investigator. It should be highlighted that the *Rnn-DBSCAN* algorithm that is a recent algorithm fails in many cases. In comparison with the majority of other algorithms that discover eligible partitions from 70% to 100% (for 5 clusters), only 40% is reached for *Rnn-DBSCAN*. This algorithm is also particularly slow compared to the other algorithms.

Table 11: Competitive results for real data bases (from R_1 to R_{10}) : min index value for $\#C$ found ≤ 10 where X means no finding.

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	%
A_1	1.39	1.06	1.16	0.69	0.38	1.85	1.11	1.84	0.81	0.73	100.00 %
A_2	0.74	0.56	0.49	0.47	0.32	X	0.49	X	X	0.39	70.00 %
A_3	0.88	0.56	0.49	0.47	0.32	1.89	0.49	X	X	0.42	80.00 %
A_4	0.82	0.56	0.34	X	X	X	X	X	X	X	30.00 %
A_5	0.85	0.77	0.34	0.93	X	X	1.10	X	X	X	40.00 %
A_6	0.85	0.79	1.10	0.94	0.40	2.14	1.11	2.02	1.13	0.81	100.00 %
A_7	0.89	0.79	1.26	0.32	0.40	1.99	1.15	1.99	1.15	0.81	100.00 %
A_8	0.88	0.77	1.10	1.03	0.40	1.77	1.16	1.79	0.87	0.98	100.00 %
A_9	0.96	0.47	0.65	1.13	0.88	1.07	0.92	0.90	X	0.49	90.00 %
A_{10}	1.23	1.08	1.30	0.98	0.37	2.16	1.07	2.07	1.49	1.50	100.00 %
A_{11}	0.80	0.49	0.28	0.13	0.38	X	1.09	1.09	0.85	1.03	90.00 %
A_{12}	0.77	0.89	1.11	0.93	0.40	1.93	0.77	1.91	0.87	0.89	100.00 %
A_{13}	X	0.46	0.88	0.87	0.34	X	0.94	X	X	X	50.00 %
A_{14}	0.99	1.16	1.84	0.88	0.43	1.90	0.93	2.14	0.88	0.80	100.00 %

S-DBSCAN appears to have more ability for the discovery task than its competitors working without specifying the number of clusters as input. It

works with only one input parameter and its great advantage is to have an automatic tuning. Except for R8, the corresponding extended David-Boulding score is less than 2 that also assesses the quality of the discovered partitions.

Running time comparison. Except of *kmeans*$^{++}$ that has $O(\lambda \times n)$ time complexity, the other algorithms in their primarily versions have $O(n^2)$ complexity. Complexity reduction is possible for all the algorithms (including our proposal) using a kd tree structure as done in [33, 32, 34, 14]. Except of *kmeans*$^{++}$ and *Dpeak* algorithms that have $O(\lambda \times n)$ space complexity, the others have $O(n^2)$ as exploiting the distance matrix in memory to highly reduce their global computational costs. Even with comparable complexities, there are significant differences in the running time between the competitors to handle the experimental databases.

The average time for one run was calculated for each algorithm and compared with the median time (*DBSCAN* is taken as reference).

The results are summarized in Table 12 where in line 2 the values are the relative multiplicative factors of rapidity in average, and in line 3 the number of parameters to be tuned (# Param).

By tuning each parameter t times, the whole time can be approximated by the product of the values in each row. For example, *kmeans*$^{++}$ is 27 times faster than *DBSCAN* for 1 run but needs to be launched t^2 times to tune its two input parameters.

Table 12: Runing time (1 run). Comparative results with DBSCAN (A2) as reference.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14
Unit	27	1	1	1	1	9.1	9.1	9.1	0.2	2.3	0.05	0.1	0.05	0.8
# Param	1	2	3	2	3	2	1	1	1	1	1	1	1	1

7.6. Additional experiments to compare S-DBSCAN with scalable algorithms to handle very large databases

In this experiment, the scalability and efficiency of the proposed method is analyzed to handle very large datasets. A comparison is done with two scalable and derivate algorithms of *DBSCAN*. Experiments were performed on 15 synthetic datasets (#D), their sizes s ranging from 5K to 15M, and dimensions d from 2 to 10. They are partitioned in 5 non overlapped clusters using a multivariate normal distribution for each cluster. The cluster centers are generated by random sampling from a uniform distribution $[-10; 10]^d$, and the covariance matrix is taken as an 2D symmetric matrix of $[d \times d]$ size where diagonal elements are also randomly generated from a uniform distribution (diagonal elements $[0.1; 1]$ and the others $[-2; 2]$). To each data set, 10% of uniform noise $[-10; 10]^d$ is added. *S-DBSCAN* is pre-processed using the progressive sampling framework in [39] with the *Protras* algorithm [38]. Then, *S-DBSCAN* is run with sampled data of size s' instead of s.

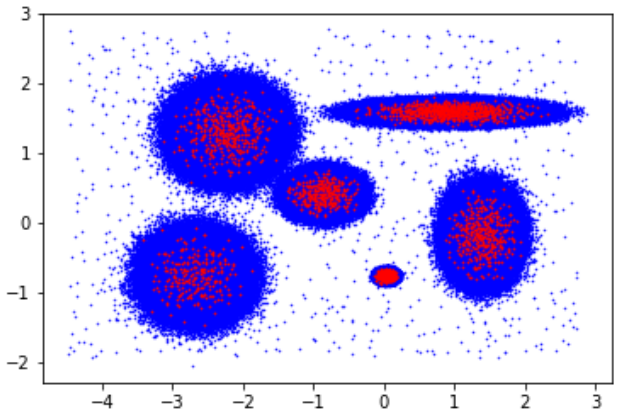

Figure 11: Sampling procedure to pre-process *S-DBSCAN*: in blue the original patterns from s and in red the patterns from s'.

A simple database example using a diagonal covariance matrix with 15M

data in 2D and is displayed in Figure 11. This database is partitioned in five clusters of sizes 1.1M, 3M, 0.5M, 8M, 1.2M, 1.2M and contains additional uniform data noise patterns. The ratio between the sizes is $r = \frac{|s'|}{|s|} = 0.017\%$ and *S-DBSCAN* can easily retrieve the 5 clusters. The results of this experiment are provided in Table 13: t_s is the computational time using a single computer to obtain s' and t_s to run *S-DBSCAN* on a given database D. $t_{IP}(s)$ is the time for *S-DBSCAN* to obtain S, NN and dens, and $t_{1run}(ms)$ the average time for one run of *S-DBSCANCORE* corresponding to a given radius; T(s)= t_s + $t_{IP}(s)$ + $t_{1run}(ms)$ is the total time. t_{alg1} and t_{alg2} are the average computational time (over 10 tests) to run *G-DBSCAN* [15] (exact *DBSCAN* clustering) and *BLOCK-DBSCAN* [14] (approximate *DBSCAN* clustering) respectively on D of size s. The time to put the data in the RAM is not considered as the same for all the algorithms. The value "X" in the Table 13 means that the computational time is more than 30 min.

Table 13 is related to 1 run. In reality, to discover a partition, *BLOCK-DBSCAN* and *G-DBSCAN* have two parameters to be tuned while *S-DBSCAN* only one. For a given number of tuning t per parameter, this means that the whole time for *S-DBSCAN* is t_s+t_{IP}+ $t \times t_{1run}$ when it is $t^2 \times t_{alg1}$ and $t^2 \times t_{alg2}$ for the competitive algorithms. Concerning *S-DBSCAN*, most of computational time is related to the pre-processing step. As the distance matrix is available, the scanning part is not dependant on the space dimension. The scalable approaches produces tiny computational time with small databases, several ms in average to process a database of 5K patterns. There are more time efficient than *S-DBSCAN* applied without sampling (D1 to D3). Despite their smart complexities (at the level of $O(n \times \log(n))$ they become more questionable for larger databases. With a single computer, one talks about tens of seconds for 100K patterns, several minutes for 1M of patterns and more than 30 minutes for 15M of patterns. As t^2 tests are needed to tune the *S-DBSCAN* parameters, the discovery process even with scalable algorithms is time consuming. It should be noted that *BLOCK-DBSCAN* appears to be faster than *G-DBSCAN*, the difference being really relevant with databases having several tens of K pat-

Table 13: Competitive results with two scalable algorithms alg_1 (*G-DBSCAN* [15]) and alg_2 (*BLOCK-DBSCAN* [14]) to handle large databases. X means that the computational time is more than 30 min.

#D	s (Million)	d	s'(Kilo)	$t_s(s)$	$t_{IP}(s)$	$t_{1run}(ms)$	T(s)	$t_{alg1}(ms)$	$t_{alg2}(ms)$
D1	0.005	2	5	0	1.190	5.89	1195.89	654	532
D2	0.005	5	5	0	1.416	6.11	1422.11	676	643
D3	0.005	10	5	0	1.527	6.71	1533.71	712	676
D4	0.01	2	3.112	2.177	1.120	5.89	3302.89	1167	861
D5	0.01	5	2.389	2.975	1.376	6.11	4357.11	1211	932
D6	0.01	10	2.521	2.106	1.324	6.71	3436.71	1312	982
D7	1	2	3.112	2.177	1.120	5.89	3362.89	$1.426\ 10^6$	$2.832\ 10^5$
D8	1	5	2.389	2.975	1.376	6.11	4357.11	$1.580\ 10^6$	$8.132\ 10^5$
D9	1	10	2.521	2.706	1.324	6.71	4036.11	$1.628\ 10^6$	$9.372\ 10^5$
D10	8	2	3.102	2.455	0.942	5.51	3302.51	X	$1.654\ 10^6$
D11	8	5	3.872	2.522	1.201	5.12	5078.12	X	$3.728\ 10^6$
D12	8	10	3.221	2.898	1.421	4.68	4323.68	X	$4.872\ 10^6$
D13	15	2	2.829	2.392	1.102	5.24	3499.24	X	X
D14	15	5	3.301	2.342	1.280	4.89	3626.49	X	X
D15	15	10	3.529	2.876	1.442	5.09	4823.09	X	X

terns. Compared to the scalable approaches, the hybridization with sampling approaches is more competitive. The computational time to obtain the sampled data is low as the process is based on a progressive sampling approach [39] that generates s' without inspecting the whole database. This time is more related to the data organization and than the size of the dataset. Thanks to this pre-processsing step, *S-DBSCAN* is always applied to a reduced set where the hierarchical partitions can be obtained in several seconds.

7.7. Behavior of the algorithm regarding the SizeMin parameter

In this experiment, the goal is to study the effect of the $SizeMin$ parameter. By default, this parameter is set up to $n/100$ (n is the data dimension) behind

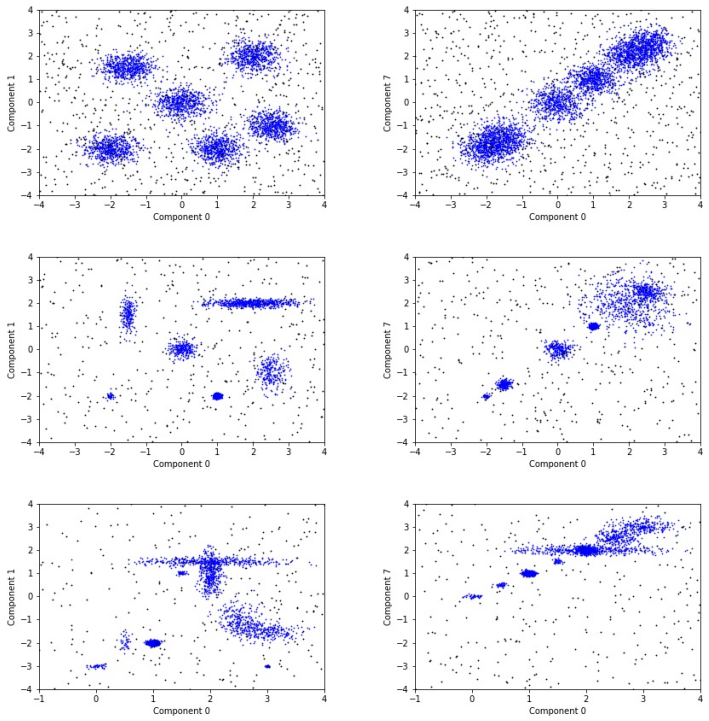

Figure 12: Projection of data in 2D for T2, T3 and T4 with different n and d.

the idea that below this value a cluster does not afford a real contribution for the final partition. Experiments were performed on four types of data for which n and d (space dimension) vary: The first one (T1) is an uniform random pattern distribution without being structured in clusters, the second one (T2) contains 6 "globular" separated clusters having similar characteristics (same size, multivariate normal Gaussian distributions with σ_{ij}=0.3 + $random(0, 0.1)$ in each dimension j). When $d > 2$ one has c_{ij}=c_{i0} for all i and j. The third one (T3) contains 6 separated clusters having different characteristics (size, shape and internal density) where one cluster has a tiny size compared to the others. The fourth one (T4) is more complex. It is a mixture of multivariate normal distributions with varied densities producing 7 clusters: 2 major clusters having complex shapes with a small overlapping, and 5 minor clusters plus or less

separated to the others.

For T3 and T4, the protocol is the same. The centers and standard deviations are defined in 2D. Concerning d, all the center components c_j are fixed according to the first one : c_{ij}=c_{i0}) for all clusters i and $j > 2$. The standard deviation σ_{ij} is fixed as follows : σ_{ij}=min(σ_{i0},σ_{i1}). To each data set, 20% of uniform noise is added. The primal size s is 1080 patterns ($900 + 20\% \times 900$). The other sizes are a multiple of s: $n = \lambda \times s$. Figure 13 shows the data projections in 2D (components 0-1 and 0-7) for T2 ($n = 5s$), T3($n = 2s$) and T4($n = 2s$) with $d = 10$. The algorithm is evaluated without preprocessing. Both n and d are varied and the effect of $SizeMin$ values has been studied for each dataset type representing 45 sub-datasets ($9[d] \times 5[n]$) for each type : n is ranged from s to $5s$ (up to available memory in a single computer), d from 2 to 10 and four $SizeMin$ values are tested : $n/1000$, $n/500$, $n/200$ and $n/100$.

Table 14 gives the computational time C_t for different couples (n, d) and each data type. It corresponds to the mean and standard deviation of the times obtained using the four $SizeMin$ values. The granularity g_r (cf. Algorithm 1 line 1) is fixed at 20 but the number of iterations of *S-DBSCAN* (cf. Algorithm1) can vary according to the end criterion. For a precise comparison, each value of C_t corresponds to the time to process I_p (t_{IP}) added to the mean time (t_{1run}) to compute 1 scan iteration (cf. Algorithms 3 and 9). It should be noted that t_{IP} does not depend on $SizeMin$ but it includes a random share.

$$C_{t_{\mu,\sigma}} = [\mu, \sigma](t_{IP} + t_{1run}) \tag{14}$$

where t_{IP} and t_{1run} are vectors in 4 dimensions.

As shown in Table 14, the computational time increases with (n, d) with very small variations due to $SizeMin$. As t_{IP} does not depend on $SizeMin$, a deeper analysis is given in Table 15 where only the t_{1run} time for each database is studied for $n = 4s$, $d = [2 \dots 10]$ and $SizeMin = [n/100, n/200, n/500, n/1000]$. It can be shown that for each data type, the t_{1run} time in milliseconds is very close to each other regarding $SizeMin$ and d. These results confirm that the

computational time essentially depends on (n, d) and not on $SizeMin$. It is mainly due to t_{Ip} that varies with (n, d) and one has $t_{1run} \ll t_{Ip}$. A Wilcoxon signed-rank was performed to confirm this point. Several couple (a, b) of series of data were considered in T1, T2, T3 and T4 with different values of n and d; a is the average time under 10 runs for one $SizeMin$ value and b for another one. The p-value is the probability of observing the computed statistic, given the experimental conditions, under the null hypothesis, meaning that both time values come from a unique population, *i.e.* there is no time difference between different $SizeMin$ values. The Wilcoxon does not support the rejection of the null hypothesis at 5%.

Table 14: Computational time C_t when n and d vary. Each value corresponds to the mean time in second to process $t_{Ip} + t_{1run}$ for 4 $SizeMin$ values. The value in parenthesis is the standard deviation.

n \ d	2	4	6	8	10
T1(s)	1.47(0.02)	1.52(0.01)	1.61(0.02)	1.64(0.03)	1.73(0.04)
T1(3s)	14.69(0.1)	14.70(0.08)	14.74(0.11)	14.81(0.09)	16.47(0.13)
T1(5s)	45.34(0.12)	46.08(0.13)	46.90(0.09)	47.55(0.12)	48.71(0.11)
T2(s)	1.47(0.01)	1.84(0.01)	1.88(0.01)	2.01(0.02)	2.05(0.01)
T2(3s)	15.28(0.12)	15.48(0.2)	15.58(0.18)	16.27(0.1)	16.71(0.13)
T2(5s)	50.25(0.2)	51.17(0.21)	52.15(0.19)	53.4(0.14)	53.84(0.23)
T3(s)	1.84(0.01)	2.13(0.02)	2.1(0.01)	1.87(0.02)	1.94(0.01)
T3(3s)	15.33(0.17)	15.67(0.12)	17.10(0.09)	17.14(0.14)	19.66(0.19)
T3(5s)	43.45(0.21)	51.92(0.22)	52.19(0.19)	53.82(0.28)	55.90(0.23)
T4(s)	2.21(0.01)	2.66(0.01)	2.91(0.02)	3.06(0.02)	3.2(0.01)
T4(3s)	17.42(0.13)	18.27(0.19)	18.82(0.16)	19.32(0.14)	19.81(0.16)
T4(5s)	51.61(0.24)	52.18(0.22)	54.72(0.28)	56.11(0.25)	57.07(0.29)

In terms of accuracy, *S-DBSCAN* perfectly detects the internal structure of the data without any ambiguity and whatever the configurations for T1 and T2.

Table 15: t_{1run} in milliseconds for $n = 4s$: each value is the mean of t_{1run} through the iterations for a given $SizeMin$ and d. The number of iterations depends on the end criterion but there are at least 20 iterations as the granularity $g_r = 20$.

$SizeMin$ \ d	2	3	4	5	6	7	8	9	10
T1($n/100$)	25.61	30.86	32.13	33.41	32.81	33.13	33.70	31.70	32.00
T1($n/200$)	26.01	30.28	29.85	28.25	28.53	31.58	31.62	31.92	31.22
T1($n/500$)	27.88	29.27	27.95	28.18	29.58	31.60	28.63	29.89	31.21
T1($n/1000$)	24.23	28.63	30.16	32.95	28.93	25.22	30.27	31.22	31.23
T2($n/100$)	12.32	12.91	10.83	10.90	10.72	11.04	11.40	11.68	10.95
T2($n/200$)	12.72	12.65	10.71	10.78	10.80	11.11	11.10	11.12	10.00
T2($n/500$)	11.71	13.03	10.74	10.68	10.82	10.99	12.12	12.01	11.12
T2($n/1000$)	8.93	13.25	10.95	10.69	10.54	11.60	11.59	11.09	11.50
T3($n/100$)	12.88	15.24	15.77	17.53	17.86	17.68	19.55	19.47	18.96
T3($n/200$)	11.74	16.90	16.76	18.28	17.98	18.24	19.95	20.53	18.47
T3($n/500$)	10.41	15.90	17.97	17.68	17.44	17.74	18.90	18.53	20.87
T3($n/1000$)	10.41	15.12	16.06	17.44	17.93	17.48	19.49	19.41	21.58
T4($n/100$)	13.19	18.46	18.45	22.36	18.36	18.63	16.5	18.33	18.22
T4($n/200$)	13.78	18.60	17.99	22.02	18.25	18.18	16.91	18.21	18.70
T4($n/500$)	13.42	19.01	18.02	21.99	20.25	20.02	17.52	19.04	19.03
T4($n/1000$)	13.11	18.31	17.41	21.71	18.77	19.18	18.13	19.22	18.83

For T3, the first 5 main clusters are always perfectly discovered but it is not the case for the 6^{th} regarding its comparable tiny size (see sub-Figures 13.1 and 13.2) for $SizeMin = n/100$. The result is similar for T4 concerning $SizeMin$. The smallest clusters are found for $SizeMin = n/1000$ but other small clusters can also be detected particularly in low dimensions (see Sub-Figures 13.5 and 13.6) according to r_{act}. It is due to the presence of noisy patterns that have more chance to be structured than in high dimensions. Some clusters are not well-separated in T4. As shown in Figure 13, *S-DBSCAN* discovers the expected

partitions with high accuracy ($Fm > 0.9$ and $MI > 0.9$) but also suggests some variants as a function of r_{act} (cf. Algorithm 1) that are easily interpretable. Expected clusters that are very close to each other can be regrouped in one cluster (see sub-Figures 13.6, 13.7 and 13.8).

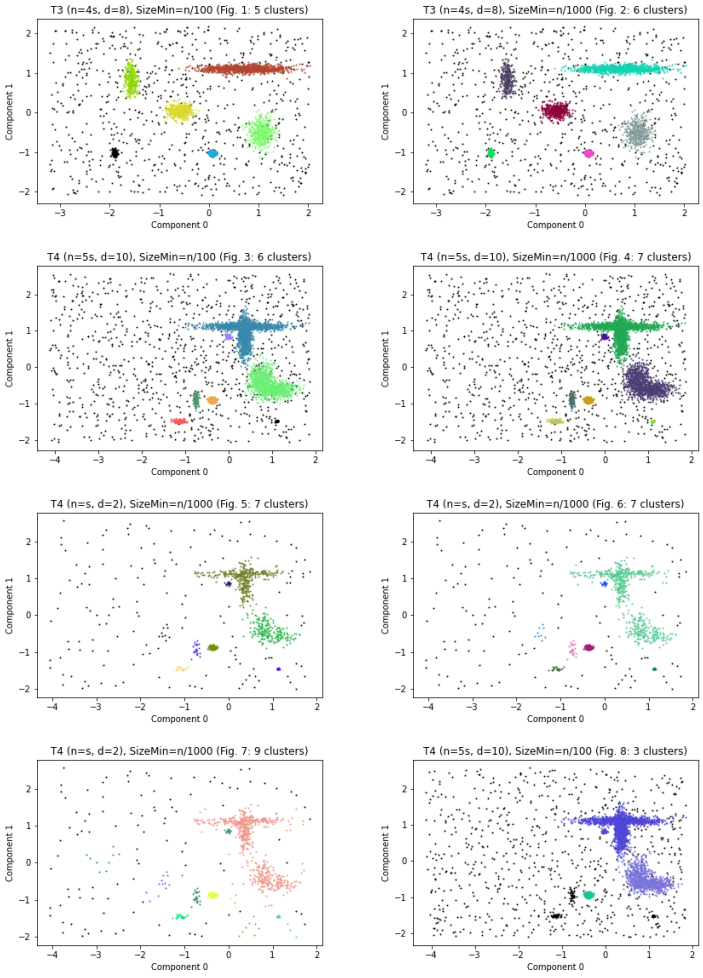

Figure 13: Data types: Examples of discovery for T3 and T4 when n, d and $SizeMin$ vary. The detected clusters (seen in 2 dimensions) are in color and the noisy patterns in black.

The synthesis is as follows:

- There is no visible effect on the computational time when $SizeMin$ varies.

This is explained by the fact that all the points are visited via the scan whatever the $SizeMin$ value.

- The computation time increase with n and d. It is due to the distance calculations but there is non link with the $SizeMin$ value in our implementation.

- With small $SizeMin$ values, more small groups can be identified, they can be real clusters or not. The major clusters remain the same.

- The algorithm does not logically allow to discover real clusters below the $SizeMin$ value. Then, according to the importance of such groups for very large data sets, the investigator can adapt the value if needed regarding n. As the scanning time is negligible compared to the preprocessing tasks (to find S, NN, k, dens and Simmax) the tuning is easy and quick. The process should be to use the default value $n/100$ for a first discovery and smaller values can serve for a deeper analysis.

8. Conclusion

S-DBSCAN is simple, deterministic, non iterative and is able to discern a wide class of data of arbitrary shapes and sizes in presence of noise and outliers. *S-DBSCAN* is almost full automatic and requires minimum interaction with the investigator. It is insensitive to the ordering of the points in the database, the process being driven by the potential of each data pattern to be a density peak.

S-DBSCAN scans the database by applying its algorithm core *S-DBSCANCORE* on the basis of an increasing series of influence zones. *S-DBSCANCORE* groups together data patterns that are closely packed together with respect to the differential density.

Experimental results showed that *S-DBSCAN* is powerful for the discovery process without needing the cluster number as input. It is efficient in terms of accuracy compared to connectivity-based competitive algorithms. When "nat-

ural" clusters exist *S-DBSCAN* can discover them without any tuning, which is a strong asset.

In its pioneer version, scalable methods perform better on computational time and *S-DBSCAN* has to be combined with sampling approaches to handle large databases. This is mainly due to its memory complexity and the necessity to process matrix distance.

Time complexity should be reduced to $O(n \times \log(n))$ using a kd-tree implementation as in [32] and space complexity to $2 \times O(k \times n) + O(n)$ as only the k nearest neighbors, their associated distances and the local density at each point is needed to run *S-DBSCAN*.

As most of clustering algorithms, the performance of *S-DBSCAN* depends on the distance measure used, the most common one being the Euclidean distance. Especially for high-dimensional data, this metric can be rendered almost useless due to the so-called "Curse of dimensionality". Concentration of pairwise distances and violation of neighborhood structure weaken the process.

The next advances could be based on a novel implementation based on an efficient indexing data structure and a systematic pre-processing integrated stage. The latter would deal with very large databases and handle high dimensional data by using either an unsupervised feature selection or an input space transform.

References

[1] A. K. Jain, Data clustering: 50 years beyond k-means, Pattern Recognition Letters 31 (2010) 651–666.

[2] X. Xu, S. Ding, Z. Shi, An improved density peaks clustering algorithm with fast finding cluster centers, Knowledge-Based Systems 158 (2018) 65–74.

[3] F. Ros, S. Guillaume, Munec: A mutual neighbor-based clustering algorithm, Information Sciences 486 (2019) 148–170.

[4] R. Salman, V. Kecman, Q. Li, R. Strack, E. Test, Fast k-means algorithm clustering, arXiv preprint arXiv:1108.1351 (2011).

[5] P. Domingos, G. Hulten, A general method for scaling up machine learning algorithms and its application to clustering, in: ICML, volume 1, 2001, pp. 106–113.

[6] Y. Lv, T. Ma, M. Tang, J. Cao, Y. Tian, A. Al-Dhelaan, M. Al-Rodhaan, An efficient and scalable density-based clustering algorithm for datasets with complex structures, Neurocomputing 171 (2016) 9–22.

[7] S. Wang, A. Gittens, M. W. Mahoney, Scalable kernel k-means clustering with nyström approximation: relative-error bounds, The Journal of Machine Learning Research 20 (2019) 431–479.

[8] R. A. Jarvis, E. A. Patrick, Clustering using a similarity measure based on shared near neighbors, IEEE Transactions on computers 100 (1973) 1025–1034.

[9] Y. Zhu, K. M. Ting, M. J. Carman, Density-ratio based clustering for discovering clusters with varying densities, Pattern Recognition 60 (2016) 983–997.

[10] J. Xie, W. Jiang, An adaptive clustering algorithm by finding density peaks, in: Pacific Rim International Conference on Artificial Intelligence, Springer, 2018, pp. 317–325.

[11] J. Xie, W. Jiang, L. Ding, Clustering by searching density peaks via local standard deviation, in: International Conference on Intelligent Data Engineering and Automated Learning, Springer, 2017, pp. 295–305.

[12] A. Bryant, K. Cios, Rnn-dbscan: A density-based clustering algorithm using reverse nearest neighbor density estimates, IEEE Transactions on Knowledge and Data Engineering 30 (2018) 1109–1121.

[13] L. Ertöz, M. Steinbach, V. Kumar, Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data, in: Proceedings of the 2003 SIAM International Conference on Data Mining, SIAM, 2003, pp. 47–58.

[14] Y. Chen, L. Zhou, N. Bouguila, C. Wang, Y. Chen, J. Du, Block-dbscan: Fast clustering for large scale data, Pattern Recognition 109 (2021) 107624.

[15] K. M. Kumar, A. R. M. Reddy, A fast dbscan clustering algorithm by accelerating neighbor searching using groups method, Pattern Recognition 58 (2016) 39–48.

[16] A. Sarma, P. Goyal, S. Kumari, A. Wani, J. S. Challa, S. Islam, N. Goyal, μdbscan: an exact scalable dbscan algorithm for big data exploiting spatial locality, in: 2019 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2019, pp. 1–11.

[17] Y. Song, Y. Gu, R. Zhang, G. Yu, Brepartition: Optimized high-dimensional knn search with bregman distances, IEEE Transactions on Knowledge and Data Engineering (2020).

[18] Z. Liu, C. Wu, Q. Peng, J. Lee, Y. Xia, Local peaks-based clustering algorithm in symmetric neighborhood graph, IEEE Access 8 (2019) 1600–1612.

[19] 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: Proceedings of the Second International Conference on Knowledge Discovery and Data, 1996, pp. 226–231.

[20] A. Hinneburg, D. A. Keim, A general approach to clustering in large databases with noise, Knowledge and Information Systems 5 (2003) 387–415.

[21] Q. Cheng, X. Lu, Z. Liu, J. Huang, G. Cheng, Spatial clustering with density-ordered tree, Physica A: Statistical Mechanics and its Applications 460 (2016) 188 – 200.

[22] F. Ros, S. Guillaume, M. El Hajji, R. Riad, Kdmutual: A novel clustering algorithm combining mutual neighboring and hierarchical approaches using a new selection criterion, Knowledge-Based Systems (2020) 106–220.

[23] A. Rodriguez, A. Laio, Clustering by fast search and find of density peaks, Science 344 (2014) 1492–1496.

[24] M. Du, S. Ding, H. Jia, Study on density peaks clustering based on k-nearest neighbors and principal component analysis, Knowledge-Based Systems 99 (2016) 135–145.

[25] Z. Li, Y. Tang, Comparative density peaks clustering, Expert Systems with Applications 95 (2018) 236–247.

[26] M. Parmar, D. Wang, X. Zhang, A.-H. Tan, C. Miao, J. Jiang, Y. Zhou, Redpc: A residual error-based density peak clustering algorithm, Neurocomputing 348 (2019) 82–96.

[27] J. Jiang, Y. Chen, D. Hao, K. Li, Dpc-lg: Density peaks clustering based on logistic distribution and gravitation, Physica A: Statistical Mechanics and its Applications 514 (2019) 25–35.

[28] S. Wang, D. Wang, C. Li, Y. Li, G. Ding, Clustering by fast search and find of density peaks with data field, Chinese Journal of Electronics 25 (2016) 397–402.

[29] G. Karypis, E.-H. Han, V. Kumar, Chameleon: Hierarchical clustering using dynamic modeling, Computer 32 (1999) 68–75.

[30] F. Ros, S. Guillaume, A hierarchical clustering algorithm and an improvement of the single linkage criterion to deal with noise, Expert Systems with Applications 128 (2019) 96–108.

[31] A. I. Maghsoodi, A. Kavian, M. Khalilzadeh, W. K. Brauers, Clus-mcda: A novel framework based on cluster analysis and multiple criteria decision theory in a supplier selection problem, Computers & Industrial Engineering 118 (2018) 409–422.

[32] B. F. Faustino, J. Moura-Pires, M. Y. Santos, G. Moreira, kd-snn: a metric data structure seconding the clustering of spatial data, in: International Conference on Computational Science and Its Applications, Springer, 2014, pp. 312–327.

[33] S. Kumari, S. Maurya, P. Goyal, S. S. Balasubramaniam, N. Goyal, Scalable parallel algorithms for shared nearest neighbor clustering, in: 2016 IEEE 23rd International Conference on High Performance Computing (HiPC), IEEE, 2016, pp. 72–81.

[34] Y. He, H. Tan, W. Luo, S. Feng, J. Fan, Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data, Frontiers of Computer Science 8 (2014) 83–99.

[35] S. Guha, R. Rastogi, K. Shim, Cure: an efficient clustering algorithm for large databases, Information Systems 26 (2001) 35 – 58.

[36] F. Ros, S. Guillaume, Dides: a fast and effective sampling for clustering algorithm, Knowledge and information systems 50 (2017) 543–568.

[37] P. K. Agarwal, S. Har-Peled, K. R. Varadarajan, et al., Geometric approximation via coresets, Combinatorial and computational geometry 52 (2005) 1–30.

[38] F. Ros, S. Guillaume, Protras: A probabilistic traversing sampling algorithm, Expert Systems with Applications 105 (2018) 65–76.

[39] F. Ros, S. Guillaume, A progressive sampling framework for clustering, Neurocomputing 450 (2021) 48–60.

[40] D. Hand, P. Christen, A note on using the f-measure for evaluating record linkage algorithms, Statistics and Computing 28 (2018) 539–547.

[41] S. Romano, J. Bailey, V. Nguyen, K. Verspoor, Standardized mutual information for clustering comparisons: one step further in adjustment for chance, in: International Conference on Machine Learning, PMLR, 2014, pp. 1143–1151.

[42] J. Hämäläinen, S. Jauhiainen, T. Kärkkäinen, Comparison of internal clustering validation indices for prototype-based clustering, Algorithms 10 (2017) 105.

[43] I. Kärkkäinen, P. Fränti, Dynamic local search algorithm for the clustering problem, Technical Report A-2002-6, Department of Computer Science, University of Joensuu, Joensuu, Finland, 2002.

[44] L. Fu, E. Medico, Flame, a novel fuzzy clustering method for the analysis of dna microarray data, BMC bioinformatics 8 (2007) 3.

[45] A. Jain, M. Law, Data Clustering: A User's Dilemma, in: Proceedings of the First international conference on Pattern Recognition and Machine Intelligence, 2005, pp. 1–10.

[46] P. Fränti, O. Virmajoki, Iterative shrinking method for clustering problems, Pattern Recognition 39 (2006) 761–765.

[47] I. Kärkkäinen, P. Fränti, Gradual model generator for single-pass clustering, Pattern Recognition 40 (2007) 784–795.

[48] P. Fränti, O. Virmajoki, V. Hautamäki, Fast agglomerative clustering using a k-nearest neighbor graph, IEEE Trans. on Pattern Analysis and Machine Intelligence 28 (2006) 1875–1881.

[49] W. Qiu, H. Joe, Generation of random clusters with specified degree of separation, Journal of Classification 23 (2006) 315–334.

[50] W. . J. Qiu, Harry, Separation index and partial membership for clustering, Computational Statistics & Data Analysis 50 (2006) 585–603.

[51] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework., Journal of Multiple-Valued Logic & Soft Computing 17 (2011).