Expert Systems With Applications 210 (2022) 118316

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Path-scan: A novel clustering algorithm based on core points and connexity

Frédéric Ros [a,*], Serge Guillaume [b], Rabia Riad [c]

[a] Laboratory PRISME, Orléans University, France
[b] ITAP, Irstea, Montpellier SupAgro, Univ Montpellier, Montpellier, France
[c] ERMAM, FPO - Ibn Zohr University, Morocco

ARTICLE INFO

Keywords:
Clustering
Natural cluster
Distance
Density
Neighbors

ABSTRACT

A new clustering algorithm *Path-scan* aiming at discovering natural partitions is proposed. It is based on the idea that a (k, ε) coreset of the original data base represented by core and support patterns can be path-connected via a density differential approach. The *Path-scan* algorithm is structured in two main parts producing a connectivity matrix where partitions can be extracted at different levels of granularity. The first one aims to identify and select core and support points while the second one extracts connected components of core points and clusters with the help of support points. A simulation experiment based on synthetic and real world data sets was conducted to show the effectiveness of the proposed method.

1. Introduction

Cluster analysis aims at identifying groups of similar objects and therefore helps to discover the distribution of patterns and interesting correlations in large data sets. The problem to be solve consists in separating a data set into different groups by optimizing a clustering criterion. This topic has been widely studied for many years in various fields (engineering, business, social sciences…) and existing clustering algorithms are very diverse. They can produce different solutions depending on the clustering criterion considered and on the choice of the input parameters involved.

The current expectations for new proposals can be grouped in two categories. The first one concerns the dimension and volume of 21st century databases involving the curse of dimensionality and algorithm scalability. The second one concerns the fact that most clustering algorithms have encountered challenges including low clustering accuracy, unequal clustering efficiency of different data sets and sensitive parameter dependence. Real-world data sets are hierarchically structured in nature but the identification of data structure without *a priori* information, such as the number of clusters, remains difficult (Ezugwu, et al., 2022).

This paper addresses this problem of knowledge discovery. For this expectation, sophisticated automatic clustering algorithms are necessary because of their flexibility and effectiveness. They have to be sufficiently self-tuning and adaptive for the result to be acceptable, whatever the input parameters, or put differently, for the same set of parameters to be able to manage various data sets.

A variety of clustering methods such as *hierarchical approaches* (Murtagh & Contreras, 2017), *Dbscan*, Ester, et al. (1996) and their variants, *Spectral Clustering* (Dhillon, Guan, & Kulis, 2004), *density peak approaches* (Rodriguez & Laio, 2014), *munec* (Ros & Guillaume, 2019b) has been proposed over the years to address this problem. They include numerous intelligent mechanisms, heuristics hybridizing density and distance notions under different technical assumptions. However, there are sometimes difficult to tune and generally do not take sufficiently into account the complexity and variability of data structure; in other words, they can work efficiently under a specific data configuration are not generic enough to cover real-world situations well. As a consequence, such methods may fail to discover the true clusters in a data set that it is not satisfactory.

The hierarchical clustering approaches (Vijaya & Bateja, 2017) tend to produce clusters of high quality, but they lose out to other methods in terms of performance and scalability. There are generally computationally expensive in time and memory, especially for large scale problems, termination criteria are ambiguous as well as the selection of appropriate metrics.

An efficient clustering algorithm by fast search and find of density peaks (DPC) (Rodriguez & Laio, 2014) was recently proposed and attracted much attention from researchers. 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. Although the 2D-plot $\rho - \delta$ provided by DPC can help users choose cluster center manually, it is still hard to distinguish the true cluster centers from all the points, especially when handling clusters with non-uniform densities and scales; In addition, DPC assigns non-center points according to their nearest neighbor with higher density, which causes a

* Corresponding author.
E-mail addresses: frederic.ros@univ-orleans.fr (F. Ros), serge.guillaume@irstea.fr (S. Guillaume), r.riad@uiz.ac.ma (R. Riad).

https://doi.org/10.1016/j.eswa.2022.118316
Received 24 June 2021; Received in revised form 12 July 2022; Accepted 27 July 2022
Available online 6 August 2022

'chain reaction' where one incorrect assignment of the highest density point could effect a whole region of points around it. There are still some complex shapes that the method cannot recognize because each density peak defined in this method is still a type of centroid, although not the centroid of a sphere.

Dbscan algorithm (Ester, et al., 1996) is a popular density-based clustering algorithm able to handle complex shapes and very suitable for the problem of discovery. In *Dbscan*, an object is regarded as the core point when the number of its neighbors MinPts in an ϵ neighborhood is larger than the predefined minimum number. *Dbscan* is able to separate the dense region from the sparse region by connecting the core points and detect clusters of any shape. Its performance is however greatly affected by its parameters, and choosing the appropriate value for them can be non-trivial. This problem can be even worse when handling clusters with varied densities as the algorithm relies on a global density threshold to find core (high-density) points. *Optics* (Ankerst, Breunig, Kriegel, & Sander, 1999) addresses somewhere one issue of *Dbscan* for handling clusters with heterogeneous densities but its clustering results still depend on ϵ and MinPts, to a certain extent like *Dbscan*. Other algorithms based on graph such as LOF-MST (Breunig, Kriegel, Ng, & Sander, 2000) and its variants including spectral techniques (Von Luxburg, 2007) can also discover clusters with arbitrary shapes, but they have to set parameters without any prior knowledge (Afzalan & Jazizadeh, 2019) and handling noise sensitivity still remains a tricky point.

Dbscan algorithm is still considered as the milestone of density-based clustering algorithms, and DPC one of the most attractive for researchers in the clustering field. In order to address above dilemma, various improved variants of these algorithms such as *Rnn-Dbscan* (Bryant & Cios, 2017), *Dcore* (Geng, et al., 2018), *RECOME* (Chen, et al., 2018), *Recon-Dbscan* (Zhu, Ting, & Carman, 2016), *Snn-radius* (Ertöz, Steinbach, & Kumar, 2003) and proposals in Ros, Guillaume, El Hajji, and Riad (2020), Xie and Jiang (2018), Xie, Jiang, and Ding (2017) have been proposed.

In the same perspective, a new approach named Path-scan is proposed where the DNA is connectivity, aiming at delivering a hierarchy that makes good sense in the perspective of human cognition. It is based on the assumption that clusters are characterized by several core and support patterns (points). These patterns aim at summarizing and understanding the data while in our case representing the "heart" (core) of the clusters. They generally represent interior points, can have a higher density than their neighbors or simply be a good representative or sample of their neighbors. Because of generally being interior points the connectivity between them is more "distinguishable" as the cluster is considered without its borders. To find a connectivity path between an interior point of a cluster to an interior point of another cluster, it is necessary to go through the borders of the two clusters as well as between the borders that are subjected of irregular data organization, presence of noisy points etc.

Path-scan is made up of two main parts that produce a connectivity matrix where partitions can be deduced at different levels of granularity. The first one aims at identifying and selecting core and support points while the second one extracts connected components of core points and clusters with the help of support points. Dealing only with these core and support points makes it possible to account for the differential density, without being disturbed by boundary points. The shortest path between core points is obtained by considering density estimation in a local context, and especially by assessing density continuity using a differential approach. This process is adapted to tackle the recurrent problem of variations in density between clusters. Because of the locality of its approach, it can handle various data organizations differently to many methods based on a global density threshold. Furthermore, it is computationally low as it is based on core and support points.

The innovation can be summarized as follows:

- The strategy in two steps based on the idea of core and support points and how they are generated and investigated via connective paths
- The concept of differential density and the idea to re-estimate the local density via the path context allowing to better handle clusters with varied densities
- The techniques behind the corresponding algorithms that are fully automatic. Natural partitions can be provided at different levels without any tedious tuning.

In the following, the state of the art approaches related to our work are reviewed in Section 2. The whole algorithm is detailed in Section 3. In Section 4 the proposal is evaluated and compared to alternative approaches using several synthetic and real data sets that illustrate the diversity of situations a clustering algorithm has to cope with. Finally the main conclusions are summarized in Section 5.

2. Related work

This section only focus on the studies that are close to the proposal regarding the mechanisms involved (core points, sampling, density peaks) as well as the finality which aims at discovering natural partitions with different granularities.

2.1. Coresets and sampling approaches

The term coreset was coined in Agarwal, Har-Peled, Varadarajan, et al. (2005) and used to compute the smallest k balls that cover a set of input points, and then similar covering problems where coresets are called certificates (Agarwal, Procopiuc, & Varadarajan, 2002). Optimization is the most common motivation for constructing coresets, where the goal is to compute an optimal query that minimizes the cost. 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 data set, sometimes after suitable post-processing. Sampling and coresets have been applied to clustering (Mahajan, Nimbhorkar, & Varadarajan, 2009) as it aims at organizing, summarizing and finally understanding the data.

Recently, three algorithms DIDES (Ros & Guillaume, 2017), DENDIS (Ros & Guillaume, 2016) and Protras (Ros & Guillaume, 2018) for unsupervised sampling were proposed. They are easy to tune, scalable and yield a small size sample. They are based on the concept of combining density and distance. The idea behind these algorithms is the farthest first traversal that allows for runtime optimization. They yield a coreset and they are driven by a single user parameter.

2.2. Linkage-based approaches

The goal of these approaches is to select the final core of clusters that will not be merged in the last step father algorithm and that are likely to form a *good* partition, *i.e.*, with compact clusters well separated from each other. Hierarchical clustering (Murtagh & Contreras, 2017) is a good representative. In Hierarchical clustering, the data are indirectly represented by a dissimilarity matrix, which provides the pairwise comparison between different elements. In this family, the agglomerative linkage criterion combines the dissimilarities between items. Hierarchical algorithms are popular as they are very easy to understand and easy to apply. However, the standard versions rarely provide the best solution as they involve arbitrary decisions. In addition, their efficiencies are largely dependent on the linkage criterion and its weaknesses are related to time complexity and the absence of any actual objective function to define the stopping criterion.

Among the linkage approaches, some of them combine the notion of distance and neighborhood such as *Chameleon* (Karypis, Han, & Kumar, 1999) that was the pioneer and is still the basis of, or a source of inspiration for, recent developments.

One of the most important open questions remains the identification of the meaningful clustering levels in the hierarchical structure.

There are numerous papers dealing with the improvement of Hierarchical clustering but it is still a challenge. One example is the recently proposed *HierOpt* algorithm (Ros & Guillaume, 2019a).

2.3. Density-based approaches

These methods can be assimilated to linkage-based approaches but are inspired more by notions of density. *Dbscan* (Ester, et al., 1996; Schubert, Sander, Ester, Kriegel, & Xu, 2017) is a typical density clustering method. *Dbscan* proceeds by computing the empirical densities for each sample point and then designating points whose densities are above a threshold as core-points. The *Dbscan* algorithm presents unique and advanced features that are useful when detecting patterns of different shapes and sizes. *Dbscan* is a good candidate to find 'natural' clusters and their arrangement within the data space when they have a comparable density without any preliminary information about the groups existing in a data set. In spite of its practical features, the original *Dbscan* algorithm fails when the border objects of two clusters are relatively close. Using spherical ϵ neighborhoods by *Dbscan* is also problematic when there are clusters with varying densities. Many algorithms such as *Denclue* (Hinneburg & Gabriel, 2007), *Optics* (Ankerst et al., 1999) or more recently *Recon-Dbscan* (Zhu et al., 2016) have addressed the drawback of *Dbscan*.

Optics generates an augmented ordering of the data to identify clusters with large variations in density. It addresses somewhere one issue of *Dbscan* for handling clusters with heterogeneous densities but its clustering results still depend on ϵ and MinPts, to a certain extent like *Dbscan*.

The recently proposed *Rnn-Dbscan* (Bryant & Cios, 2017) algorithm has the great advantage that only one parameter needs to be specified. 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. It cannot however recognize a cluster of non-core objects with different densities, the core idea being the union of k-nearest neighbor and reverse nearest neighbor to expand the cluster. In any cases, the fixed distance parameter ϵ limits the ability of the algorithm to handle clusters with heterogeneous densities. Moreover, the process of cluster expansion requires heavy memory that makes it computationally inefficient.

The *CLUS-MCDA* (Maghsoodi, Kavian, Khalilzadeh, & Brauers, 2018) algorithm can locate clusters of arbitrary shape and is dependent 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 when the overall density is basically the same.

While *Dbscan* defines reachable points using two parameters, the radius ϵ and the minimum number of points in the corresponding volume, $Minpts$, *Recon-Dbscan* 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. It requires more tuning than *Dbscan*, which affects its usability. Among other optimization versions, *HDbscan* (Campello, Moulavi, & Sander, 2013; McInnes, Healy, & Astels, 2017) extends *Dbscan* by converting it into a hierarchical clustering algorithm, and then using a technique to extract a flat clustering based on the stability of clusters.

The Shared Nearest Neighbor algorithm, *Snn* (Jarvis & Patrick, 1973), as well as its variants (Ertöz et al., 2003), is a density based clustering algorithm working similarly to *Dbscan*, but induced by the nearest neighbors. Fixing a unique k is not enough: one wants to discriminate via a unique sharing while in data structure the sharing configuration varies. The shared neighborhood is no longer controlled that may lead to not relevant partitions. The concept has however been introduced in a novel algorithm (Liu, Wang, & Yu, 2018) improving DPC algorithms.

2.4. Other recent approaches

Among recent algorithms the Density Peaks Clustering algorithm (*DPC*) was proposed in 2014 (Rodriguez & Laio, 2014) 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. mei2021efficient

Improvements were recently proposed (Du, Ding, & Jia, 2016; Guo, et al., 2022; Jiang, Chen, Hao, & Li, 2019; Li & Tang, 2018; Parmar, et al., 2019; Xie, Gao, Xie, Liu, & Grant, 2016; Xie & Jiang, 2018; Xie et al., 2017; Xu, Ding, & Shi, 2018; Zhang, Miao, Tian, & Wang, 2022). Among them, there are research on novel density measures (Liu et al., 2018), novel strategies (Tong, Liu, & Gao, 2021; Yang, Cai, Yang, & Zhao, 2022; Yaohui, Zhengming, & Fang, 2017) to help DPC select cluster centers automatically. In Wang, Wang, Li, Li, and Ding (2016), the threshold distance d_c is now automatically set using the potential entropy of the data field from the original data set. In the *comparative density peaks* algorithm (Li & Tang, 2018), the idea is to consider 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.

In Xie, et al. (2016), the idea of this density-based algorithm is to compute the local density ρ_i of point i relative to its k-nearest neighbors for any size data set independent of the cutoff distance d_c. Within this process, the remaining points are assigned to the most probable clusters.

In Guo, et al. (2022), a graph-based strategy is proposed to estimate the connectivity information between local centers, allowing to better assign all local centers.

A common drawback of most of the methods discussed above is that they cannot provide a unified strategy to define the number of clusters, and thus may lead to poor performance with complex data organization.

The *Munec* algorithm (Ros & Guillaume, 2019b) is based on an iterative process that merges mutual nearest neighbors. Three heuristic conditions are introduced in order to discriminate nuanced situations. They are based on a combination of several notions such as distances between mutual neighbors, nearest neighbor group of higher size and local neighborhood density. The algorithm is driven by a single user parameter, u, that needs to be tuned which is not straightforward as its heuristic conditions are complex.

The *KdMutual* algorithm (Ros et al., 2020) 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. It is however not dedicated to a discovery process as it is based on the number of clusters as input.

Several density-based methods have been recently proposed to deal with large scale data. Among them, the *Mr-Dbscan* (He, Tan, Luo, Feng, & Fan, 2014) that can achieve an ideal load balance in a severely skewed data environment. The latter was extended to *Isb-Dbscan* (Lv, et al., 2016) by focusing on clustering non-core objects, which is undetermined when two core objects are equidistant from a non-core object.

The *Block-Dbscan* (Chen, et al., 2021) is a recent algorithm that consists in an approximate and grid-based clustering approach. This technique has the advantage of being able to handle large data sets, but it does not focus to the discovery process improvement.

Community discovery is not the topic of this paper but this field has analogies to the clustering one. Community detection algorithms are essentially based on graph theory while using pattern recognition techniques. Many promising algorithms (Jiang, Fang, Li, & Li, 2022; Li, et al., 2018; Lu, Liu, Zuo, & Li, 2021; Xu, Zhuang, Li, & Zhou, 2018)

Table 1
Notations.

Notations	Description
$D\ (n \times p)$	The multidimensional data set of size n and p dimensions to be processed.
$S\ (n' \times p)$	Reduced set of $D(n' \leq n)$
$P,\ P^+$	Sets of core and support points.
$d_{refglob}$	Global distance reference used to evaluate local density.
$M^{dp}\ (\|S\| \times \|S\|)$	Euclidean distance matrix of the S points.
$KNN(\|S\| \times \|S\|)$	Neighborhood matrix for the S points. The first entry in each row indicates the pattern under consideration, the second entry indicates the first nearest neighbor, the third entry indicates the second nearest neighbor, and so on.
k_{nn}	k nearest neighbors related to the set S.
$N^k(x)$	$N^k(x) = \{x_{(1)}, x_{(2)}, \ldots, x_{(k)}\}$, where $\{x_{(1)}, x_{(2)}, \ldots, x_{(n-1)}\}$ is the permutation of the elements of $X \setminus \{x\}$ such that $\|\|x_{(1)} - x\|\| \leq \|\|x_{(2)} - x\|\| \leq \ldots \leq \|\|x_{(n-1)} - x\|\|$
$N^{knn}(j)$	Subset of points representing the k_{nn} nearest neighbors of j.
$M^r\ (S \times S)$	Reachability matrix in the range $[0, 1]$ processed from M^{dp}.
$M^C\ (S \times S)$	Connexity matrix in the range $[0, 1]$.
$M^{diff}\ (S \times S)$	Density differential matrix in the range $[0, 1]$.
$M^B\ (S \times S)$	Binary matrix from M^C.

have been recently proposed for static and temporal networks. These advances could be benefit the clustering field under the condition to obtain an efficient graph from data. Specific techniques such as Epsilon-ball, kNN and many others can be considered as well as a similarity matrix. The latter can be viewed as the adjacency matrix of a fully connected, weighted graph, where the nodes correspond to data points and the edge between two nodes is weighted by their similarity.

3. Path-scan: method and algorithms

The rationale of the approach assumes that density peaks are good representatives of core clusters, but that methods based only on density peaks fail to identify complex shaped clusters. Neighboring approaches, *e.g. Dbscan*, are, on the other way, good at this task.

The algorithm is made up of two main parts that gives the essential elements to produce the final partition via dedicated mechanisms. The first one aims at identifying and selecting core and support points while the second one extracts connected components of core points and clusters with the help of support points. The different notations used are summarized in Table 1.

Core and support points represents a (k, ε) coreset of the original data base (example in Fig. 1. Dealing only with these core and support points makes it possible to account for the differential density, without being disturbed by boundary points. Their discrimination (cf. Alg. 2) is based on a specific filtering process where neighborhood concepts (k nearest and mutual neighbors) are investigated: the idea is that only a subset of core cluster representatives are required; the remaining ones are likely to jeopardize the data structure identification.

A first partition in connex components represented by a reachability matrix M^r is deduced. This partition is relevant in case of ideal situations (well-separated clusters, no noise...) as a given cluster refers to only one connex component. In reality, each connex component can represent several clusters as shown in Fig. 1 (on the left) and a finer analysis related to the level of linkage between the core points is needed to produce a partition (on the right).

The level of linkage involves a path search mechanism while considering a new metric based on the differential density between connected core or support points. The min–max ratio of the densities is taken as the measure of differential density. Local densities are re-estimated in the path context allowing clusters with varied densities to be better managed. The shortest path between each pair of core points is found by giving a set of ordered waypoints. It is evaluated via a mechanism considering the density differential within its waypoints and between two consecutive waypoints.

A connectivity graph formalized by a matrix (coefficients in the range $[0, 1]$) is produced and hierarchically explored to provide segmentation in clusters at different levels accomplishing a discovery task. As shown in Fig. 2 6 clusters are fully detected in a large range while other partitions for other levels. The whole hierarchical process is fully automated.

3.1. Part 1: Core and support point identification

The first part of the algorithm deals with core and support point identification. An overview is given in Algorithm 1.

A sampling algorithm is first used to clean the data and select representatives. In this work, this is achieved by the *ProTraS* algorithm (Ros & Guillaume, 2018). It is a recursive partitioning algorithm: at each step a new representative is added to the sample until the cost falls below a threshold. The new representative is chosen as the farthest from the one already selected in the group with the highest probability of cost reduction. This probability is computed according to the within group distance and to the representativeness of the sample item, assessed by the number of items in the whole set it represents. This algorithm is fully driven by a unique parameter: the sampling cost. The lower the cost, the more accurate the representation and the larger the sampling set. The cost value is empirically set at 0.05 (line 3) on the basis of the study of cost sensitivity done in Ros and Guillaume (2018).

Algorithm 1 Core and support point identification

1: Input: $D(n \times p)$
2: Output: P, P^+ {Sets of core and support points}
3: S_1= **ProTraS**$(D, cost = 0.05)$
4: S_2=**kmeans**$(D, |S_1|, S_1)$ {kmeans is run on the whole with a number of clusters the size of S_1 and with a S_1 seeding.}
5: S=AdjustCenters$(S2, D)$ {Each center computed by the kmeans is replaced by its nearest neighbor in D.}
6: k_{nn} = **GetNumberOfNN**(S) {Determine automatically the number of nearest neighbors to consider to ensure the $\sqrt{(n)}$ neighbor can be reached using a k_{nn} connectivity for all items in S}
7: $dens$=**LocalDensity**(S, D, r) {Local density estimation in a r radius hypervolume, r is the median of the average distances between the k_{nn} nearest neighbors in S.}
8: $[P, P^+]$ = **GetPoints**$(n, dens, k_{nn}, S)$ {Detailed in Algorithm 2.}

To ensure that the representatives are located at the center of their group (not the middle of clusters), a *k-means* is run with a number of clusters the size of the sample set yielded by *ProTraS* and the representatives used as initial centers instead of the classical random seeding (line 4). To complete this process, each computed center is replaced by its nearest neighbor in D (line 5).

The core and support point selection function is detailed in Algorithm 2. It is based upon the local density estimation and the neighborhood defined by the number of neighbors, k_{nn}, automatically set and yields two sets of items: P is the set of core points, P^+ is a set of support points that will also be used in the second part of the process. The first set is computed in lines 4–16, the second one in lines 17–19.

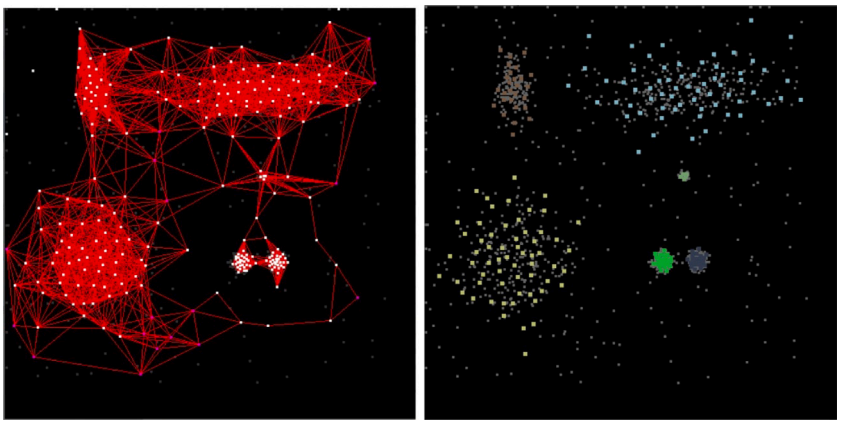

Fig. 1. Illustration of the process: on the left the reachable prototypes (core and support points) obtained via neighborhood concepts and on the right the prototype partition for a level of linkage equal t=0.6.

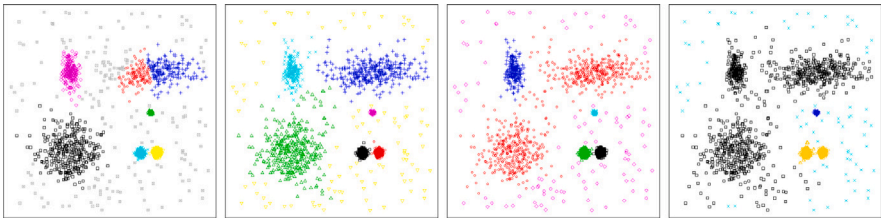

Fig. 2. Hierarchical process from decreasing threshold levels t of linkage $t = 0.7$ (7 clusters), $t = [0.6, 0.2]$ (6 clusters), $t = 0.1$ (5 clusters), $t = [0.05, 0.01]$ (4 clusters) from left to right. Before $t = 0.7$, the partitions found are not eligible as the clusters are not representative enough $(\%M) \leq 0.5$).

Algorithm 2 *GetPoints*

1: Inputs: n, dens, k_{nn}, S
2: Outputs: P, P^+ {Sets of core and support points}
3: $P = P^+ = \emptyset$, $\varepsilon = n/100$ {$n/100$ is the minimum size for a cluster}
4: **for all** $(i \in S)$ **do**
5: **if** $(dens[i] \geq 2\varepsilon)$ **then**
6: $P = P \cup \{i\}$
7: Continue;
8: **end if**
9: $N(i) = \{j | j \in N^{k_{nn}}(i)$ **&** $i \in N^{k_{nn}}(j)\}$ {$N(i)$ is the set of mutual neighbors of i among its k_{nn} nearest neighbors}
10: **if** $(N(i) == \emptyset$ & $dens[i] \geq \varepsilon)$ **then**
11: $P = P \cup \{i\}$
12: **else**
13: u=$\underset{j \in N(i)}{argmax}(dens[j])$
14: $P = P \cup \{u\}$
15: **end if**
16: **end for**
17: **for all** $(l \in S \setminus P)$ **do**
18: $P^+ = P^+ \cup \{j | j \in N^{2k_{nn}}(l)$ & $l \in N^{2k_{nn}}(j)$ & $dens[j] > \epsilon/2\}$
19: **end for**
20: **return** P, P^+

For each sample item, one point is added to the set P. This does not mean that P is as large as S: the same item can be chosen at various iterations. There are three levels of selection for each item:

1. the sample point is dense enough to be added (line 5). The threshold is twice the minimum size for a cluster, set at $0.01n$;
2. the densest item in its mutual neighborhood is added (line 13) ;
3. the sample point has no mutual neighbors but it is dense enough to belong to a small cluster (line 10).

This combination of density and neighborhood criteria allows the management of various densities in the input space without user parameters. The first condition is based only on density. The assumption is that when the high density threshold, $2\varepsilon = n/50$ is reached, even if the density is computed according to a global distance, the point is a core point. The second and third ones combine density and mutual neighborhood, defined by the number of neighbors automatically computed, k_{nn}. When there exist mutual neighbors, the densest item is selected, whatever its density. This makes it possible to manage different densities without a distance threshold. Otherwise, when the mutual neighborhood is empty, the points with a moderate density, $\varepsilon = n/100$ are kept. The goal is not to skip isolated clusters.

Once the core points have been selected, the second loop, lines 17–19, identifies the set of support points, P^+. They do not meet the conditions of density and neighborhood to be in P but they are likely to help in keeping density peaks connected. These points are expected to be dense enough and located in a mutual wider neighborhood, defined by twice the number of neighbors, $2k_{nn}$.

The output of Algorithm 2, the two sets P and P^+, is the basis of the second part of the whole process.

Nearest neighbors determination:. The idea is to define 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. Let $f^k(x)$ be the search function for the k nearest neighbors of x:

$$N^k(x) = f^k(x) \tag{1}$$

The number of iterations of $f^k(x)$ is such that $\forall y \in \Omega_k(x) \quad N^k(y) \subset \Omega_k(x)$ where $\Omega_k(x) = f^k \circ \ldots \circ f^k(x)$. Then, y is reachable from x, if $y \in \Omega_k(x)$. The process consists in testing a subset of $N = 1000$ patterns and for each pattern determining the number of neighbors needed to reach its $v_{max} = \sqrt{n}$ neighbor ($y \in N^{\sqrt{n}}(x)$) and then selecting the minimum value so that y is reachable from x for a majority of the tested patterns ($\geq 95\%$).

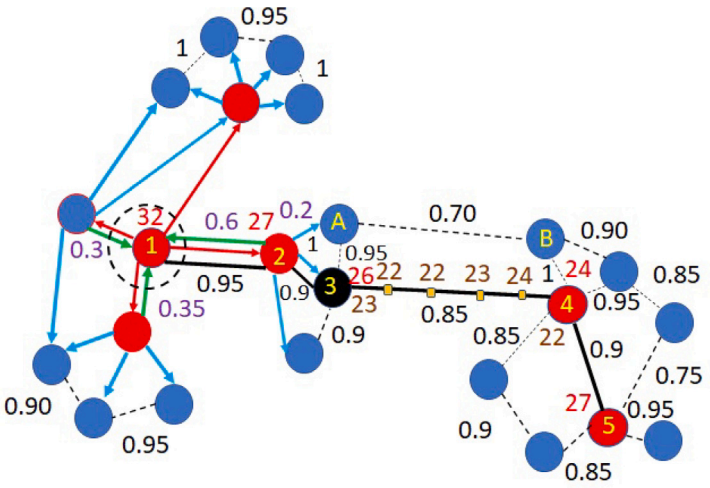

Fig. 3. Toy example for connected component extraction : The optimal path between the core points 1 and 5 (1→ 2 → 3 → 4 → 5) is found using the Dijkstra algorithm where the metric is not the Euclidean distance but the differential density. The figures in black and purple are the level of connectivity (between 0 an 1) "inverse" of the differential density.

Local density estimation:. In order to estimate the local density of each sample item, the distance between the k_{nn} nearest neighbors in S is computed. It is used as the radius of the hypervolume for the density estimation in the whole data set, D. This automatic setting ensures that the data characteristics are taken into account and avoids the *a priori* setting of such a sensitive parameter.

$$dens(x) = \sum_{i=1}^{n} e^{-\frac{\|x_i - x\|^2}{r}} \tag{2}$$

3.2. Part 2: Connected component extraction

3.2.1. Main ideas

The extraction of connected components is done in two main steps mainly described in algorithm 5. For a better understanding, the process is explained via a toy example shown in Fig. 3.

Two types of points are represented in Fig. 3, which represents a single connex component: set P (in red), set P^+ (in blue and black). The connectivity matrix M^c is computed connex component by connex component. From each of them, sub-connex components are processed and their aggregation enables clusters to be found via filtering and fusion operations. The connected components are previously determined from the reachability matrix M^r (*cf.* Algorithm 3).

Definition 1. Two points i and j are reachable ($M^r_{ij} \neq \infty$) if they belong to ($P \cup P^+$), are neighboring mutuals (neighborhood size = 2 k_{nn}) and with non-negligible densities (*cf.* Algorithm 3: lines 5 and 7).

All the points of Fig. 3 which are connected by an arrow or a line are reachable, and all belong to the set ($P \cup P^+$). There is always a path allowing to reach any point from any other point in the sense of reachability: 2 points i and j not belonging to the same connex component are not reachable and therefore have a zero connectivity $M^c_{ij} = 0$. Fig. 4 provides an example of reachability and shows the different sets of points involved in the process.

In our clustering strategy only the level of connection between points of P in each connex component is useful. As the points P^+ have a supporting role to connect them, the connectivity of the reachable points of ($P \cup P^+$) also needs to be computed. The algorithm is divided into 2 steps: In the 1st step (*cf.* 3.2.2), the connectivity is computed for all the points which are reachable (linked by an arrow or a line in Fig. 3). In a second step (*cf.* 3.2.3), these connexities serve as support to process the connectivity of the points that are not reachable (restricted to the points of P).

Algorithm 3 *GetReachability* (Reachability matrix M^r)

1: Inputs: n, P, P^+, k_{nn}, M^{dp}, dens
2: Outputs: M^r
3: ϵ=$n/100$
4: **for all** (i,j $\in (P \cup P^+) \times (P \cup P^+)$) **do**
5: **if** (!($i \in (P \cup P^+)$ & $j \in (P \cup P^+)$) || ($dens[i] \leq \frac{\epsilon}{2}$ || $dens[j] \leq \frac{\epsilon}{2}$)) **then**
6: $M^r_{ij} = M^r_{ji} = \infty$
7: **else if** ($j \in N^{2k_{nn}}(i)$ & $i \in N^{2k_{nn}}(j)$) **then**
8: M^r_{ij}=M^r_{ji}=M^{dp}_{ij}
9: **else**
10: $M^r_{ij} = M^r_{ji} = \infty$
11: **end if**
12: **end for**

Fig. 4. In gray: D points, in blue $S \setminus (P \cup P^+)$, in white P, in pink: P^+, visible link between i, $j \Leftrightarrow M^r_{ij} ! = \infty$. There are a big connex component and 6 isolated P points.

3.2.2. Connexity between reachable points

The process of calculating the connectivity between two reachable points is illustrated between points 3 and 4 and formalized in Algorithm 5 lines 6 to 12. The 1st step consists in determining a reference Euclidean distance relating to these 2 points which is the minimum

of their own distance references. This distance will be used to estimate the local probability densities at points 3 and 4 via points from the original data D. The computational time is highly optimized as the set S serves as intermediate to select only the influential points. If the Euclidean distance between the contact points (here 3 and 4) is greater than this reference distance, the corresponding segment (3–4) will be linearly subdivided in order to estimate the densities in the segment (*cf.* Algorithm 5 line 10) via a virtual interpolated point that serves to check a real continuity of the densities inside the segment. The subdivision is illustrated between point 3 and 4. The calculation of a reference distance is formalized in algorithm 4 and illustrated for point 1 (which is circled for clarity's sake) in the toy example where it is assumed that k_{nn}=4. Point 1 has 4 neighbors but only 3 of the 4 neighbors are reciprocal neighbors as they admit point 1 among their own 4 nearest neighbors. The red arrows connect the 4 neighbors of the test point (1).

Algorithm 4 *Getdist*: reference distance calculation

1: Inputs: path, M^{dp}, KNN, k_{nn}, S, $d_{refglob}$
2: Outputs: d_{ref}
3: $d_{refloc} = \infty$, count = 0
4: **for all** ($i \in$ path) **do**
5: $d_{path} = 0$
6: **for all** ($j \in N^{knn}(i)$) **do**
7: **if** (i $\in N^{knn}(j)$) **then**
8: v=KNN[j][1] {v is the 1 nearest neighbor of j in S}
9: $d_{path} = d_{path} + M^{dp}_{j,v}$
10: count = count + 1
11: **end if**
12: **end for**
13: **if** (count != 0) **then**
14: $d_{refloc} = min(d_{refloc}, \frac{d_{path}}{count})$
15: **end if**
16: **end for**
17: **if** (d_{refloc} != ∞) **then**
18: return d_{refloc}
19: **else**
20: return $d_{refglob}$
21: **end if**

The dark blue arrows show the 4 neighbors of the point tested (1) and the links are in green. Thus for point 1, the average of the distances to 1 nearest neighbor of the considered neighbors is taken: (0.3+0.35+0.2)/3.; 0.3, 0.35 and 0.2 are the Euclidean distances. dref(3,4)=min(dref(3),dref(4))=0.22 is therefore used at 2 levels: (i) Determine virtual points between points 3 and 4: $d_{eucl}(2,3)$=1.4 then (1.4/0.2) gives the number of points which covers the segment regarding the reference distance (*cf.* algorithm 5 line 10). (ii) Calculate the densities of all virtual points including points 3 and 4 (in orange). The ratio of the min density on the maximum density deduced here is 0.85 (*cf.* algorithm 5 line 11). In synthesis, the connexity is calculated for all reachable points. Connectivity M^c_{ij} is a score between 0 and 1 and the values are in black in the figure for some connections. The difference in the sense of density M^{diff}_{ij} is 1 - M^c_{ij}. For example, for points 3 and 4: $M^c_{3,4}$=0.85 so $M^{diff}_{3,4}$=0.15. For non-reachable points the Euclidean distance is infinite, M^c_{ij}=0 and therefore the distance M^{diff}_{ij}=1 is maximum.

Definition 2. Connexity level: two points i and j are not reachable and not connex, then their connexity score is M^c_{ij}=0. If they are reachable M^c_{ij} = 1 - M^{diff}_{ij}, where M^{diff}_{ij} is the density differential between i and j that is locally estimated via a reference distance (*cf.* Algorithm 4).

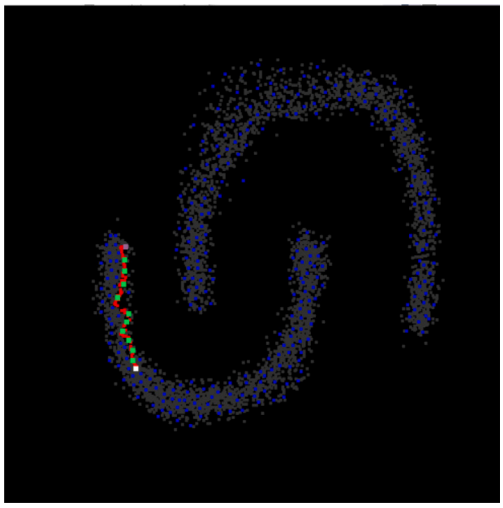

Fig. 5. A Dijkstra path: the starting point is in white, The end point in magenta, ($P \cup P^+$) in green and the virtual points in red.

3.2.3. Path search between non reachable core points

Once the connexities have been established for all reachable points, any point of a connected set can be linked with any other point via a path of reachable points. In Fig. 3, the connexities between reachable points are in black as for example $M^c_{1,2}$=0.95 and $M^c_{3,4}$=0.85. The only feature that interest us are the connections between the points P (*cf.* algorithm 5 line 15) of the same connex component such as the ones in red in Fig. 3. By thresholding via the Euclidean distance, the number of connexity calculations (*cf.* algorithm 5 line 16) can be reduced without affecting the algorithm regarding the meaning of k_{nn}. There are different possible paths, as each point of ($P \cup P^+$) may have several reachable points. The question now is how to select a "good" path. It is done on the assumption that the points of the same cluster admit a lower density differential than between the points of 2 different clusters. This assumption makes it possible to objectify the search for the optimal path (as for example between 1 and 5). It is done by minimizing the travel distance between those 2 points under a *density-based distance* and not a *Euclidean one*. The search for the optimal path is established from the Dijkstra algorithm which has as entry M^{diff} (*cf.* algorithm 5 line 17). The values for all the reachable points (Section 3.2.2 and 5 lines 6 to 12) have been previously calculated. For example, $M^{diff}_{1,2}$=0.05 and $M^{diff}_{3,4}$=0.15. The Dijkstra path found between point 1 and 5 is (1,2,3,4,5). All these points are necessarily linked by successive reachable points (1→ 2 → 3 → 4 → 5).

Definition 3. Path: there is a path between i and j if there exists a subset of successive points that are mutually reachable. The best path is the one corresponding to the Dijkstra algorithm optimization where the metric is based on density differentiation (*cf.* Algorithm 5)

The links are in black dotted lines (or in green) in Fig. 3. Point 3 in black is the only to be in P^+. The path distance is 0.35 ⇒ (1-0.95)+(1-0.95)+(1-0.85)+(1-0.9). Note that according to the Dijkstra algorithm, the optimal path does not pass through point A even if the density-based distance between point 2 and A is 0. This passage is penalized by a distance of 0.3 between point A and B. Fig. 5 shows an example of the path found by the Dijkstra algorithm. The starting and end points are close to the border. The path is roughly formed by border points having relatively similar densities.

3.2.4. Connexity evaluation for a path

All the ingredients are available to evaluate the level of connexity.

Algorithm 5 *GetConnexity*: Connexity matrix

1: Inputs: P, P^+, k_{nn},KNN, M^r, $d_{refglob}$
2: Outputs: M^c
3: C_c=**GetConnex**(M^r) {C_c represent the connex components according to M^r}
4: $M^c = \infty$, $M^{diff} = \infty$
5: **for all** ($k \in C_c$) **do**
6: **for all** ($i, j \in ((P \cup P^+) \times (P \cup P^+)) \cap C_c[k]$) **do**
7: **if** (M^r_{ij} !=∞ & i!=j)) **then**
8: $path = \{i, j\}$
9: d_{ref}=**Getdist**(path,KNN,k_{nn},$(P \cup P^+)$,$d_{refglob}$)
10: $subpath_{ij}$=**Divlin**(i, j, d_{ref})
11: $M^c_{ij} = M^c_{ji} = \frac{\underset{t \in subpath_{ij}}{argmin}(dens(t))}{\underset{t \in subpath_{ij}}{argmax}(dens(t))}$
12: $M^{diff}_{ji} = M^{diff}_{ij} = 1 - M^c_{ij}$
13: **end if**
14: **end for**
15: **for all** ($i, j \in (P \times P) \cap C_c[k]$) **do**
16: **if** ($M^c_{ij} == \infty$ & $d_{ij} \leq 2 \times max(d_{i,KNN(i,2k_{nn})}, d_{j,KNN(j,2k_{nn})})$ **then**
17: path=**AlgDijkstra**(i, j, M^{diff},$(P \cup P^+)$)
18: d_{ref}=**Getdist**(path,KNN,k_{nn},$(P \cup P^+)$,$d_{refglob}$)
19: $M^c_{ij} = \mathbf{min}(\underset{t \in [1,|path|-1]}{argmin}(M^c_{Path_t\ t+1}), \frac{\underset{t \in path_{ij)}}{argmin}(dens(t))}{\underset{t \in path_{ij}}{argmax}(dens(t))})$
20: **end if**
21: **end for**
22: **end for**
23: return M^c

Definition 4. Connexity level on a path: when a path concerns reachable points the value is directly processable (Algorithm 5 line 19) via the density differential. On the contrary, the path is formed with successive reachable points (*cf.* Section 3.2.3). The connexity value refers to local and global density differentiation (Algorithm 5 line 19).

Let us now see how to calculate the connectivity taking the path 1–5 as an example. $M^c_{1,5}$=min(s, s') where s is a score based on the reachable points of its path and s' is based on the local density of each point *reestimated* in the context of the path, *i.e.* its reference distance (*cf.* algorithm 5 line 19)

$$s = \min(M^c_{1,2} \ldots M^c_{4,5}) / \max(M^c_{1,2} \ldots M^c_{4,5}) = 0.85/1 = 0.85. \quad (3)$$

$$s' = \min(dens(1), \ldots, dens(5)) / \max(dens(1), \ldots, dens(5)) \quad (4)$$

$$dref_{1,5} = \min(dref(1), \ldots, dred(5)) = 24/32 = 0.75 \quad (5)$$

where $dens(i)$ is the local density calculated using $dref_{1,5}$ as the reference distance. It is deduced that $M^c_{1,5}$=0.75. Note that the densities for example at points 3 and 4 are not identical when calculated from the path 3, 4 or as part of the Dijkstra path [1, 5]. In red, the densities in the path given by the Dijkstra algorithm and in orange when calculating the local connectivity between point 3 and 4.

3.3. Part 3: From connected components to cluster partitions

This part consists in exploring M^c to provide cluster partitions in a hierarchical way according to the level of connectivity t (main loop from line 6 to 16 of algorithm 6) providing M^B (a binary matrix) from which the resulting partition is processed. The primary output for the user is how to evolve the cluster number. The presence of a plateau highlights the presence of consistent partitions. A deeper exploration provides useful information related to the cluster proximity. Logically, clusters having the smallest density differential will appear before the others while there is no ordering when clusters have different densities.

Two mechanisms are involved in the discovery process: the first one checks the consistency of the connected elements regarding their representativeness while the second extracts potential clusters via filtering and fusion operations. The latter improves the discovery process by anticipating at a step t the formation of a cluster (from connected elements) at step t+1. At each step, M^B is obtained from M^c by a single thresholding from which C_A (*cf.* algorithm 6 line 7) can be deduced using a single recursive operation (*cf.* Algorithm 6 line 8).

$$C_A = C_{A1} \cup C_{A2} \ldots \cup C_{Ak} \quad (6)$$

Fig. 6 shows candidate clusters coming from this operation where each connected pair (i, j) is linked with a red line.

Definition 5. Candidate cluster: i and j belong to the same cluster at a t level if $M^B_{i,j}$=1 or if there is a series of l intermediate points $k_1, \ldots, k_l$ such that M^B_{i,k_1}=1, $M^B_{k_t,k_{t+1}}$=1 (t ∈ [1, l]) and $M^B_{k_n,j}$=1

To avoid a lack of representativeness possibly at a high level thresholding, C_A is conditionally explored (line 9). $|C_{Al}|$ is the sum of points coming from D attached to its representative in $(P \cup P^+)$. The goal in this stage is twofold: to select the k' final core clusters and to obtain the final partition. The clusters are sorted in decreasing order via a criterion Q involving the minimum distance to the nearest cluster of higher cardinality. Clusters under a minimum cardinality ($\leq n/50$) are assigned a score of zero. z_i, $z_i \subset C_A$ such as $Q[z_{(1)}] \geq Q[z_{(2)}] \geq \cdots Q[z_{(k)}]$

$$Q[i] = |c_{Ai}|\ d_{near+}[i] \quad (7)$$

where $d_{near+}[i]$ is the minimum single link distance between cluster i to the nearest cluster of higher cardinality. k' core clusters are then selected among k (*cf.* Algorithm 6 line 10) using a single decision rule depending on one internal parameter α aiming at identifying a large difference in Q score between two candidate clusters; α is fixed at 0.1 in our algorithm but remains tunable by the investigator if needed:

$$k' = \{r |\ \frac{Q[z_{(r+1)}]}{Q[z_{(r)}]} \leq \alpha\} \quad (8)$$

The remaining clusters are attached to the selected ones using the Q criterion in a single iterative process. A noise filtering operation may be added on the basis of the Q criterion. It is not done in the current version of the algorithm that classifies all the patterns if the discovered clusters are eligible (*cf.* Algorithm 6 line 9). It should be underlined that this partition operation is computationally very low as it operates from the S set which is a subset of D.

Once an eligible partition is found in the prototype space, one can retrieve the final partition in the original space. Each prototype concerned by the discovered partition represents a sub-set of original patterns (see Algorithm 1) that are directly associated. The others are considered as noisy patterns.

4. Experiments and results

This section aims at demonstrating that the algorithm is efficient for a large variety of data structures involving cluster complexities. Its efficiency was evaluated at two levels: the capacity of discovering natural clusters if they exist and the accuracy of the discovered partition. In all experiments, the algorithm was run with the same setting to check whether it is able to identify the data structure of various data sets without any tuning, the goal being to empirically demonstrate its self-tuning power. When the ground truth is available the accuracy is was measured both with the Mutual Information Index (MI) (Romano, Bailey, Nguyen, & Verspoor, 2014) and F-score (Sokolova, Japkowicz, & Szpakowicz, 2006) that differently compare the expected partition with the one given by the algorithm. The scores have were calculated by skipping the noise both for the ground truth and the result. In

Algorithm 6 *GetPartition:* Hierarchical partition distribution

Input: S, M^C, l, N_{test}
Ouput: N_p
$Max_l = 0.9$, $Min_l = 0.05$ $\epsilon = n/100$
$\Delta = (Max_l - Min_l)/N_{test}$
level = Max_l
for (i :1 to N_{test}) **do**
  M^B = **GetBinary**(M^C, level)
  C_A = **GetBinaryGraph**(M^B)
  if $\left(\frac{\sum_{l=1}^{k}(|C_{Al}| \geq \epsilon)}{\sum_{l=1}^{k}(|C_{Al}|)} \geq \frac{1}{2} \right)$ **then**
    G = **Findpartition**(ϵ, C_A)
    $N_p[i] = |G|$
  else
    $N_p[i] = 0$
  end if
  level -= Δ
end for
return

Fig. 6. At a given threshold, there are $k = 6$ potential clusters, one having a very low cardinality. P Points of each clusters are in white and connected by red lines, P^+ points are in green.

real life the reference is unknown and clustering algorithms are used to analyze the data. External validation provides more relevant and meaningful measures for testing on simulated data, but the hypothesis of having access to the desired solution is obviously not realistic in practical implementations. In the case where there is no reference, in order to characterize the partition, only internal validation indices (Hämäläinen, Jauhiainen, & Kärkkäinen, 2017) are considered.

Different kinds of experiments are proposed. The first deals with 2-dimensional data sets as they allow for a human assessment of what partitions are acceptable. The second illustrates the behavior of the proposal in various difficulties of higher dimensions involving Gaussian and non Gaussian data structures.

The third deals with real data where no ground truth is available. The pre-processing includes a $\{\mu, \sigma\}$ standardization step and, for the data sets with more than four thousand items, a sampling (Ros & Guillaume, 2018) algorithm is applied in order to store the distance matrix in memory and to complete the tests in a reasonable amount of time.

All simulations were carried out on a standard personal computer (DELL Precision 5530) with the C programming language. The original sources of several competitive algorithms were integrated when provided by the authors such as the *Rnn-Dbscan* and *Block-Dbscan* algorithms.

4.1. Competitive algorithms

The proposal was compared to 15 competitive algorithms, described in Table 2 with their free parameters. Some of them require the number of clusters as input. They are rather recent clustering algorithms except for $kmeans^{++}$ (*cf.* Table 2). These algorithms were run by varying the number of clusters from 2 to 20. Other algorithms such as dbscan and it variants (*cf.* Table 2 in bold) are more interesting for the discovery aspect as they do not require the number of clusters as input.

4.2. Experiments with synthetic data sets

4.2.1. Benchmark in 2D

A wide range of data-sets (12) is used as benchmarks in this section. The acceptable partitions given via the ground truth are displayed in Fig. 7. The data may include some variations in the clusters. The main sources of variation with their associated code are given in Table 3. They concern the shape and the size of the clusters, their level of separation, the variation of density either between or within clusters and the amount of noise. The twelve data-sets and their classification are described in Table 4.

Some were proposed in the published literature[1,2,3], or are from the data clustering repository of the computing school of Eastern Finland University.[4] while others come from the GitHub repository[5] . These data sets are usually considered for testing new clustering algorithms. To complete the diversity, homemade data were added[6] They represent additional configurations where clusters are different in size, shape, density, amount of noise and degree of separation.

4.2.2. Data sets of higher dimension

Three kinds of experiments were carried out to assess the behavior of the algorithm in higher-dimensional spaces. The idea is to illustrate the behavior with clusters of variable configurations (density peak, pseudo uniform data, more or less separated, different densities). In the first one, a series of high-dimensional data sets with Gaussian clusters was tested. The first sub series (from "dimsets low"(Kärkkäinen & Fränti, 2007)) is of moderate dimensions and the data patterns are partitioned into 9 Gaussian clusters in $d = \{2, 5, 10, 15\}$ (Fig. 8 at left). The second (from "dimset high" (Fränti, Virmajoki, & Hautamäki, 2006)) contains high-dimensional data sets with 16 Gaussian clusters in $d = \{32, 64, 128, 256\}$ (Fig. 8 at middle and right). Each cluster in a set has the same number of points, and the number of points in each cluster increases linearly as dimensionality increases. These sets were proposed in Fränti et al. (2006) and clusters are rather well separated. They are denominated $G_1, \ldots, G_8$.

In the second one, 8 data sets, $d = \{6, 9, 10, 12, 18, 30, 42, 54\}$, with 3 clusters were tested. They are denominated $S_1, \ldots, S_8$. While they are also Gaussian based, they are more difficult to discriminate. The separation level is low, clusters do not have an identical shape and density. In addition, one cluster is a mixture of two Gaussian distributions and some amount of noise is introduced via irrelevant features. The formulas used for cluster generation are as follows, i being the dimension:

[1] (Kärkkäinen & Fränti, 2002)
[2] (Fränti & Virmajoki, 2006)
[3] (Fu & Medico, 2007)
[4] http://cs.joensuu.fi/sipu/datasets/
[5] https://github.com/deric/clustering-benchmark/tree/master/src/main/resources/datasets
[6] http://r-riad.net/

Table 2
The competitive algorithms with their parameters.

Algorithm	Parameters	Range	Ref
$Kmeans^{++}(A_1)$	c	[2, 20]	Jain (2010)
Dbscan(A_2)	ϵ, Minpts	[0.05, 0.25] $\sqrt{n}\cdot$[0.05, 0.25]	Ester, et al. (1996)
Recon-Dbscan(A_3)	ϵ, θ, τ	$\sqrt{n}\cdot$[0.05, 0.25], $\epsilon\cdot$[1, 5], [0.25, 0.5]	Zhu et al. (2016)
Snn(A_4)	$MinPts$, k	[2, 10], $\sqrt{n}\cdot$[0.05, 0.5]	Ertöz et al. (2003)
Snn-radius(A_5)	MinPts, k, ϵ	[2, 10], $\sqrt{n}\cdot$[0.05, 0.5] [0.05, 0.3]	Ertöz et al. (2003)
HierGowda(A_6)	k	$\sqrt{n}\cdot$[0.05, 0.5]	Gowda and Krishna (1978)
DPeaks (pioneer)(A_7)	c, d_c	[2, 20], [0.018, 0.05]	Rodriguez and Laio (2014)
DPdataField(A_8)	c	[2, 20]	Wang et al. (2016)
ComparativeDP(A_9)	c	[2, 20]	Li and Tang (2018)
Scdot(A_{10})	c	[2, 20]	Cheng, Lu, Liu, Huang, and Cheng (2016)
Munec (A_{11})	u	[0.01, 0.1]	Ros and Guillaume (2019b)
HierOpt(A_{12})	c	[2, 20]	Ros and Guillaume (2019a)
Kdmutual(A_{13})	c	[2, 20]	Ros et al. (2020)
Rnn-Dbscan(A_{14})	k	$\sqrt{n}\cdot$[0.05, 0.5]	Bryant and Cios (2017)
Block-Dbscan(A_{15})	ϵ, Minpts	[0.05, 0.25] $\sqrt{n}\cdot$[0.05, 0.25]	Chen, et al. (2021)
Path-scan(A_{16})	t	[0.05, 0.9]	X

Table 3
The main sources of variation and their corresponding code.

	1	2	3
Size(c1)	Similar	Small variation	Large variation
Shape(c2)	Spherical/square	Long or thin	Ring, arbitrary
Separation(c3)	Well separated	Low/very Low	Small Overlap
Noise(c4)	None	Small amount	Large amount
Density(c5)	No variation	Inter clusters	Inter and intra clusters

Table 4
The twelve data sets and their classification (c1: size, c2: shape, c3: separation level, c4: noise, c5: density.

	Size	#C	Name	Origin	c1	c2	c3	c4	c5
D_1	3000	4	A.set 1	Foot 1	2	3	2	1	2
D_2	5250	2	A.set 2	Foot 1	1	2	1	1	1
D_3	240	2	FLAME	Foot 3	1	2	3	1	1
D_4	373	2	Circle	Foot 4	1	2	2	1	3
D_5	5401	15	S.sets	Foot 2	1	2	2	1	1
D_6	5000	15	S.sets	Foot 2	2	2	1	1	1
D_7	10000	9	Cluto-t7.10k	Foot 5	3	3	3	3	1
D_8	5401	15	VariousSize	Foot 6	2	2	1	1	3
D_9	2200	4	Concentric	Foot 6	1	3	1	2	1
D_{10}	2000	2	Spiralnoise	Foot 6	1	3	2	2	1
D_{11}	2500	15	Complexshape	Foot 6	1	3	1	2	2
D_{12}	3800	6	Densdiversity	Foot 6	3	2	2	2	3

- the first cluster (green in Fig. 9) is non spherical: $\mu_i = -2$ if $i \neq 1$ $\mu_1 = 0$, $\sigma_i = 0.5$ if i is odd otherwise $\sigma_i = 0.1 + 2 * rand(0.1)$;
- 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: $\mu_i = -1$, $\sigma_i = 0.3$.

Moreover some random features were added. Their number, d_r, depends on the initial dimension space, d_f.

The final dimension is $d = d_f + d_r$. d_r is computed according to Eq. (9).

$$d_r = \begin{cases} d_f/2 & if \quad d_f < 10 \\ d_f & otherwise \end{cases} \tag{9}$$

The third experiment was based on the *genRandomClust* R package. The idea is to illustrate the behavior when clusters are represented by pseudo uniform data instead of density peaks. Hence, clusters have different size, shapes, densities and irregularities. This is an implementation of the method proposed in Qiu and Joe (2006a). 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 Qiu and Joe (2006b). 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 co-variance 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. To each data set 20% of uniform noise is added. To sum up, the generated clusters are spherical based, more or less elongated. The difficulty stems from the hybridization of sources of clustering issues at especially low separation levels (0.1 and 0.2). The tests were carried out in d={2, 3, 5, 7, 9, 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 $[100+20d, 300+20d]$ while increasing linearly with the dimension. The number of clusters was also randomly chosen ($rangeK = [3, 6]$) at each configuration to provide diversity. These data sets are denominated $U_1, \ldots, U_{28}$. For low values of $SepVal$, the clusters become less and less separable (see example in Fig. 10). The phenomenon is amplified when the space dimension increases due to the curse of dimensionality. At a given level, there is no separation.

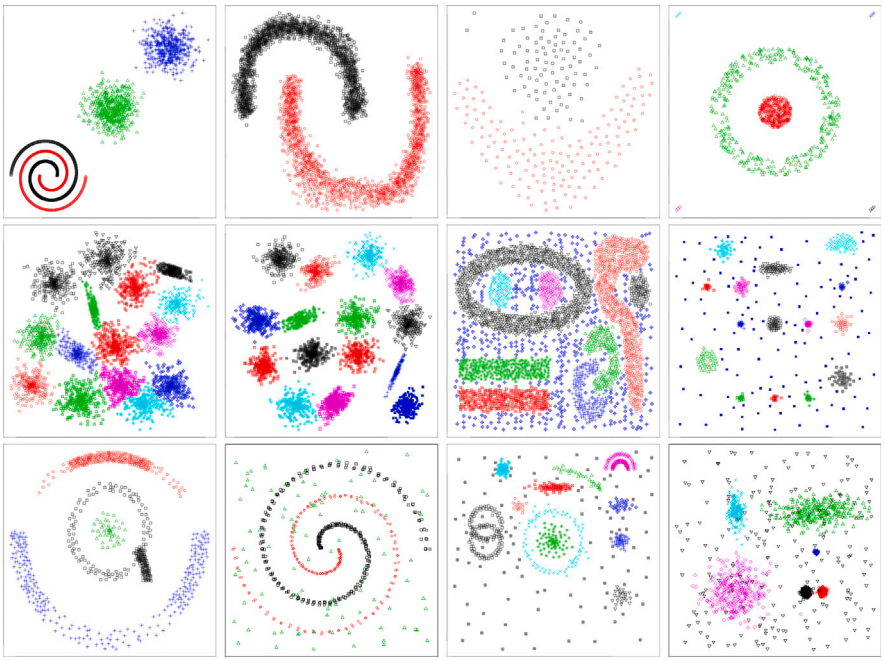

Fig. 7. The twelve data sets ($D_1 - D_{12}$ from left to right and top to bottom). The axis labels are the x and y coordinates.

Fig. 8. Views of the data set examples (8 and 16 clusters) in the first two dimensions.

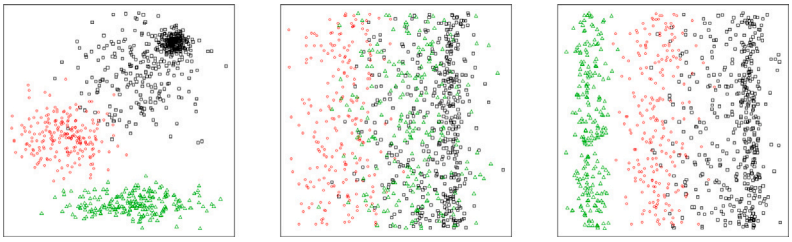

Fig. 9. Views of the first data set, respectively 1 – 2, 1 – 3 and 2 – 3 axes.

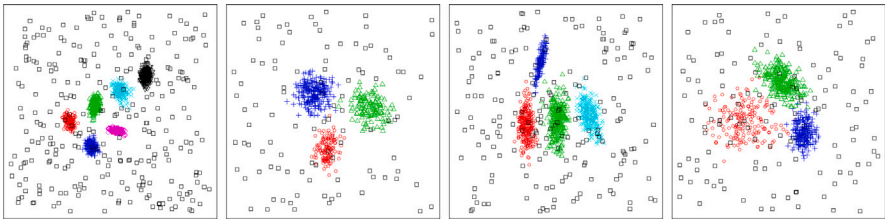

Fig. 10. Four configurations of $SepVal$ for random cluster generation in $d = 2$ (axes x and y), from left to right: 0.4, 0.3, 0.2 and 0.1.

4.3. Results and discussion

4.3.1. Results related to Path-scan

When dealing with synthetic data sets, the ground truth clusters are easily discovered by the algorithm with high efficiency regarding both $Mutual_I$ and F_{score} indices that are all around 0.9 and more. Table 5 gives details of the discovery process for the data set G_1. The complete scores resulting from 10 runs are provided (average and standard deviation) in Tables 6 and 7. For the ($U_1 - U_{28}$) series, the results for Sep-val=0.3, 0.4 ($U_3 - U_4$, ..., $U_{27} - U_{28}$) are not

Table 5
Hierarchical evolution: data set G_1.

t	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
#C	26	26	9	9	9	9	9	9	9
%M	0.9	0.91	0.57	0.88	0.94	0.96	0.99	0.99	0.99

Table 6
Path-Scan results: results: series D and G.

	#C	Found #C	Best Mutual-I	Best F-score
D_1	4	4	1	1
D_2	2	2	1	1
D_3	2	2	0.963(0.02)	0.995(0.01)
D_4	2	2	1	0.984(0.02)
D_5	15	13	0.954(0.02)	0.965(0.02)
D_6	15	15	0.995(0.01)	0.955(0.02)
D_7	9	9	0.989(0.01)	0,99(0.00.)
D_8	15	15	0.934(0.02)	0.895(0.02)
D_9	4	4	1	1
D_{10}	2	2	1	1
D_{11}	9	9	0.940(0.02)	0.901(0.02)
D_{12}	6	6	0.995(0.01)	0.998(0.01)
G_1	9	9	1	1
G_2	9	9	1	1
G_3	9	9	1	1
G_4	9	9	1	1
G_5	16	16	1	1
G_6	16	16	1	1
G_7	16	16	1	1
G_8	16	16	1	0.937(0.02)

Table 7
Path-Scan results: series S and U.

	#C	Found #C	Best Mutual-I	Best F-score
S_1	3	3	0.974(0.02)	0.997(0.01)
S_2	3	3	1	1
S_3	3	3	1	1
S_4	3	3	0.867(0.02)	0.970(0.01)
S_5	3	3	1	1
S_6	3	3	0.967(0.02)	0.996(0.01)
S_7	3	3	1	1
S_8	3	3	1	1
U_1	5	5	0.893(0.02)	0.954(0.02)
U_2	4	4	0.992(0.01)	0.998(0.01)
U_5	6	6	0.919(0.02)	0.967(0.01)
U_6	4	4	0.994(0.01)	0.999(0.01)
U_9	5	5	0.787(0.03)	0.868(0.02)
U_{10}	4	4	0.975(0.01)	0.995(0.01)
U_{13}	5	5	0.805(0.02)	0.933(0.01)
U_{14}	4	4	0.886(0.02)	0.910(0.02)
U_{17}	5	4	0.742(0.01)	0.806(0.01)
U_{18}	5	5	0.933(0.01)	0.984(0.01)
U_{21}	4	2	0.387(0.05)	0.503(0.04)
U_{22}	4	3	0.649(0.03)	0.762(0.04)
U_{25}	5	2	0.278(0.03)	0.568(0.02)
U_{26}	4	2	0.305(0.02)	0.536(0.01)

reported as all are above 0.95 for the two indices. The results for Sepval={0.1, 0.2} are more interesting as the efficiency slows down with the increase in the space dimension. For all data sets, the stronger the separation between clusters, the more the algorithm provides a very distinguishable solution while in other cases suggesting several partitions as for D_1 one can see that 2, 3 and 4 partitions are clustering compatible.

The data set D_7 contains many noise points but the two indices are very high. By extending the repeatability tests (1000 runs) it was observed that there are some rare cases where some clusters are not perfectly identified (less than 2%). The variation is due to the sampling algorithm which includes some randomness that modifies the frontier configuration between clusters. As shown in Fig. 11, the nine clusters are perfectly identified by the algorithm despite the presence of many noisy patterns, complex shapes and partial overlapping. The original data set was initially sampled from 10000 to 4000 patterns: 453 prototype patterns were detected, among which 224 were qualified as core and support patterns (see algorithm 1). From the connexity matrix (see algorithm 5) and a given threshold t, each cluster is identified by a sub-set of core/support patterns (specific color) and regions of noisy patterns are covered by core/support patterns (in gray) that are not representative and not connectable via a path. For t=0.6, the discovered partition is qualified as more than 50% of original patterns are attached to the nine connex components (among 40 found). Each of them has a minimum of representativeness (1%). This is the condition (line 9) formalized in Algorithm 6.

The performances are also promising in higher dimension as can be shown in the $(G_1 - G_8)$, $(S_1 - S_8)$ and $(U_1 - U_{28})$ series. For the $(G_1 - G_8)$ series a mean of the two indices is more than 0.99 and for the $(S_1 - S_8)$ series which contains irrelevant features both scores are more than 0.95. Table 5 gives the partitions found (#C) from $t = 0.9$ to $t = 0.1$ and the G_1 data set containing 9 rather well-separated clusters. They are easily discovered from $t = 0.7$ and are similar for lower values. This highlights a clear separation.

For the $(U_1 - U_{28})$ series, the scores are slightly lower. For $SepVal = 0.3$ and 0.4, they are always high (more than 0.95) even in high dimensions. The algorithm however fails in its discovery process with this series for low $SepVal$ values (0.1 and 0.2) when the dimension increases (from U_8 to U_{24}). It cannot discover all the expected clusters. In reality, the difference in the distance between pairs of items diminishes with the increase in dimension and therefore the discrimination is no longer possible as the overlapping is too strong. This is attested by the level of the Silhouette index, which is very low in this case. From $d = 10$ and $SepVal = 0.1$, the Silhouette index is less than 0.1, excluding the possibility of discovery via a connective approach.

4.3.2. Comparison results with 15 competitors

Synthetic comparison:. A synthesis of the results is given in Table 8 and a timing comparison in Table 9. A majority of competitors succeed in discovering clusters when they are well-separated such as the $(G_1 - G_8)$ series even in high dimension. Twelve of them obtain a fully recognition with score of 1. They fail, however, when there is a large overlap even with data sets of moderate dimensions such as the $(U_1 - U_{28})$ series. Connective algorithms are generally more efficient when dealing with complex shapes than density peaks algorithms such as *Kmeans*$^{++}$ and the *Dpeak* family. Some algorithms appear to be globally less competitive such as *HierGowda* and *Snn*.

Between these two situations, the performances are less clear-cut and more varied. Depending on the selected databases, our algorithm performs similarly to but better than connective approaches. It is less accurate at detecting single peaks than *Kmeans*$^{++}$ or *Dpeaks* with the $(U_1 - U_{28})$ series but it can handle more varied situations than these algorithms.

Table 8 reports the results obtained for the four series of data. For the sake of clarity, average and standard deviation are given for the Mutual Index and F-Score by series of data sets. The scores more than 0.9 are in bold to better illustrate the difference between competitors when faced with different clustering situations. *Path-Scan* and *Kdmutual* are the only algorithms to provide relevant results in all the series.

There is a disparity between processing times and the number of parameters to be tuned. *Kmeans*$^{++}$, *Dpeak* and *Block-Dbscan* are the fastest algorithms while *HierOpt* and *Rnn-Dbscan* the slowest. *Path-scan* is in the middle and appears to be computationally interesting as shown in Table 9). The number of tuning "soars" with the number of parameters as the computational time is multiplicative. *Path-Scan* is

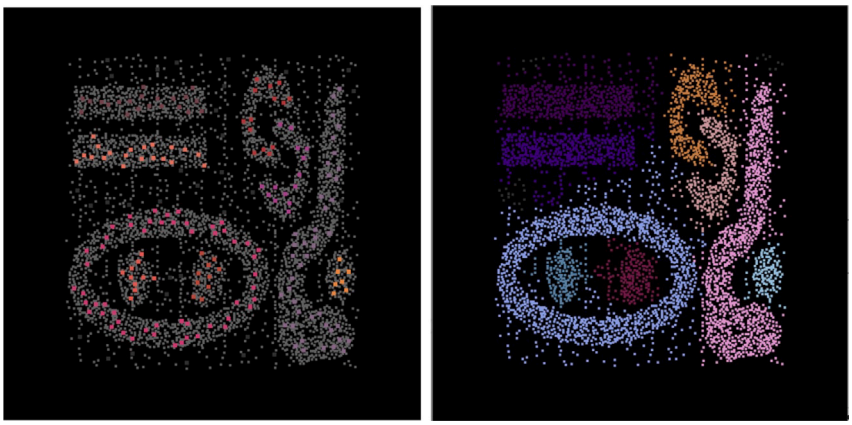

Fig. 11. Discovered partition for D_7 and t=0.6: on the left the partition via the prototypes and on the right the final partition in the original space.

Table 8
Synthetic data sets (average and standard deviations for 2 indices).

	$D_1 - D_{12}$		$G_1 - G_8$		$S_1 - S_8$		$U_1 - U_{28}$	
	Mutual I	F-score	Mutual I	F-Score	Mutual I	F-Score	Mutual I	F-Score
A_1	0.697(.22)	0.792(.13)	0.828(.15)	0.808(.17)	**1**	**1**	**0.953(.04)**	**0.985(.01)**
A_2	**0.949(.13)**	0.926(.09)	**1**	**1**	0.375(.51)	0.428(.53)	**0.96(.13)**	**0.945(.0.02)**
A_3	**0.959(.09)**	0.931(.08)	**1**	**1**	0.375(.51)	0.428(.53)	**0.96(.13)**	**0.942(.12)**
A_4	0.833(.22)	0.871(.12)	**1**	**1**	0.189(.2)	0.332(.41)	0.63(.37)	0.695(.39)
A_5	0.868(.15)	0.885(.11)	**1**	**1**	0.714(.2)	0.332(.41)	0.63(.37)	0.657(.38)
A_6	0.501(.38)	0.601(.30)	0.801(.2)	0.804(.6)	**0.974(.44)**	**0.924(.12)**	0.37(.43)	0.600(.20)
A_7	0.747(.19)	0.798(.15)	**1**	**1**	**0.975(.06)**	**0.99(.01)**	0.852(.18)	**0.905(.12)**
A_8	0.807(.19)	0.822(.16)	**1**	**1**	**0.975(.06)**	**0.993(.02)**	0.845(.18)	0.896(.13)
A_9	0.748(.21)	0.798(.15)	**1**	**1**	**0.902(.06)**	**0.99(.01)**	0.869(.18)	**0.918(.13)**
A_{10}	0.834(.07)	0.856(.13)	**1**	**1**	0.972(.05)	**0.982(.02)**	0.703(.30)	0.75(.25)
A_{11}	**0.936(.08)**	**0.947(.09)**	**1**	**1**	**0.902(.14)**	**0.977(.04)**	0.795(.35)	0.819(.35)
A_{12}	**0.914(.09)**	**0.932(.07)**	**1**	**1**	**0.98(.03)**	**0.99(.0)**	0.725(.32)	0.774(.25)
A_{13}	**0.915(.05)**	**0.924(.08)**	**1**	**1**	**0.99(.0)**	**0.99(.0)**	**0.923(.08)**	**0.909(.09)**
A_{14}	0.889(.08)	**0.907(.06)**	0.714(.48)	0.714(.48)	0.886(.21)	**0.983(.04)**	0.712(.34)	0.742(.34)
A_{15}	**0.931(.08)**	**0.959(.05)**	0.708(.48)	0.708(.48)	0.560(.47)	0.497(.53)	0.797(.32)	0.813(.14)
A_{16}	**0.965(.05)**	**0.943(.08)**	**0.991(.02)**	**0.991(.02)**	**0.976(0.04)**	**0.99(0.01)**	0.857(.22)	**0.906(.15)**

Table 9
Timing comparison (in ms) between competitive algorithms for one run (average time on databases) using a single personal computer. (DELL Precision 5530, intel Core 17).

Method	μ	σ	# para	Method	μ	σ	# para
A_1	3.3 10^2	92.8	1	A_9	3.1 10^2	493.1	1
A_2	1.6 10^3	807.4	2	A_{10}	1.3 10^4	1814.3	1
A_3	1.6 10^3	823.5	3	A_{11}	1.5 10^3	2308.0	1
A_4	5.6 10^3	1078.5	2	A_{12}	1.1 10^4	5874.7	1
A_5	5.6 10^3	1110.4	3	A_{13}	1.5 10^3	23320.0	1
A_6	4.5 10^3	2289.4	1	A_{14}	1.4 10^4	82308.4	1
A_7	2.9 10^2	183.4	1	A_{15}	4.2 10^2	1021.1	2
A_8	5.7 10^3	1454.2	2	A_{16}	5.8 10^3	1256.4	1

almost "self-tuning" as only dependent on the threshold linkage level. Despite its internal complexity, the running time of *Path-scan* for one run is located at the median of the competitive approaches, i.e. worse than that of *Block-Dbscan* but better than that of *Rnn-Dbscan*.

Detailed comparisons:.

Snns (A_4, A_5) and Dbscan algorithms (A_2, A_3). These methods provide scores that are similar to or lower than *Path-scan*. The reason for this advance is that the proposal inherits the strengths of *Snn* and *Dbscan* while being self-tuning and more adaptive as it is driven in a differential mode. In addition, a tedious and computational multi-tuning is required without any guarantee of success even in moderate dimensions. The algorithms derived from *Recon-Dbscan* and *Snn-radius* produce interesting results. They are however less practical they require the tuning of additional parameters.

HierGowda. This algorithm does not appear to be competitive except when the clusters are well separated. It appears to be limited when tackling variations in data organization and overlapping contexts.

DPC algorithms (A_7, A_8, A_9 and A_{10}) and Kmeans^{++}. These algorithms are highly efficient when clusters match well with density peaks even in presence of noise or overlapping. They succeed well with the $(U_1 - U_{28})$ series even in high dimensions whereas all the connective algorithms fail including our algorithm for low Sepval values. On the contrary, they systematically failed in presence of irregular shapes, large size variation even with low dimensions as in the $(D_1 - D_{12})$ series. *Scdot* performs well but not better than *Path-scan* while being very slow compared to the other DP algorithms.

HierOpt. This algorithm performs similarly to the most competitive algorithms as shown in Table 8. It can handle complex cluster shapes and deal with overlapping as well as varied density cases as in the $(S_1 - S_8)$ series. Its performances are however worse for the $(U_1 - U_{28})$

Table 10
The real world data sets.

	Size	# d	Name	Origin
R_1	857357	3	Transactions90k	Alcalá-Fdez, et al. (2011)
R_2	34112	3	House8	Fränti and Virmajoki (2006)
R_3	45781	3	Tamildanu	*UCI*
R_4	245057	3	Skin segmentation	*UCI*
R_5	1044506	9	House power	*UCI*
R_6	440	8	Wholesale Customers	*UCI*
R_7	150	4	IRIS	*UCI*
R_8	569	32	Breast cancer	*UCI*
R_9	19020	11	Magic Gamma Telescope	*UCI*
R_{10}	4339	6	CTG	*UCI*
R_{11}	5000	8	Bank	*UCI*
R_{12}	1372	4	Banknote	*UCI*
R_{13}	2126	21	Breast Cancer	*UCI*
R_{14}	68040	9	Color moment	Alcalá-Fdez, et al. (2011)
R_{15}	1473	9	Contraceptive	Alcalá-Fdez, et al. (2011)
R_{16}	6876	13	Marketing	Alcalá-Fdez, et al. (2011)
R_{17}	7200	21	Tyroid	Alcalá-Fdez, et al. (2011)
R_{18}	2201	3	Titanic	Alcalá-Fdez, et al. (2011)

series as it is quickly penalized when the space dimension is not low. A serious shortcoming is that it is one of the slowest competitive algorithms (*cf.* Table 9), which is due to the requirement of many distance calculations.

Munec. This algorithm achieves more comparable results and appears to be very competitive, giving scores higher than 0.9 for the three first series. It has some difficulties with the $(U_1 - U_{28})$ series (SepVal=0.1 and 0.2 and dim $\geq$ 5) and requires the u parameter to be tuned.

Kdmutual. This algorithm obtains relevant scores for a majority of the tested data sets. *Kdmutual* is not however ideal for a discovery process as it requires the number of clusters as input.

Rnn-Dbscan. This algorithm gives good results for the $(D_1 - D_{12})$ series but performs the least well among the connective approaches. The method obtains high scores for the $(G_1 - G_8)$ and $(S_1 - S_8)$ series, but poor results with data sets of high dimension spaces. For the $(U_1 - U_{28})$ series, the performances are at the median of the competitors. In higher dimensions, *Rnn-Dbscan* appears to be less accurate but its greatest weakness resides in the computational time.

Block-Dbscan. This algorithm achieves comparable results to *Path-scan* when dealing with data of low dimension spaces or when the level of separation between clusters is high. For the $(S_1 - S_8)$ series, the scores are mitigated at very low dimensions such as for S_1, very high at medium ones and no partition is found for S_7 and S_8. It gives worse results than *Path-scan* with this series as well as for the $(G_1 - G_8)$ and $(U_1 - U_{28})$ ones while providing similar results for the $(D_1 - D_{12})$ series. Its computational time is very low, and much better than most other competitive algorithms except *Kmeans*$^{++}$.

4.4. Experiments with real Data sets

In real life, data contain multi-internal data structures and data distributions are not as smart as those of synthetic data sets. For a discovery process, the original labels of data are unknown (unknown ground truth) and the role of clustering algorithms consists in analyzing the data without any other information. The real objective is to discover if natural partitions exist and if so, to assess their quality. The silhouette index was selected as the metrics to measure the level of the partitions discovered by the algorithms. Eighteen bench-marked data sets denoted R_1 to R_{18} covering a vast array of applications were selected in order to satisfy a number of criteria. These data come from popular repository databases that are widely used in the literature, as shown in Table 10.

4.4.1. Results related to Path-scan

The discovery result is depicted in Table 11. For each database, 10 configurations are evaluated (from $t = 0.9$ to $t = 0.05$). If a partition is found it is represented by its number of clusters and the corresponding silhouette score; X(X) means that the algorithm fails to find a partition. This is the case when the final solution contains more than 20 eligible clusters, or only 1 eligible cluster meaning that no partition has been found. An eligible cluster contains more than 2% of the whole patterns in a given data set.

One can see that for some data sets, such as R_2, no partitions are found. For others, such as R_6, several partitions corresponding to different thresholds are found. For others such as R_8 and R_9 one partition appears to dominate the others: different thresholds give the same cluster number with high silhouette scores. At this stage, one can only say that some partitions can be found by our algorithm that sometimes fails in its discovery process. It should be underlined that no conclusion can be drawn from these results: they may be due to a weakness of the algorithm itself as well as a consequence related to the data structure that does not contain natural clusters. The real relevance can only be assessed by comparing with the competitors submitted to the same rule. The discovery aspect and the partition quality for the competitive algorithms are given in the following section.

4.4.2. Comparison results with 15 competitors

As for the task done for *Path-scan*, each parameter tuning produces a partition that is formalized by the number of clusters and the associated maximum silhouette score. The difference between algorithms is related to the input parameters, which differ in number and nature. The delivered partition is then analyzed. The retained configurations are those that provide at least two clusters, less than or equal to 20 clusters (each cluster being sufficiently representative, *i.e.* including 2% of patterns), and no more than 20% of noise.

Table 12 summarizes the main discovery results. It distinguishes three ranges of cluster number and four ranges of silhouette values. Then, for each algorithm (from A_1 to A_{16}) there are 12 (3×4) values. Each value is the percent of discovery for all the data sets according to the number of clusters and the silhouette score. The higher the silhouette score the more relevant the discovery process is. The best values are in bold for each couple of #C and Silhouette value (Sil). The interpretability with the number of clusters is more questionable in absolute terms. It is however admissible that a clustering algorithm able to discover good partitions (according to the silhouette score) at different levels has more ability than another one that discovers on only one level. Hence, a good score with a small number of clusters is more interesting for a discovery process.

It can be seen that the results highlight significant variation between the competitive algorithms. For several databases, more than half of

Table 11
Results: real data bases, discovery score as a function of #*C* found and silhouette index.

t	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.05
R_1	12(0.66)	20(0.65)	19(0.67)	20(0.72)	20(0.68)	20(0.66)	20(0.66)	22(0.60)	22(0.57)	X(X)
R_2	X(X)	3(0.91)	3(0.83)	3(0.81)	3(0.78)	3(0.79)	3(0.79)	3(0.79)	3(0.79)	3(0.78)
R_3	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)
R_4	16(0.30)	19(0.44)	X(X)	10(0.53)	13(0.50)	11(0.54)	11(0.61)	8(0.63)	6(0.56)	6(0.56)
R_5	X(X)	X(X)	X(X)	X(X)	11(0.31)	11(0.34)	11(0.28)	7(0.25)	4(0.39)	4(0.37)
R_6	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)	X(X)
R_7	X(X)	X(X)	14(0.25)	11(0.30)	10(0.20)	4(0.50)	4(0.50)	4(0.50)	4(0.50)	5(0.5)
R_8	X(X)	X(X)	X(X)	2(0.99)	2(0.99)	2(0.99)	2(0.99)	2(0.99)	2(0.99)	2(0.99)
R_9	X(X)	X(X)	X(X)	2(0.99)	2(0.99)	2(0.99)	2(0.99)	2(0.99)	2(0.99)	2(0.99)
R_{10}	3(0.697)	2(0.792)	X(X)	2(0.232)	3(0.697)	2(0.792)	X(X)	2(0.232)	X(X)	2(0.232)
R_{11}	X(X)	X(X)	X(X)	13(0.30)	14(0.23)	11(0.20)	13(0.15)	X(X)	5(0.21)	4(0.38)
R_{12}	X(X)	X(X)	X(X)	X(X)	X(X)	9(0.35)	8(0.30)	6(0.24)	3(0.24)	2(0.99)
R_{13}	X(X)	X(X)	X(X)	X(X)	X(X)	2(0.99)	2(0.99)	2(0.99)	2(0.99)	2(0.99)
R_{14}	2(1.0)	3(0.81)	3(0.81)	4(0.90)	3(0.82)	3(0.82)	3(0.90)	3(0.91)	2(0.78)	2(0.78)
R_{15}	X(X)	X(X)	X(X)	X(X)	11(0.1)	7(0.09)	6(0.13)	6(0.2)	6(0.21)	5(0.21)
R_{16}	X(X)	X(X)	X(X)	2(0.03)	X(X)	X(X)	X(X)	X(X)	2(0.99)	2(0.99)
R_{17}	X(X)	X(X)	8(0.07)	7(0.25)	13(0.26)	12(0.28)	11(0.26)	10(0.3)	8(0.32)	7(0.33)
R_{18}	9(0.98)	9(0.98)	8(0.92)	8(0.92)	3(0.72)	4(0.71)	3(0.66)	3(0.66)	3(0.66)	3(0.66)

Table 12
Results: real data bases, discovery score as a function of #*C* found and silhouette index.

#*C*	≤ 5				≤ 10				≤ 20			
Sil	≥ 0.3	≥ 0.4	≥ 0.5	≥ 0.6	≥ 0.3	≥ 0.4	≥ 0.5	≥ 0.6	≥ 0.3	≥ 0.4	≥ 0.5	≥ 0.6
A_1	72.22	50.00	33.33	27.78	72.22	50.00	38.89	27.78	72.22	50.00	38.89	27.78
A_2	22.22	5.56	5.56	5.56	27.78	11.11	11.11	11.11	33.33	16.67	11.11	11.11
A_3	5.56	0.00	0.00	0.00	11.11	5.56	5.56	5.56	16.67	11.11	11.11	11.11
A_4	5.56	5.56	0.00	0.00	16.67	16.67	11.11	11.11	22.22	22.22	16.67	16.67
A_5	5.56	5.56	0.00	0.00	11.11	11.11	5.56	5.56	16.67	16.67	11.11	11.11
A_6	27.78	22.22	22.22	16.67	27.78	22.22	22.22	16.67	27.78	22.22	22.22	16.67
A_7	44.44	33.33	27.78	16.67	44.44	33.33	27.78	22.22	44.44	33.33	33.33	22.22
A_8	42.86	28.57	21.43	7.14	42.86	28.57	21.43	7.14	42.86	28.57	21.43	7.14
A_9	38.89	33.33	27.78	11.11	38.89	33.33	27.78	16.67	38.89	33.33	27.78	16.67
A_{10}	61.11	38.89	27.78	22.22	66.67	44.44	33.33	22.22	66.67	44.44	33.33	22.22
A_{11}	33.33	16.67	16.67	16.67	38.89	33.33	22.22	22.22	38.89	33.33	22.22	22.22
A_{12}	**77.78**	**55.56**	50.00	38.89	**77.78**	**55.56**	50.00	38.89	**77.78**	55.56	50.00	38.89
A_{13}	61.11	33.33	22.22	11.11	61.11	33.33	22.22	11.11	61.11	33.33	22.22	11.11
A_{14}	38.89	27.78	22.22	22.22	38.89	27.78	22.22	22.22	44.44	33.33	27.78	27.78
A_{15}	33.33	22.22	22.22	16.67	38.89	27.78	27.78	22.22	38.89	27.78	27.78	22.22
A_{16}	61.11	50.00	**50.00**	**44.44**	72.22	**55.56**	**55.56**	**50.00**	**77.78**	**61.11**	**61.11**	**55.56**

the algorithms are unable to discover eligible "natural" partitions such as R_9, R_{10} and R_{14}. *HierGowda* fail for about half of the data sets. Several algorithms such as *Dbscan* and *Snn* families as well as *Snn-Radius* require 10^2 and 10^3 tests, the usability of *Path-scan* is clearly higher. Better results could possibly have been obtained with more tuning but this is a weakness. *Dbscan*, *Snn*, *Recon-Dbscan* and The *Rnn-Dbscan* algorithm provides correct results, the best among the variants of *Dbscan* but lower than our algorithm which obtains the best percentage for 7 configurations and equals the results of *Kdmutual* in two cases. Concerning *Rnn-Dbscan*, there is a serious issue concerning its computational cost. The *Block-Dbscan* algorithm is very fast but its global performances on the real world data sets appear to be weaker compared to the other algorithms with a discovery score of 13% for a high silhouette index threshold.

The algorithms taking the number of clusters as input succeed in finding partitions more frequently but the level of silhouette score is relatively low. For example, the *Kmeans*$^{++}$ algorithm provides several partitions for R_1 but the maximum silhouette score is less than 0.3.

In synthesis, the best algorithms for the discovery process appear to be *Path-scan* and *Kdmutual*, the latter requiring the number of cluster as input. For the tested databases, they are the only algorithms able to discover more than 60% of eligible partitions. For the others, there is a clear-cut difference between the performances. *Path-scan* is capable of providing eligible natural partitions in different cluster ranges without tedious tuning.

5. Conclusion

Path-scan is a novel clustering algorithm based on core points and connexity able to discern a wide class of data of arbitrary shapes and sizes in presence of noise and outliers. It requires minimum interaction with the investigator and its results are easily understandable. Its core idea is to work with core and support patterns that are linked via connective paths. It is based on the concept of density differential and the local density is re-estimated via the path context. This gives more flexibility than approaches dealing with static parameters requiring laborious tuning. While the notions involved are shared with other recent studies that aim at introducing more intelligence in clustering algorithms, the approach is conceptually rather novel.

Experimental results showed that it is a powerful algorithm for the discovery process without needing the number of clusters as input. An empirical evaluation was performed comparing the Path-scan algorithm with 15 competitive algorithms. In terms of efficiency, it is at least comparable with the most recent connectivity-based competitive algorithms while being faster or easier to tune. As it works with the support of prototype patterns (including core and support patterns), it is faster than most connective algorithms despite its internal complexity. When "natural" clusters exist, the algorithm can discover them without any tuning, which is a strong asset. When partitions are less "obvious", it produces results that highlight the data structure, which remains the main objective.

There are several areas for improvement that are of interest to the community. If specific mechanisms are introduced to re-estimate densities in the local context of a path, there is a dependency with

the first estimation that remains global. Accuracy should be higher if this first estimation could take the local context better into account at this stage. Partitions are extracted from the connexity matrix which is strongly dependent on the density. A novel mechanism could be devised to involve a notion of distance that could group two clusters that are strongly separated under the density criterion but spatially very close.

To handle the "curse of dimensionality" problem, a future development could be based on a systematic pre-processing stage using either an unsupervised feature selection or an input space transform. Lastly, another perspective is to overcome the limitation of knowledge extraction from very large data sets using a specific data structure and/or distributed approaches and parallel algorithms.

CRediT authorship contribution statement

Frédéric Ros: Conceptualization, Methodology, Formal analysis, Software. **Serge Guillaume:** Methodology, Writing – original draft, Writing – review & editing. **Rabia Riad:** Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Afzalan, M., & Jazizadeh, F. (2019). An automated spectral clustering for multi-scale data. *Neurocomputing*, *347*, 94–108.

Agarwal, P. K., Har-Peled, S., Varadarajan, K. R., et al. (2005). Geometric approximation via coresets. *Combinatorial and Computational Geometry*, *52*, 1–30.

Agarwal, P. K., Procopiuc, C. M., & Varadarajan, K. R. (2002). Approximation algorithms for k-line center. In *European symposium on algorithms* (pp. 54–63). Springer.

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

Ankerst, M., Breunig, M. M., Kriegel, H.-P., & Sander, J. (1999). OPTICS: Ordering points to identify the clustering structure. *ACM Sigmod Record*, *28*(2), 49–60.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on management of data* (pp. 93–104).

Bryant, A., & Cios, K. (2017). RNN-DBSCAN: A density-based clustering algorithm using reverse nearest neighbor density estimates. *IEEE Transactions on Knowledge and Data Engineering*, *30*(6), 1109–1121.

Campello, R. J., Moulavi, D., & Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining* (pp. 160–172). Springer.

Chen, Y., Tang, S., Zhou, L., Wang, C., Du, J., Wang, T., et al. (2018). Decentralized clustering by finding loose and distributed density cores. *Information Sciences*, *433*, 510–526.

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

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

Dhillon, I. S., Guan, Y., & Kulis, B. (2004). Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 551–556).

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

Ertöz, L., Steinbach, M., & Kumar, V. (2003). Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM international conference on data mining* (pp. 47–58). SIAM.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd. Vol. 96* (34), (pp. 226–231).

Ezugwu, A. E., Ikotun, A. M., Oyelade, O. O., Abualigah, L., Agushaka, J. O., Eke, C. I., et al. (2022). A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, *110*, Article 104743.

Fränti, P., & Virmajoki, O. (2006). Iterative shrinking method for clustering problems. *Pattern Recognition*, *39*(5), 761–765.

Fränti, P., Virmajoki, O., & Hautamäki, V. (2006). Fast agglomerative clustering using a k-nearest neighbor graph. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(11), 1875–1881.

Fu, L., & Medico, E. (2007). FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data. *BMC Bioinformatics*, *8*(1), 3.

Geng, Y.-a., Li, Q., Zheng, R., Zhuang, F., He, R., & Xiong, N. (2018). RECOME: A new density-based clustering algorithm using relative KNN kernel density. *Information Sciences*, *436*, 13–30.

Gowda, K. C., & Krishna, G. (1978). Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern Recognition*, *10*(2), 105–112.

Guo, W., Wang, W., Zhao, S., Niu, Y., Zhang, Z., & Liu, X. (2022). Density peak clustering with connectivity estimation. *Knowledge-Based Systems*, Article 108501.

Hämäläinen, J., Jauhiainen, S., & Kärkkäinen, T. (2017). Comparison of internal clustering validation indices for prototype-based clustering. *Algorithms*, *10*(3), 105.

He, Y., Tan, H., Luo, W., Feng, S., & Fan, J. (2014). MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data. *Frontiers of Computer Science*, *8*(1), 83–99.

Hinneburg, A., & Gabriel, H.-H. (2007). Denclue 2.0: Fast clustering based on kernel density estimation. In *International symposium on intelligent data analysis* (pp. 70–80). Springer.

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, *31*(8), 651–666.

Jarvis, R. A., & Patrick, E. A. (1973). Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, *100*(11), 1025–1034.

Jiang, J., Chen, Y., Hao, D., & Li, K. (2019). DPC-LG: Density peaks clustering based on logistic distribution and gravitation. *Physica A: Statistical Mechanics and its Applications*, *514*, 25–35.

Jiang, J.-L., Fang, H., Li, S.-Q., & Li, W.-M. (2022). Identifying important nodes for temporal networks based on the ASAM model. *Physica A: Statistical Mechanics and its Applications*, *586*, Article 126455.

Kärkkäinen, I., & Fränti, P. (2002). *Dynamic local search algorithm for the clustering problem: Technical Report*, (A-2002-6), Joensuu, Finland: Department of Computer Science, University of Joensuu.

Kärkkäinen, I., & Fränti, P. (2007). Gradual model generator for single-pass clustering. *Pattern Recognition*, *40*(3), 784–795.

Karypis, G., Han, E.-H., & Kumar, V. (1999). Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, *32*(8), 68–75.

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

Li, W., Zhu, H., Liu, W., Chen, D., Jiang, J., & Jin, Q. (2018). An anti-noise process mining algorithm based on minimum spanning tree clustering. *IEEE Access*, *6*, 48756–48764.

Liu, R., Wang, H., & Yu, X. (2018). Shared-nearest-neighbor-based clustering by fast search and find of density peaks. *Information Sciences*, *450*, 200–226.

Lu, R.-k., Liu, J.-w., Zuo, X., & Li, W.-m. (2021). Multi-view subspace clustering with consistent and view-specific latent factors and coefficient matrices. In *2021 International joint conference on neural networks* (pp. 1–8). IEEE.

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

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

Mahajan, M., Nimbhorkar, P., & Varadarajan, K. (2009). The planar k-means problem is NP-hard. In *International workshop on algorithms and computation* (pp. 274–285). Springer.

McInnes, L., Healy, J., & Astels, S. (2017). Hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, *2*(11), 205.

Murtagh, F., & Contreras, P. (2017). Algorithms for hierarchical clustering: an overview, II. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *7*(6), Article e1219.

Parmar, M., Wang, D., Zhang, X., Tan, A.-H., Miao, C., Jiang, J., et al. (2019). REDPC: A residual error-based density peak clustering algorithm. *Neurocomputing*, *348*, 82–96.

Qiu, W., & Joe, H. (2006a). Generation of random clusters with specified degree of separation. *Journal of Classification*, *23*(2), 315–334.

Qiu, W., & Joe, H. (2006b). Separation index and partial membership for clustering. *Computational Statistics & Data Analysis*, *50*(3), 585–603.

Rodriguez, A., & Laio, A. (2014). Clustering by fast search and find of density peaks. *Science*, *344*(6191), 1492–1496.

Romano, S., Bailey, J., Nguyen, V., & Verspoor, K. (2014). Standardized mutual information for clustering comparisons: one step further in adjustment for chance. In *International conference on machine learning* (pp. 1143–1151).

Ros, F., & Guillaume, S. (2016). DENDIS: A new density-based sampling for clustering algorithm. *Expert Systems with Applications*, *56*, 349–359. http://dx.doi.org/10.1016/j.eswa.2016.03.008.

Ros, F., & Guillaume, S. (2017). DIDES: a fast and effective sampling for clustering algorithm. *Knowledge and Information Systems*, *50*(2), 543–568.

Ros, F., & Guillaume, S. (2018). ProTraS: A probabilistic traversing sampling algorithm. *Expert Systems with Applications*, *105*, 65–76. http://dx.doi.org/10.1016/j.eswa.2018.03.052.

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

Ros, F., & Guillaume, S. (2019b). Munec: A mutual neighbor-based clustering algorithm. *Information Sciences*, *486*, 148–170. http://dx.doi.org/10.1016/j.ins.2019.02.051.

Ros, F., Guillaume, S., El Hajji, M., & Riad, R. (2020). KdMutual: A novel clustering algorithm combining mutual neighboring and hierarchical approaches using a new selection criterion. *Knowledge-Based Systems*, Article 106220.

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017). DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems*, *42*(3), 1–21.

Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence* (pp. 1015–1021). Springer.

Tong, W., Liu, S., & Gao, X.-Z. (2021). A density-peak-based clustering algorithm of automatically determining the number of clusters. *Neurocomputing*, *458*, 655–666.

Vijaya, A. S., & Bateja, R. (2017). A review on hierarchical clustering algorithms. *J. Eng. Appl. Sci*, *12*(24), 7501–7507.

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, *17*(4), 395–416.

Wang, S., Wang, D., Li, C., Li, Y., & Ding, G. (2016). Clustering by fast search and find of density peaks with data field. *Chinese Journal of Electronics*, *25*(3), 397–402.

Xie, J., Gao, H., Xie, W., Liu, X., & Grant, P. W. (2016). Robust clustering by detecting density peaks and assigning points based on fuzzy weighted K-nearest neighbors. *Information Sciences*, *354*, 19–40.

Xie, J., & Jiang, W. (2018). An adaptive clustering algorithm by finding density peaks. In *Pacific rim international conference on artificial intelligence* (pp. 317–325). Springer.

Xie, J., Jiang, W., & Ding, L. (2017). Clustering by searching density peaks via local standard deviation. In *International conference on intelligent data engineering and automated learning* (pp. 295–305). Springer.

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

Xu, Y., Zhuang, Z., Li, W., & Zhou, X. (2018). Effective community division based on improved spectral clustering. *Neurocomputing*, *279*, 54–62.

Yang, Y., Cai, J., Yang, H., & Zhao, X. (2022). Density clustering with divergence distance and automatic center selection. *Information Sciences*.

Yaohui, L., Zhengming, M., & Fang, Y. (2017). Adaptive density peak clustering based on K-nearest neighbors with aggregating strategy. *Knowledge-Based Systems*, *133*, 208–220.

Zhang, R., Miao, Z., Tian, Y., & Wang, H. (2022). A novel density peaks clustering algorithm based on hopkins statistic. *Expert Systems with Applications*, Article 116892.

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