*Journal of Statistical Software*

*October 2019, Volume 91, Issue 1.* [*doi: 10.18637/jss.v091.i01*](https://doi.org/10.18637/jss.v091.i01)

dbscan**: Fast Density-Based Clustering with R**

**Michael Hahsler** Southern Methodist University

**Matthew Piekenbrock** Wright State University

**Derek Doran** Wright State University

**Abstract**

This article describes the implementation and use of the R package dbscan, which provides complete and fast implementations of the popular density-based clustering algo- rithm DBSCAN and the augmented ordering algorithm OPTICS. Package dbscan uses advanced open-source spatial indexing data structures implemented in C++ to speed up computation. An important advantage of this implementation is that it is up-to-date with several improvements that have been added since the original algorithms were pub- lications (e.g., artifact corrections and dendrogram extraction methods for OPTICS). We provide a consistent presentation of the DBSCAN and OPTICS algorithms, and compare dbscan’s implementation with other popular libraries such as the R package fpc, ELKI, WEKA, PyClustering, SciKit-Learn, and SPMF in terms of available features and using an experimental comparison.

*Keywords*: DBSCAN, OPTICS, density-based clustering, hierarchical clustering.

# Introduction

Clustering is typically described as the process of finding structure in data by grouping sim- ilar objects together, where the resulting groups are called clusters. The premise of many clustering algorithms is that objects assigned to the same cluster should be more similar to each other than to objects in other clusters. Similarity is often captured through some notion of distance, stemming from the fact that objects are assumed to be data points embedded in a data space in which a distance measure can be defined. Typical clustering methods either focus on solving the *k*-means problem ([MacQueen *et al.*](#_bookmark69)[1967](#_bookmark69)) or rely on parametric mixture models, each of which cluster by finding the parameters of a probabilistic model from which the observed data were most likely to have arisen. For these methods well-established soft- ware packages are available (e.g., mclust, [Scrucca, Fop, Murphy, and Raftery](#_bookmark82) [2016](#_bookmark82); mixtools, [Benaglia, Chauveau, Hunter, and Young](#_bookmark32) [2009](#_bookmark32)). Another approach is hierarchical clustering, which uses a similarity measure and a linkage criterion nested groupings of objects, often rep-

resented as a tree structure. Detailed reviews of popular clustering algorithms are provided by [Kaufman and Rousseeuw](#_bookmark65) ([1990](#_bookmark65)), [Jain, Murty, and Flynn](#_bookmark61) ([1999](#_bookmark61)), and more recently in [Aggarwal and Reddy](#_bookmark28) ([2013](#_bookmark28)).

Many of these approaches assume clusters with convex, hyper-spherical shapes ([Jain *et al.*](#_bookmark61)[1999](#_bookmark61)). In contrast, density-based clustering approaches have become increasingly popular due to their ability to capture clusters of arbitrary shapes, including non-convex shapes. Density- based approaches posit merely that clusters are contiguous dense regions in the data space (i.e., regions of high point density), separated by areas of low density ([Sander](#_bookmark79) [2011](#_bookmark79)). Density- based clustering also can handle noise, where points that reside in areas of very low density are not assigned a cluster label, but instead, are treated as outliers or noisy observations. These properties provide advantages for many applications. For example, geospatial data may be fraught with noisy data points due to estimation errors in GPS-enabled sensors ([Chen, Ji, and](#_bookmark41) [Wang](#_bookmark41) [2014](#_bookmark41)) and may have non-convex cluster shapes caused by the topology of the physical space in which the data was captured. Density-based clustering has also shown advantages for characterizing high-dimensional data ([Kailing, Kriegel, and Kröger](#_bookmark63) [2004](#_bookmark63)), where partitions are challenging to discover, and where the physical shape constraints assumed by model-based methods are more likely to be violated.

This paper focuses on an efficient implementation of the DBSCAN algorithm ([Ester, Kriegel,](#_bookmark47) [Sander, Xu *et al.*](#_bookmark47)[1996](#_bookmark47)), one of the most popular density-based clustering algorithms, whose impact earned it the SIGKDD 2014’s Test of Time Award ([SIGKDD](#_bookmark83) [2014](#_bookmark83)), and OPTICS ([Ankerst, Breunig, Kriegel, and Sander](#_bookmark29) [1999](#_bookmark29)), often thought of as an extension of DBSCAN. While surveying software tools that implemented various density-based clustering algorithms, it was discovered that existing implementations vary significantly in performance ([Kriegel,](#_bookmark67) [Schubert, and Zimek](#_bookmark67) [2016](#_bookmark67)) and may also lack important components and corrections. Specif- ically, for the statistical computing environment R ([R Core Team](#_bookmark76) [2019](#_bookmark76)), only naive DBSCAN implementations without the use of fast spatial data structures are available. An example is the implementation in the well-known Flexible Procedures for Clustering package fpc ([Hennig](#_bookmark58) [2019](#_bookmark58)). OPTICS was not available for R before the introduction of dbscan. This motivated the development of a R package for density-based clustering with DBSCAN and related algo- rithms called dbscan.

This article presents an overview of the R package dbscan, focusing on the operation of DBSCAN and OPTICS. It also provides a comparison of dbscan with a number of other open- source implementations on various benchmark datasets. We start with a brief introduction of the concepts of density-based clustering and the DBSCAN and OPTICS algorithms in Section [2](#_bookmark0). The section also gives a short review of existing software packages implementing these algorithms. Section [3](#_bookmark6) contains detailed examples that show how to use DBSCAN and OPTICS in dbscan. A performance evaluation is presented in Section [4](#_bookmark23). Concluding remarks are offered in Section [5](#_bookmark27).

# Density-based clustering

Density-based clustering is now well-studied. Conceptually, the idea behind density-based clustering is simple: given a set of data points, define a structure that accurately reflects the underlying density ([Sander](#_bookmark79) [2011](#_bookmark79)). An important distinction between density-based clustering and alternative approaches to cluster analysis, such as (Gaussian) mixture models (see, e.g., [Jain *et al.*](#_bookmark61)[1999](#_bookmark61)), is that the latter represents a parametric approach in which the observed

data are assumed to have been produced by a mixture of parametric distributions (often as- sumed to be Gaussian). While useful in many applications, parametric approaches naturally assume clusters will exhibit some convex (hyper-spherical or hyper-elliptical) shape. Other approaches, such as *k*-means clustering (where the *k* parameter signifies the user-specified number of clusters to find), share this common theme by assuming that good clusters can be found by minimizing some measure of intra-cluster variance (often referred to as cluster cohesion) and maximizing the inter-cluster variance (cluster separation, [Arbelaitz, Gurrutx-](#_bookmark30) [aga, Muguerza, Pérez, and Perona](#_bookmark30) [2013](#_bookmark30)) leading also to convex cluster shapes. Conversely, density-based clustering methods do not assume parametric distributions or use variance, and thus are capable of finding arbitrarily-shaped clusters, handle varying amounts of noise, and require no prior knowledge regarding how to set the number of clusters. Next, we discuss the most popular density-based clustering algorithm, called DBSCAN.

## DBSCAN: Density-based spatial clustering of applications with noise

As one of the most cited density-based clustering algorithms ([Microsoft Academic Search](#_bookmark70) [2017](#_bookmark70)), DBSCAN ([Ester *et al.*](#_bookmark47)[1996](#_bookmark47)) is likely the most used density-based clustering algorithm in the scientific community today. The central idea behind DBSCAN and its extensions and revisions is the notion that points are assigned to the same cluster if they are *density- reachable* from each other. To understand this concept, we have to go through the most important definitions used in DBSCAN and related algorithms first. The definitions and the presented pseudo code follows the original by [Ester *et al.*](#_bookmark47)([1996](#_bookmark47)), but are adapted to provide a more consistent presentation with the other algorithms discussed in this paper.

Clustering starts with a dataset *D* containing a set of points *p ∈ D*. Density-based algorithms need to obtain a density estimate over the data space. DBSCAN estimates the density around each point using the concept of *E*-neighborhood.

**Definition 1** *E*-neighborhood*. The E-neighborhood, NE*(*p*)*, of a data point p is the set of points within a specified radius E around p.*

*NE*(*p*) = *{q ∈ D | d*(*p, q*) *< E}*

*where d is some distance measure and E ∈* R+*. Note that together with p ∈ D this definition implies that point p is always part of its own E-neighborhood, i.e., p ∈ NE*(*p*) *always holds.*

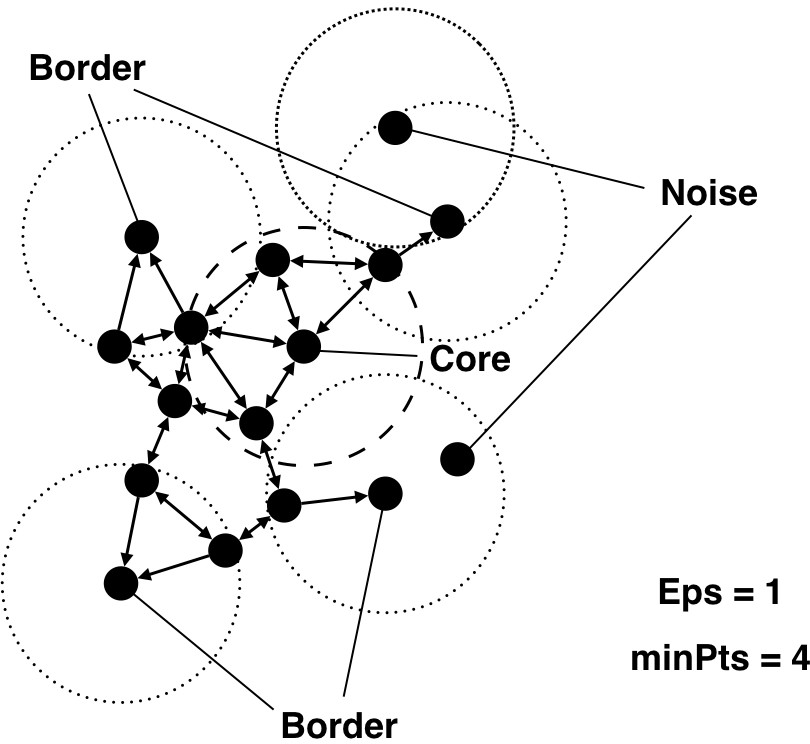
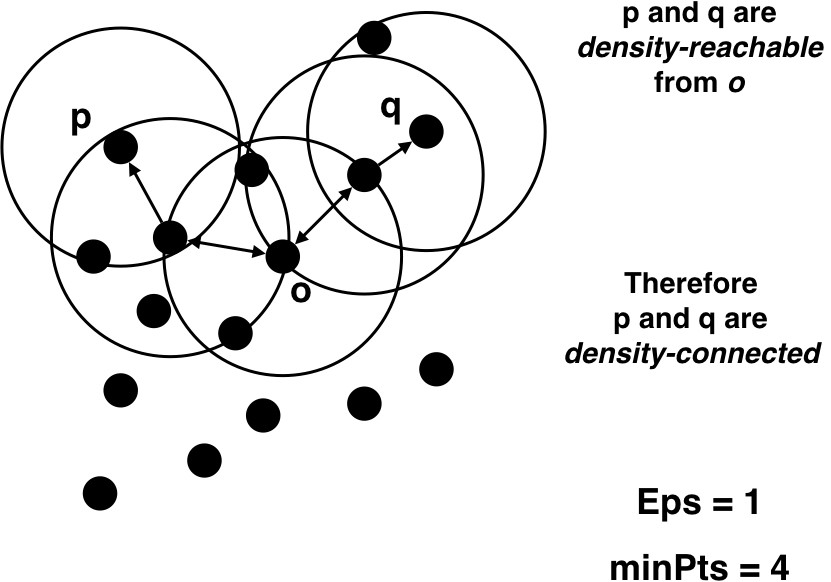
Following this definition, the size of the neighborhood *|NE*(*p*)*|* can be seen as a simple un- normalized kernel density estimate around *p* using a uniform kernel with a bandwidth of *E*. DBSCAN uses *NE*(*p*) and a threshold called *minPts* to detect dense regions and to classify the points in a dataset into *core*, *border*, or *noise* points.

**Definition 2** Point classes*. A point p ∈ D is classified as*

* *a* core point *if NE*(*p*) *has high density, i.e., |NE*(*p*)*| ≥ minPts where minPts ∈* Z+ *is a user-specified density threshold,*
* *a* border point *if p is not a core point, but it is in the neighborhood of a core point*

*q ∈ D, i.e., p ∈ NE*(*q*)*, or*

* *a* noise point*, otherwise.*

(a) (b)

Figure 1: Concepts used in the DBSCAN family of algorithms. (a) shows examples for the three point classes, core, border, and noise points, (b) illustrates the concept of density- reachability and density-connectivity.

An example of different point classes is shown in Figure [1](#_bookmark1)(a). The size of the neighborhood for some points is shown as a circle, and their class is shown as an annotation.

To form contiguous dense regions from individual points, DBSCAN defines the notions of reachability and connectedness.

**Definition 3** Directly density-reachable*. A point q ∈ D is directly density-reachable from a point p ∈ D with respect to E and minPts if, and only if,*

* + 1. *|NE*(*p*)*| ≥ minPts, and*
    2. *q ∈ NE*(*p*)*.*

*That is, p is a core point and q is in its E-neighborhood.*

**Definition 4** Density-reachable*. A point p is density-reachable from q if there exists in D an ordered sequence of points* (*p*1*, p*2*, . . . , pn*) *with q* = *p*1 *and p* = *pn such that pi*+1 *directly density-reachable from pi ∀ i ∈ {*1*,* 2*, . . . , n −* 1*}.*

**Definition 5** Density-connected*. A point p ∈ D is density-connected to a point q ∈ D if there is a point o ∈ D such that both p and q are density-reachable from o.*

Figure [1](#_bookmark1)(b) gives examples of density-reachable and density connected points.

The notion of density-connection can be used to form clusters as contiguous dense regions.

**Definition 6** Density-based cluster*. A density-based cluster C is a non-empty subset of D*

*satisfying the following conditions:*

1. Maximality*: If p ∈ C and q is density-reachable from p, then q ∈ C.*
2. Connectivity*: ∀ p, q ∈ C, p is density-connected to q.*

The DBSCAN algorithm identifies all such clusters by systematically finding all core points and expanding each to all density-reachable points. The algorithm begin with an arbitrary point *p* and retrieves its *E*-neighborhood. If it is a core point, then it will start a new cluster that is expanded by assigning all points in its neighborhood to the cluster. If an additional core point is found in the neighborhood, then the search is expanded to include also all points in its neighborhood. If no more core points are found in the expanded neighborhood, then the cluster is complete, and the remaining points are searched to see if another core point can be found to start a new cluster. After processing all points, points which were not assigned to a cluster have to be noise points.

Determining the appropriate values for the two parameters, *E* and *minPts*, is not always easy. The parameters depend on the dataset and influence each other. For example, increasing *minPts* typically also requires increasing *E* and vice versa. The clustering result is often also quite sensitive to small parameters changes. We present examples of how this issue can be addressed in Section [3.2](#_bookmark8).

In the DBSCAN algorithm, core points are always part of the same cluster, independent of the order in which the points in the dataset are processed. This is different for border points. Border points might be density-reachable from core points in several clusters and the original DBSCAN algorithm assigns them to the first of these clusters processed which depends on the order of the data points and the particular implementation of the algorithm. To alleviate this behavior, [Campello, Moulavi, Zimek, and Sander](#_bookmark38) ([2015](#_bookmark38)) suggest a modification called DBSCAN\* which considers all border points as noise instead and leaves them unassigned.

## OPTICS: Ordering points to identify clustering structure

The inability to find clusters of varying density is a notable drawback of DBSCAN result- ing from the fact that a combination of a specific neighborhood size with a single density threshold *minPts* is used to determine if a point resides in a dense neighborhood. There are many instances where it would be useful to detect clusters of varying density. From identifying regions of similar seawater characteristics ([Birant and Kut](#_bookmark34) [2007](#_bookmark34)), to network in- trusion detection systems ([Ertöz, Steinbach, and Kumar](#_bookmark46) [2003](#_bookmark46)), point of interest detection using geo-tagged photos ([Kisilevich, Mansmann, and Keim](#_bookmark66) [2010](#_bookmark66)), classifying cancerous skin lesions ([Celebi, Aslandogan, and Bergstresser](#_bookmark39) [2005](#_bookmark39)), the motivations for detecting clusters of varying densities are numerous.

In 1999, some of the authors of DBSCAN developed OPTICS ([Ankerst *et al.*](#_bookmark29)[1999](#_bookmark29)) to address this concern. OPTICS borrows the core density-reachable concept from DBSCAN. But while DBSCAN may be thought of as a clustering algorithm, searching for natural groups in data, OPTICS is an *augmented ordering algorithm* from which either flat or hierarchical clustering results can be derived. OPTICS requires the same *E* and *minPts* parameters as DBSCAN. However, the *E* parameter is theoretically unnecessary and is only used for the practical purpose of reducing the runtime complexity of the algorithm. To describe OPTICS, we introduce two additional concepts called *core-distance* and *reachability-distance*.

**Definition 7** Core-distance*. The core-distance of a point p ∈ D with respect to minPts and*

*E is defined as*

core-dist( ; *minPts*) = *UNDEFINED if |NE*(*p*)*| < minPts, and*

*p E,* f

*minPts*-*dist*(*p*) *otherwise.*

*where minPts*-*dist*(*p*) *is the distance from p to its minPts −* 1 *nearest neighbor, i.e., the minimal radius a neighborhood of size minPts centered at and including p would have.*

**Definition 8** Reachability-distance*. The reachability-distance of a point p ∈ D to a point*

*q ∈ D parameterized by E and minPts is defined as*

*p, q E,* f

reachability-dist( ; *minPts*) = *UNDEFINED if |NE*(*p*)*| < minPts, and*

max(core-dist(*p*)*, d*(*p, q*)) *otherwise.*

The reachability-distance of a core point *p* with respect to object *q* is the smallest neighbor- hood radius such that *p* would be directly density-reachable from *q*. Note that the parameters, although they have the same name, work differently than in DBSCAN. In OPTICS, *E* is typ- ically set to a very large value compared to DBSCAN. Therefore, OPTICS will considered more nearest neighbors in the core-distance calculation and *minPts* affects the smoothness of the reachability distribution, where larger values will lead to a smoother reachability distri- bution. This needs to be kept in mind when choosing appropriate parameters. It is worth noting that the *E* parameter is strictly there for computational reasons—it is used to restrict the number of points considered in the neighborhood search. It can safely be set to the max- imum *k*-nearest neighbor distance, where *k* = *minPts*, and achieve the same result as if *E*

were set to *∞*.

OPTICS provides an augmented ordering. The algorithm starts with a point and expands its neighborhood like DBSCAN, but it explores new points in the order of lowest to highest core-distance. The order in which the points are explored along with each point’s core- and reachability-distance is the final result of the algorithm. An example result of OPTICS is shown in the form of a reachability plot in Figure [2](#_bookmark3). The data points are shown in the resulting order on the *x*-axis, while the reachability-distance of each point is shown on the *y*-axis. Low reachability-distances are shown as valleys represent clusters separated by peaks representing points with larger distances. This density representation essentially conveys similar information as a dendrogram, a tree structure often used to represent the result of hierarchical clustering. This is why OPTICS is often also presented as a visualization tool. [Sander, Qin, Lu, Niu, and Kovarsky](#_bookmark80) ([2003](#_bookmark80)) showed how the output of OPTICS could be converted into an equivalent dendrogram, and that under certain conditions, the dendrogram produced by hierarchical clustering with single linkage is equivalent to running OPTICS with *minPts* = 2. [Ankerst *et al.*](#_bookmark29)([1999](#_bookmark29)) discuss two ways to group points into clusters based on the order discovered by OPTICS. We will refer to these as the ExtractDBSCAN method and the Extract-*ξ* method summarized below:

* + 1. *ExtractDBSCAN* uses a single global reachability-distance threshold *El* to extract a clustering. This can be seen as a horizontal line in the reachability plot in Figure [2](#_bookmark3). Peaks above the cut-off represent noise points and separate the clusters.

0.04

**Reachability Plot**

Reachability dist.

0.08

0.12

0 100 200 300 400

Order

Figure 2: OPTICS reachability plot example for a dataset with four clusters of 100 data points each.

* + 1. *Extract-ξ* identifies clusters *hierarchically* by scanning through the ordering that OP- TICS produces to identify significant, relative changes in reachability-distance. The authors of OPTICS noted that clusters could be thought of as identifying ‘dents’ in the reachability plot.

The ExtractDBSCAN method extracts a clustering similar to DBSCAN\* (i.e., DBSCAN where border points stay unassigned). Because this method extracts clusters like DBSCAN, it is hard to identify partitions that exhibit very significant differences in density. Clusters of significantly different density can only be identified if the data is well separated and very little noise is present. The second method, Extract-*ξ*1, identifies a cluster hierarchy and replaces the data dependent global *El* used in ExtractDBSCAN parameter with *ξ*, a data-independent density-threshold parameter ranging between 0 and 1. One interpretation of *ξ* is that it describes the relative magnitude of the change of cluster density (i.e., reachability). Significant changes in relative reachability allow for clusters to manifest themselves hierarchically as dents in the ordering structure. Extract-*ξ* finds a hierarchical clustering which allows us to find clusters of varying densities.

With its two ways of extracting clusters from the ordering, whether through either the global *El* or relative *ξ* threshold, OPTICS can be seen as a generalization of DBSCAN. For applica- tions where one needs to find clusters of similar density, OPTICS’s ExtractDBSCAN yields a DBSCAN-like solution, while for other applications, Extract-*ξ* can generate a hierarchy rep- resenting clusters of varying density. It is thus interesting to note that while DBSCAN has reached critical acclaim, even motivating numerous extensions ([Rehman, Asghar, Fong, and](#_bookmark77) [Sarasvady](#_bookmark77) [2014](#_bookmark77)), OPTICS has received decidedly less attention. Perhaps one of the reasons for this is because the powerful Extract-*ξ* method for grouping points into clusters has gone largely unnoticed, as it is not implemented in most open-source software packages that adver- tise an implementation of OPTICS. This includes implementations in WEKA ([Hall, Frank,](#_bookmark57) [Holmes, Pfahringer, Reutemann, and Witten](#_bookmark57) [2009](#_bookmark57)), SPMF ([Fournier-Viger *et al.*](#_bookmark48)[2014](#_bookmark48)), and

1In the original OPTICS publication ([Ankerst *et al.*](#_bookmark29)[1999](#_bookmark29)), the algorithm was outlined in Figure 19 and called the *ExtractClusters* algorithm, where the clusters extracted were referred to as *ξ*-clusters. To distinguish the method uniquely, we refer to it in dbscan as the Extract-*ξ* method.

*Library/Package DBSCAN OPTICS ExtractDBSCAN Extract-ξ*

dbscan ✓ ✓ ✓ ✓

ELKI ✓ ✓ ✓ ✓

SPMF ✓ ✓ ✓

PyClustering ✓ ✓ ✓

WEKA ✓ ✓ ✓

SciKit-Learn ✓

fpc ✓

*Library/Package Index Acceleration Dendrogram for OPTICS Language*

dbscan ✓ ✓ R

ELKI ✓ ✓ Java

SPMF ✓ Java

PyClustering ✓ Python

WEKA Java

SciKit-Learn ✓ Python

fpc R

Table 1: A comparison of DBSCAN and OPTICS implementations in various open-source statistical software libraries and packages. A ✓ symbol denotes availability.

the PyClustering ([Novikov](#_bookmark72) [2019](#_bookmark72)) and SciKit-Learn ([Pedregosa *et al.*](#_bookmark74)[2011](#_bookmark74)) libraries for Python ([Van Rossum *et al.*](#_bookmark84)[2011](#_bookmark84)). To the best of our knowledge, the only other open-source library currently providing a complete implementation of OPTICS is ELKI ([Schubert, Koos, Emrich,](#_bookmark81) [Züfle, Schmid, and Zimek](#_bookmark81) [2015](#_bookmark81)), written in Java ([Gosling, Joy, Steele, and Bracha](#_bookmark53) [2000](#_bookmark53)).

In fact, perhaps due to the (incomplete) implementations of OPTICS cluster extraction across various software libraries, there has been some confusion regarding the usage of OPTICS, and the benefits it offers compared to DBSCAN. Several papers motivate DBSCAN extensions or devise new algorithms by citing OPTICS as incapable of finding density-heterogeneous clusters ([Ghanbarpour and Minaei](#_bookmark51) [2014](#_bookmark51); [Chowdhury, Mollah, and Rahman](#_bookmark42) [2010](#_bookmark42); [Gupta, Liu,](#_bookmark54) [and Ghosh](#_bookmark54) [2010](#_bookmark54); [Duan, Xu, Guo, Lee, and Yan](#_bookmark44) [2007](#_bookmark44)). Along the same line of thought, others cite OPTICS as capable of finding clusters of varying density, but either use the DBSCAN-like global density threshold extraction method or refer to OPTICS as a clustering algorithm, without mention of which cluster extraction method was used in their experimen- tation ([Verma, Srivastava, Chack, Diswar, and Gupta](#_bookmark86) [2012](#_bookmark86); [Roy and Bhattacharyya](#_bookmark78) [2005](#_bookmark78); [Liu, Zhou, and Wu](#_bookmark68) [2007](#_bookmark68); [Pei, Jasra, Hand, Zhu, and Zhou](#_bookmark75) [2009](#_bookmark75)). However, OPTICS returns an ordering of the data points which can be post-processed to extract either (1) a flat clus- tering with clusters of relatively similar density or (2) a cluster hierarchy, which is adaptive to representing local densities within the data. To clear up this confusion, it seems to be important to add complete implementations to existing software packages and introduce new complete implementations of OPTICS like the R package dbscan described in this paper.

## Current implementations of DBSCAN and OPTICS

Implementations of DBSCAN and OPTICS are available in many statistical software pack- ages. We focus here on open-source solutions. These include the Waikato Environment for Knowledge Analysis (WEKA, [Hall *et al.*](#_bookmark57)[2009](#_bookmark57)), the Sequential Pattern Mining Framework

(SPMF, [Fournier-Viger *et al.*](#_bookmark48)[2014](#_bookmark48)), the Environment for Developing KDD-Application sup- ported by Index Structures (ELKI, [Schubert *et al.*](#_bookmark81)[2015](#_bookmark81)), the Python library SciKit-Learn ([Pedregosa *et al.*](#_bookmark74)[2011](#_bookmark74)), the PyClustering Data Mining library ([Novikov](#_bookmark72) [2019](#_bookmark72)), the Flexible Procedures for Clustering R package ([Hennig](#_bookmark58) [2019](#_bookmark58)), and the dbscan package ([Hahsler and](#_bookmark56) [Piekenbrock](#_bookmark56) [2019](#_bookmark56)) introduced in this paper. Table [1](#_bookmark5) presents a comparison of the features offered by these packages. All packages support DBSCAN and most use some form of index acceleration to speed up the *E*-neighborhood queries involved in both DBSCAN and OPTICS algorithms, the known bottleneck that typically dominates the runtime and is essential for processing larger datasets. dbscan supports index acceleration for Euclidean distance using *k*- d trees with 5 different splitting methods (see [Mount and Arya](#_bookmark71) [2010](#_bookmark71) for more details). Other software libraries mentioned also may support a variety of other distance measures and other methods of index acceleration. OPTICS with ExtractDBSCAN is also widely implemented, but the Extract-*ξ* method, as well as the use of dendrograms with OPTICS, are features currently only available in dbscan and ELKI.

1. **The** dbscan **package**

The package dbscan provides high performance code for DBSCAN and OPTICS through a C++ implementation (interfaced via the Rcpp package by [Eddelbuettel and François](#_bookmark45) [2011](#_bookmark45)) using the *k*-d tree data structure implemented in the C++ library ANN ([Mount and Arya](#_bookmark71) [2010](#_bookmark71)) to improve *k* nearest neighbor (kNN) and fixed-radius nearest neighbor search speed. DBSCAN and OPTICS share a similar interface.

dbscan(x, eps, minPts = 5, weights = NULL, borderPoints = TRUE, ...) optics(x, eps, minPts = 5, ...)

The first argument x is the dataset in form of a data.frame or a matrix. The implemen- tations use by default Euclidean distance for neighborhood computation. Alternatively, a precomputed set of pair-wise distances between data points stored in a dist object can be supplied. Using precomputed distances, arbitrary distance metrics can be used, however, note that *k*-d trees are not used for distance data, but lists of nearest neighbors are precomputed. For dbscan() and optics(), the parameter eps represents the radius of the *E*-neighborhood considered for density estimation and minPts represents the density threshold to identify core points. Note that eps is not strictly necessary for OPTICS but is only used as an upper limit for the considered neighborhood size used to reduce computational complexity. dbscan() also can use weights for the data points in x. The density in a neighborhood is calculated as the sum of the weights of the points inside the neighborhood. By default, each data point has a weight of one, so the density estimate for the neighborhood is the number of data points inside the neighborhood. Using weights, the importance of points can be changed.

The original DBSCAN implementation assigns border points to the first cluster it is density- reachable from. Since this may result in different clustering results if the data points are processed in a different order, [Campello *et al.*](#_bookmark38)([2015](#_bookmark38)) suggest for DBSCAN\* to consider all border points as noise. This can be achieved by using borderPoints = FALSE. All functions accept additional arguments. These arguments are passed on to the fixed-radius nearest neighbor search. More details about the implementation of the nearest neighbor search will be presented in Section [3.1](#_bookmark7) below.

To extract clusters from the linear order produced by OPTICS, dbscan implements the cluster extraction methods for ExtractDBSCAN and Extract-*ξ*:

extractDBSCAN(object, eps\_cl)

extractXi(object, xi, minimum = FALSE, correctPredecessor = TRUE)

extractDBSCAN() extracts a clustering from an OPTICS ordering that is similar to what DBSCAN would produce with a single global *El* set to eps\_cl. extractXi() extracts clusters hierarchically based on the steepness of the reachability plot. minimum controls whether only the minimal (non-overlapping) cluster are extracted. correctPredecessor corrects a common artifact known of the original *ξ* method presented in [Ankerst *et al.*](#_bookmark29)([1999](#_bookmark29)) by pruning the steep up area for points that have predecessors not in the cluster (see technical note in Appendix [A](#_bookmark88) for details).

## Nearest neighbor search

The density-based algorithms in dbscan rely heavily on forming neighborhoods, i.e., finding all points belonging to an *E*-neighborhood. A simple approach is to perform a linear search, i.e., always calculating the distances to all other points to find the closest points. This requires *O*(*n*) operations, with *n* being the number of data points, for each time a neighborhood is needed. Since DBSCAN and OPTICS process each data point once, this results in a runtime complexity of *O*(*n*2). A naive solution is to compute a distance matrix with all pairwise distances between points and sort the distances for each point (row in the distance matrix) to precompute the nearest neighbors for each point. However, this method has the drawback that the size of the full distance matrix is on the order of *O*(*n*2), and becomes very large for medium to large datasets.

To avoid computing the complete distance matrix, dbscan relies on a space-partitioning data structure called a *k*-d tree ([Bentley](#_bookmark33) [1975](#_bookmark33)). This data structure allows dbscan to identify the kNN or all neighbors within a fixed radius *eps* more efficiently in sub-linear time using on average only *O*(*log*(*n*)) operations per query. This results in a reduced runtime complexity of *O*(*nlog*(*n*)) for DBSCAN and OPTICS. However, note that *k*-d trees are known to degenerate for high-dimensional data requiring *O*(*n*) operations and leading to a performance no better than linear search. Fast kNN search and fixed-radius nearest neighbor search are used in DBSCAN and OPTICS, but we also provide a direct interface in dbscan, since they are useful in their own right.

kNN(x, k, sort = TRUE, search = "kdtree", bucketSize = 10, splitRule = "suggest", approx = 0)

frNN(x, eps, sort = TRUE, search = "kdtree", bucketSize = 10, splitRule = "suggest", approx = 0)

The interfaces only differ in the way that kNN() requires to specify k while frNN() needs the radius eps. All other arguments are the same. x is the data, and the result will be a list with the neighbors for each point in x. sort controls if the returned points are sorted by the distance. search controls what searching method should be used. Available search methods are "kdtree", "linear" and "dist". The linear search method does not build a search data structure but performs a complete linear search to find the nearest neighbors. The dist method

precomputes a dissimilarity matrix which is very fast for small datasets, but problematic for large sets. The default method is to build a *k*-d tree. *k*-d trees are implemented in C++ using a modified version of the ANN library ([Mount and Arya](#_bookmark71) [2010](#_bookmark71)) compiled for Euclidean distances. Parameters bucketSize, splitRule and approx are algorithmic parameters which control the way the *k*-d tree is built. bucketSize controls the maximal size of the *k*-d tree leaf nodes. splitRule specifies the method how the *k*-d tree partitions the data space. We use "suggest", which uses the best guess of the ANN library given the data. approx greater than zero uses approximate *NN* search to significantly speed up search, however, some actual neighbors may be omitted. Note that using this feature for DBSCAN or OPTICS is discouraged since it will lead to incorrect results by introducing spurious clusters and noise points. For more details, we refer the reader to the documentation of the ANN library ([Mount](#_bookmark71) [and Arya](#_bookmark71) [2010](#_bookmark71)). dbscan() and optics() use internally frNN() and the additional arguments in ... are passed on to the nearest neighbor search method.

## Clustering with DBSCAN

In this section, we present how to use DBSCAN, find appropriate values for the two parame- ters, and evaluate parameter sensitivity. As an example, we use a very simple artificial dataset of four slightly overlapping Gaussians in two-dimensional space with 100 points each. We load dbscan, set the seed of the random number generator to make the results reproducible and create the dataset.

*R> library("dbscan") R> set.seed(2)*

*R> n <- 400*

*R> x <- cbind(x = runif(4, 0, 1) + rnorm(n, sd = 0.1),*

*+ y = runif(4, 0, 1) + rnorm(n, sd = 0.1)) R> true\_clusters <- rep(1:4, time = 100)*

*R> plot(x, col = true\_clusters, pch = true\_clusters)*

The resulting dataset is shown in Figure [3](#_bookmark9).

To apply DBSCAN, we need to decide on the neighborhood radius eps and the density threshold minPts. The rule of thumb for setting minPts is to use at least the number of dimensions of the dataset plus one. In our case, this is 3. To find a suitable value for eps, we can plot the points’ kNN distances (i.e., the distance of each point to its *k*-th nearest neighbor) in decreasing order and look for a knee in the plot. The idea behind this heuristic is that points located inside of clusters will have a small *k*-nearest neighbor distance, because they are close to other points in the same cluster, while noise points are more isolated and will have a rather large kNN distance. dbscan provides a function called kNNdistplot() to make this easier. For *k* we use the minPts value of 3.

*R> kNNdistplot(x, k = 3)*

*R> abline(h = 0.05, col = "red", lty = 2)*

The kNN distance plot is shown in Figure [4](#_bookmark10). A knee is visible at around a 3-NN distance of

0.05. We have added a horizontal line manually for reference. Now we can perform the clustering with the chosen parameters.

0.00

y

0.6

0.8

1.0

●

●

● ● ● ●●

●●

●● ●

●●●●●●● ● ●● ●

●● ●●●

●

● ● ● ● ● ● ●

●● ●

●

●●●●● ● ●● ●●●●

●●●●● ●● ●

● ● ●●●●

● ● ●● ●●

● ● ●

● ●●● ●

●

●

● ●

●●●

●

●

0.0 0.2 0.4 0.6 0.8

0.0

0.2

0.4

x

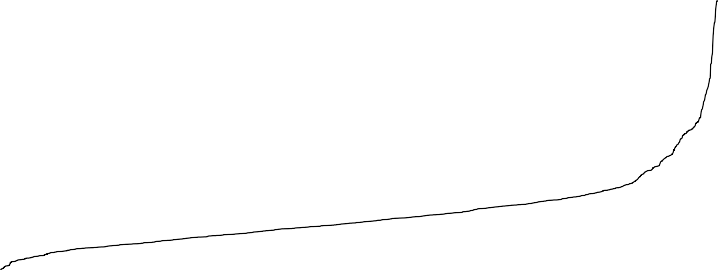
Figure 3: The sample dataset, consisting of 4 noisy Gaussian distributions with slight overlap.

3−NN distance

0.05

0.10

0.15



0 200 400 600 800 1000 1200

Points (sample) sorted by distance

Figure 4: *k*-nearest neighbor distance plot.

*R> res <- dbscan(x, eps = 0.05, minPts = 3) R> res*

DBSCAN clustering for 400 objects. Parameters: eps = 0.05, minPts = 3

The clustering contains 6 cluster(s) and 30 noise points.

|  |  |  |
| --- | --- | --- |
| 0 1 2 3 4 | 5 | 6 |
| 30 185 87 89 3 | 3 | 3 |

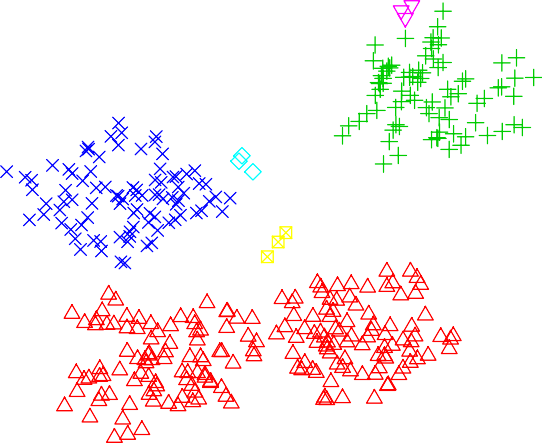
Available fields: cluster, eps, minPts

y

0.6

0.8

1.0



● ●

●

●

●

●

●

●

●

●

●

●

●

●

●

● ●

● ●●

●

●

●

●

●●

●●

●

0.0 0.2 0.4 0.6 0.8

0.0

0.2

0.4

x

Figure 5: Result of clustering with DBSCAN. Noise is represented as black circles.

The resulting clustering identifies one large cluster with 185 member points and two medium size clusters of between 87 and 89 points, three very small clusters and 30 noise points (repre- sented by cluster id 0). The available fields can be directly accessed using the list extraction operator $. For example, the cluster assignment information can be used to plot the data with the clusters identified by different labels and colors.

*R> plot(x, col = res$cluster + 1L, pch = res$cluster + 1L)*

Note that we add one to the cluster labels since noise points have a cluster label of zero and thus would not be visible in the plot. The resulting scatter plot in Figure [5](#_bookmark11) shows that the clustering algorithm correctly identified the upper two clusters, but merged the lower two clusters because the region between them has a high enough density. The small clusters are isolated groups of 3 points (passing *minPts*) and the noise points (black circles) are isolated points. dbscan also provides a plot that adds convex cluster hulls to the scatter plot shown in Figure [6](#_bookmark12).

*R> hullplot(x, res)*

A clustering can also be used to find out to which clusters new data points would be assigned using predict(object, newdata = NULL, data, ...). The predict method uses nearest neighbor assignment to core points and needs the original dataset. Additional parameters are passed on to the nearest neighbor search method. Here we obtain the cluster assignment for the first 25 data points. Note that an assignment to cluster 0 means that the data point is considered noise because it is not in the *E*-neighborhood of any core point.

*R> predict(res, x[1:25, ], data = x)*

[1] 1 2 1 0 1 2 1 3 1 2 1 3 1 0 1 3 1 2 0 3 1 2 1 3 1

y

0.0

0.2

0.4

### Convex Cluster Hulls

0.6

0.8

1.0



●

●

● ●

●

●

●

● ●●

●

●●

●

●●

●

●

●● ●

●

●●●

●● ● ●

● ●

● ●●●●

●●

●

●●

●

●

● ●

● ●

●●

●

● ●●

●

● ●

●

●

●

●

● ● ●

●

●

●

● ● ● ● ●

● ● ●

●

● ●

●

●

●

●

●

●

●

●●

●

●

●●

● ● ● ● ●

●

●●

●

●●

●

● ●

●

●

●

●

●

●

●

● ● ●

●

●

● ●● ●

●

●●

●

●●

●

●

● ●

● ●

●●

●

●

●● ●● ●● ● ●

● ● ●

●

● ●

●

● ● ● ●

● ●

●● ●

●

●

●

● ●● ●

● ●

●

● ● ●

● ●

●

●

● ●● ● ●

● ●

●

●

●

● ●

●

●

●●

●

●

● ●

●

●

● ●

●

●

●

● ●

● ● ● ● ●

●

●

●

● ●

●

● ●

●

●

●● ●

●

●

●

●

●

●●

● ●

● ●● ● ● ● ● ●

● ●

●

● ● ● ● ●

●

●

● ●

● ● ●

●●

●

● ●

●

● ● ●

● ● ●●

●●

● ● ●

● ●

● ●

●

● ● ● ● ●

●●

● ●

●

● ● ●

●

●

●

●

●

●

● ●

● ● ● ● ●●

●

●

●

●

●

●

●●

●● ● ●

● ● ●● ●

●

● ●

●

● ●●

●

●● ●● ● ●

● ●

●

●

● ● ●

●

● ●

● ● ●

●

● ●●

●●

● ● ● ● ● ●●

● ●

●

●

●

●

●●

● ● ●

●

●

● ●

● ●

●

●

●

●

0.0 0.2 0.4 0.6 0.8

x

Figure 6: Convex hull plot of the DBSCAN clustering. Noise points are black. Note that density-based clusters do not need to be convex and thus noise points and points of another cluster may lie within the convex hull of a different cluster.

Even using the method based on the *k*-nearest neighbor distance plot described above, it is sometimes hard to find appropriate parameter values. This is in part because the parameters are interrelated and the algorithm can be very sensitive to small parameter changes. We show here how a simple sensitivity analysis can be performed. The idea is to create a grid of parameter combinations, cluster with each combination, and then calculate the clustering quality using a cluster validity measure. Here we have ground truth information (the correct cluster label) available, and therefore we can use the adjusted Rand index (ARI) ([Hubert and](#_bookmark59) [Arabie](#_bookmark59) [1985](#_bookmark59)), a well-known external cluster validity measure. ARI compares the fraction of pairs of objects in agreement across the ground truth partition and the partition given by the clustering to be evaluated. The index is adjusted for agreement by chance. In the absence of a ground truth, internal cluster validity measures can be used.

We start with a reasonable grid.

*R> minPts\_grid <- 1:20*

*R> eps\_grid <- seq(0.01, 0.2, by = 0.01)*

Next, we define a function that clusters the data given a set of parameters and returns the adjusted Rand index (defined in package mclust, [Scrucca *et al.*](#_bookmark82)[2016](#_bookmark82)). Note that the function uses the existing data, x, and the ground truth, true\_cluster, from the parent environment. Finally, we apply the (vectorized) function to the grid of parameters.

*R> library("mclust")*

Frequency

0

50

100

150

200

0.0 0.2 0.4 0.6 0.8 1.0

ARI

Figure 7: Histogram of the ARI values for ranges of settings of (minPts, eps).

*R> clust <- function(minPts, eps) {*

*+ res <- dbscan(x, eps = eps, minPts = minPts)*

*+ adjustedRandIndex(res$cluster, true\_clusters)*

*+ }*

*R> res\_mat <- outer(minPts\_grid, eps\_grid, FUN = Vectorize(clust)) R> dimnames(res\_mat) <- list(minPts\_grid, eps\_grid)*

The sensitivity of the performance of DBSCAN to parameter variations can be assessed using a histogram of the ARI values.

*R> hist(res\_mat, breaks = 20, xlab = "ARI", xlim = c(0, 1), main = NULL)*

The histogram in Figure [7](#_bookmark13) shows that many combinations of minPts and eps lead to an undesirable clustering result with low ARI. Only a very small number of combinations result in a reasonably high ARI of around 0*.*8 which indicates that the clustering captures much of the ground truth. The results of the sensitivity analysis can also be visualized as a color image to understanding how sensitive the DBSCAN algorithm is to parameter changes for the particular dataset. We use here the image plot provided in package seriation ([Hahsler,](#_bookmark55) [Hornik, and Buchta](#_bookmark55) [2008](#_bookmark55)).

*R> library("seriation")*

*R> pimage(res\_mat, xlab = "eps", ylab = "minPts", key.lab = "ARI")*

Figure [8](#_bookmark14) shows a band of higher ARI values where with increasing minPts, eps should be decreased. We can see that a good ARI value can be achieved by using minPts = 15 and eps

= 0.09.

1 0.8



2

3

4

5

6 0.6

7

8

9

minPts

10 0.4

ARI

11

12

13

14

15 0.2

16

17

18

19

20 0

0.01

0.02

0.03

0.04

0.05

0.06

0.07

0.08

0.09

0.1

0.11

0.12

0.13

0.14

0.15

0.16

0.17

0.18

0.19

0.2

Figure 8: Image plot of the ARI values for a grid of settings of minPts and eps.

*R> res <- dbscan(x, eps = 0.09, minPts = 15) R> res*

DBSCAN clustering for 400 objects. Parameters: eps = 0.09, minPts = 15

The clustering contains 4 cluster(s) and 36 noise points.

0 1 2 3 4

36 99 87 89 89

Available fields: cluster, eps, minPts

*R> hullplot(x, res)*

Figure [9](#_bookmark15) shows that the four clusters can be successfully identified with the parameters ob- tained using grid search with known ground truth.

## Clustering with OPTICS

Unless OPTICS is purely used to extract a DBSCAN clustering, its parameters have a different effect than for DBSCAN: eps is typically chosen rather large (we use ten here) and minPts mostly affects core and reachability-distance calculation, where larger values have a smoothing effect. We also use 10, i.e., the core-distance is defined as the distance to the 9th nearest neighbor (spanning a neighborhood of 10 points including the point itself).

*R> res <- optics(x, eps = 10, minPts = 10) R> res*

y

0.0

0.2

0.4

### Convex Cluster Hulls

0.6

0.8

1.0

●

●

● ●

●

●

●

● ●●

●

●●

●

●●

●

●

●● ●

●

●●●

●● ● ●

● ●

● ●●●●

●●

●

●●

●

●

● ●

● ●

●●

●

● ●●

●

● ●

●

●

●

●

● ● ●

●

●

●

● ● ● ● ●

● ● ●

●

● ●

●

●

●

●

●

●

●

●●

●

●

●●

● ● ● ● ●

●

●●

●

●●

●

● ●

●

●

●

●

●

●

●

● ● ●

●

●

● ●● ●

●

●●

●

●●

●

●

● ●

● ●

●●

●

●

●● ●● ●● ● ●

● ● ●

●

● ●

●

● ● ● ●

● ●

●● ●

●

●

●

● ●● ●

● ●

●

● ● ●

● ●

●

●

● ●● ● ●

● ●

●

●

●

● ●

●

●

●●

●

●

● ●

●

●

● ●

●

●

●

● ●

● ● ● ● ●

●

●

●

● ●

●

● ●

●

●

●● ●

●

●

●

●

●

●●

● ●

● ●● ● ● ● ● ●

● ●

●

● ● ● ● ●

●

●

● ●

● ● ●

●●

●

● ●

●

● ● ●

● ● ●●

●●

● ● ●

● ●

● ●

●

● ● ● ● ●

●●

● ●

●

● ● ●

●

●

●

●

●

●

● ●

● ● ● ● ●●

●

●

●

●

●

●

●●

●● ● ●

● ● ●● ●

●

● ●

●

● ●●

●

●● ●● ● ●

● ●

●

●

● ● ●

●

● ●

● ● ●

●

● ●●

●●

● ● ● ● ● ●●

● ●

●

●

●

●

●●

● ● ●

●

●

● ●

● ●

●

●

●

●

0.0 0.2 0.4 0.6 0.8

x

Figure 9: Convex hull plot of a DBSCAN clustering using minPts = 15 and eps = 0.09.

OPTICS ordering/clustering for 400 objects.

Parameters: minPts = 10, eps = 10, eps\_cl = NA, xi = NA Available fields: order, reachdist, coredist, predecessor,

minPts, eps, eps\_cl, xi

OPTICS is an augmented ordering algorithm, which stores the computed order of the points it found in the order element of the returned object.

*R> head(res$order, n = 15)*

[1] 1 363 209 349 337 301 357 333 321 285 281 253 241 177 153

This means that data point 1 in the dataset is the first in the order, data point 363 is the second and so forth. The density-based order produced by OPTICS can be directly plotted as a reachability plot.

*R> plot(res)*

The reachability plot in Figure [10](#_bookmark16) shows the reachability distance for points ordered by OPTICS. Valleys represent potential clusters separated by peaks. Very high peaks may indicate noise points. To visualize the order on the original datasets, we can plot a line connecting the points in order.

*R> plot(x, col = "grey") R> polygon(x[res$order, ])*

0.04

y

0.0

0.2

0.4

**Reachability Plot**

Reachability dist.

0.08

0.12

0 100 200 300 400

Order

Figure 10: OPTICS reachability plot. Note that the first reachability value is always UNDE- FINED.

0.6

0.8

1.0

● ●

●

●

●● ●

● ●

● ● ●●

●

●

●●

● ●●● ●

●

●●

●

●

●

●

●

●

●

●

● ● ●●● ● ●● ●● ●

●● ● ●

● ●

● ●

●●

●● ● ● ●

●●

●● ●

●● ● ● ●

●● ●

●

●

●●●● ●●

●●

●

●

● ●

●

●●● ● ●●●●

●●

●

●

● ●

●●● ● ●●● ●●

●

●

● ●●●●●●● ●●

●●● ●

● ●● ● ●

●● ●

●●●

● ●● ●●●●

●●●●●

●●● ●●●● ●

●

● ● ●

● ● ●

●

●

●

● ●

●

● ● ● ●●

●●

● ●

● ●

●●●●● ●● ●●

●●

●

●●●●●●● ● ●● ●

●● ●●●

●● ●●●●

● ●●●

● ●

●● ●

● ● ●● ● ● ●

● ●

● ● ● ●

● ● ● ●●●●●●● ●● ●●

●●●●● ●● ●

●

● ● ● ●

●●●●●● ●● ● ● ●● ●● ●●●

●

●

●●● ●●●

●●●●

● ●●

●●● ●●●●●● ●●

●

● ●

● ●

● ●●●●● ●

●● ●●

●

●

●

●

● ●

●

● ●

●●●

●●

●

●

●

0.0 0.2 0.4 0.6 0.8

x

Figure 11: OPTICS order of data points represented as a line.

Figure [11](#_bookmark17) shows that points in each cluster are visited in consecutive order starting with the points in the center (the densest region) and then the points in the surrounding area.

As noted in Section [2.2](#_bookmark2), OPTICS has two primary cluster extraction methods using the ordered reachability structure it produces. A DBSCAN-type clustering can be extracted using extractDBSCAN() by specifying the global neighborhood size *El*. The reachability plot in Figure [10](#_bookmark16) shows four peaks, i.e., points with a high reachability-distance. These points indicate boundaries between clusters four clusters. A threshold that separates the four clusters can be visually determined. In this case we use eps\_cl of 0.065.

*R> res <- extractDBSCAN(res, eps\_cl = 0.065)*

*R> res*

OPTICS ordering/clustering for 400 objects.

Parameters: minPts = 10, eps = 10, eps\_cl = 0.065, xi = NA The clustering contains 4 cluster(s) and 92 noise points.

0 1 2 3 4

92 81 84 72 71

Available fields: order, reachdist, coredist, predecessor,

minPts, eps, eps\_cl, xi, cluster

*R> plot(res)*

*R> hullplot(x, res)*

The resulting reachability and corresponding clusters are shown in Figures [12](#_bookmark18) and [13](#_bookmark19). The clustering resembles the original structure of the four clusters with which the data were generated closely, with the only difference that points on the boundary of the clusters are marked as noise points.

dbscan also provides extractXi() to extract a hierarchical cluster structure. We use here a

xi value of 0.05.

*R> res <- extractXi(res, xi = 0.05) R> res*

OPTICS ordering/clustering for 400 objects.

Parameters: minPts = 10, eps = 10, eps\_cl = NA, xi = 0.05 The clustering contains 7 cluster(s) and 1 noise points.

Available fields: order, reachdist, coredist, predecessor,

minPts, eps, eps\_cl, xi, cluster, clusters\_xi

The *ξ* method results in a hierarchical clustering structure, and thus points can be members of several nested clusters. Clusters are represented as contiguous ranges in the reachability plot and are available the field clusters\_xi.

*R> res$clusters\_xi*

start end cluster\_id

1 1 194 1

2 1 301 2

3 8 23 3

4 94 106 4

5 196 288 5

6 302 399 6

7 308 335 7

0.04

y

0.0

0.2

0.4

**Reachability Plot**

Reachability dist.

0.08

0.12

0 100 200 300 400

Order

Figure 12: Reachability plot for a DBSCAN-type clustering extracted at global *E* = 0*.*065 results in four clusters.

**Convex Cluster Hulls**

0.6

0.8

1.0

●

●

●

●

●

●

●

●

● ●

●

●

●

●

● ●

●

●

●

● ●

●

●●●●

● ● ●

●

● ●

● ●●● ●

●●

●

●●

●

●

● ●

●

●

● ●

●●

●

●

●

● ●

● ● ●

●

● ●

●

●

●

●

● ● ● ●

● ●

● ● ●

●

●

●

●

●

●

●●

● ●

● ●

●

●

●

●

●

● ● ● ● ●

●●

●

●

●●

●

●

●

●

●

●

●

●

●

● ● ●

●

●

●

●

●

●

● ●

● ●

● ●● ●

● ●●

● ●

●● ●

●● ●● ●● ● ●

● ●

● ● ● ●

●

●

●

●

● ● ●

●

●

●

● ● ● ●● ●

●

●

● ●

● ●

● ●

●

●

●

●

●

●

● ●● ● ●

●

●

●

●

●

● ●

●

●●

●

●

● ●

●

● ● ●

●

●

● ● ●

●

●

●

●

●

●

● ●

●

● ●

●

●●

●

●

●

●

●

●

●

●●

●

● ●

● ●

● ●

● ●● ● ● ● ●

● ● ●

●

● ●

●

●

●

●

● ● ●

●

● ●

●●

●

● ●

● ●● ●

● ● ● ● ● ● ●

●

●

●

●

●

●

● ●● ● ●

●

●

●

●

●

● ●

●●

● ●

●

●

● ● ●

●

● ●● ●

●

●

●

● ●

●

●● ●

●

●

●●

●

● ●

●● ●

●● ●

● ● ● ●

● ●

●

●● ●● ●

● ●

●

●

●

● ●

●

●

●

● ●

●

● ●

●

●

●

● ●

●

●● ●

● ● ●

●

●

● ●

●

● ● ● ●

●

●

● ●

●

●

●

● ●

●

0.0 0.2 0.4 0.6 0.8

x

Figure 13: Convex hull plot for a DBSCAN-type clustering extracted at global *E* = 0*.*065 results in four clusters.

Here we have seven clusters. The clusters are also visible in the reachability plot.

*R> plot(res)*

*R> hullplot(x, res)*

Figure [14](#_bookmark20) shows the reachability plot with clusters represented using colors and vertical bars below the plot. The clusters themselves can also be plotted with the convex hull plot function

y

0.0

0.2

0.4

**Reachability Plot**

Reachability dist.

0.08

0.12

Figure 14: Reachability plot of a hierarchical clustering extracted with Extract-*ξ*.

0.00

0.04

### Convex Cluster Hulls

0.6

0.8

1.0



● ●

● ●

●

●

●

● ●●

●

●●

●

●●

●

●

●● ●

●

●●●

●● ● ●

● ●

● ●●●●

● ●●

●

●

●●

● ●●

●

●

●

●

● ●

● ●

●●

●

● ● ● ● ●

●

●

●

●

● ● ● ● ●

● ● ●

●

● ●

●

●

●

●●

●

●●

●

●

●

●

●

●

●●

●

● ● ● ● ●

●●

●

● ●

●

●

●

●

●

●

● ● ●

●

●

●●

●

●

●

● ●● ●

●

●

●

● ●

● ●●

●●

●

●

●● ●● ●● ● ●

● ● ●

●

● ●

● ● ● ●

● ● ●

●● ●

●

●

●

● ●

● ● ●

● ●

●

● ●● ●

●

●

● ●● ● ●

● ●

●

●

●

● ●

●

●

●●

●

●

● ●

●

●

●

● ● ● ●

● ● ●

●

●

●

●

●

●

●

● ●

●

● ●

●

●

●● ●

●

●

●

●

●

●

●●

● ●

●

● ●

● ●● ● ● ● ● ● ●

● ● ● ● ●

● ●

●

● ●

● ● ●

●●

●●

●

● ● ●

● ● ●●

● ● ●

● ●

● ●

●

● ●

● ● ●

●●

●

● ●

●

● ● ●

●

●

●

●

● ●

● ● ●● ● ●

● ●

●

● ●

● ● ● ● ●●

●

●

●

●●

●

● ● ●● ●

● ●●

●

●● ●● ● ●

● ●

●

●

● ● ●

●

● ●

● ● ●

●

● ●●

●●

● ● ● ● ● ●●

● ●

●

●

●

●

●●

● ● ●

●

●

● ●

● ●

●

●

●

●

0.0 0.2 0.4 0.6 0.8

x

Figure 15: Convex hull plot of a hierarchical clustering extracted with Extract-*ξ*.

shown in Figure [15](#_bookmark21). Note how the nested structure is shown by clusters inside of clusters. Also note that it is possible for the convex hull, while useful for visualizations, to contain a point that is not considered as part of a cluster grouping.

## Reachability and dendrograms

Reachability plots can be converted into equivalent dendrograms ([Sander *et al.*](#_bookmark80)[2003](#_bookmark80)). db- scan contains a fast implementation of the reachability-to-dendrogram conversion algorithm

0.12

Figure 16: Dendrogram structure of OPTICS reordering.

Reachability dist.

00

0.04

0.08

through an implementation of the disjoint-set data structure ([Cormen, Leiserson, Rivest, and](#_bookmark43) [Stein](#_bookmark43) [2001](#_bookmark43); [Patwary, Blair, and Manne](#_bookmark73) [2010](#_bookmark73)), allowing the user to choose which hierarchi- cal representation they prefer. The conversion algorithm can be directly called for OPTICS objects using the coercion method as.dendrogram().

*R> dend <- as.dendrogram(res)*

*R> dend*

'dendrogram' with 2 branches and 400 members total, at height 0.1363267

The dendrogram can be plotted using the standard plot method.

*R> plot(dend, ylab = "Reachability dist.", leaflab = "none")*

Note how the dendrogram in Figure [16](#_bookmark22) closely resembles the reachability plots with added binary splits. Since the object is a standard dendrogram (from package stats), it can be used like any other dendrogram created with hierarchical clustering.

# Performance comparison

Finally, we compare the performance of dbscan’s implementation of DBSCAN and OPTICS with other open-source implementations. This is not a comprehensive performance evaluation study but is used rather to give the reader an idea about the performance of different DB- SCAN and OPTICS implementations on datasets of varying sizes and number of dimensions. A comparative test was performed using both DBSCAN and OPTICS algorithms, where sup- ported, for the libraries listed in Table [1](#_bookmark5) on page [8](#_bookmark5). The used datasets and their sizes are listed in Table [2](#_bookmark24). It is worth noting that we only consider Euclidean distance here. The datasets tested include s1 and s2, the randomly generated but moderately-separated Gaussian clusters often used for agglomerative cluster analysis ([Fränti and Virmajoki](#_bookmark49) [2006](#_bookmark49)), the R15 validation dataset used for the maximum variance based clustering approach by [Veenman, Reinders, and](#_bookmark85) [Backer](#_bookmark85) ([2002](#_bookmark85)), the well-known spatial dataset DS3 used for validation of the CHAMELEON

|  |  |  |  |
| --- | --- | --- | --- |
| *Data set* | *Size* | *Dimensionality* | *Type* |
| Aggregation | 788 | 2 | Benchmark |
| Compound | 399 | 2 | Benchmark |
| D31 | 3,100 | 2 | Benchmark |
| flame | 240 | 2 | Benchmark |
| jain | 373 | 2 | Benchmark |
| pathbased | 300 | 2 | Benchmark |
| R15 | 600 | 2 | Benchmark |
| s1 | 5,000 | 2 | Benchmark |
| s4 | 5,000 | 2 | Benchmark |
| spiral | 312 | 2 | Benchmark |
| t4.8k | 8,000 | 2 | Benchmark |
| CG6d\_5k | 5,000 | 6 | Synthetic (Correlated) |
| CG6d\_10k | 10,000 | 6 | Synthetic (Correlated) |
| CG6d\_15k | 15,000 | 6 | Synthetic (Correlated) |
| UG6d\_5k | 5,000 | 6 | Synthetic (Uncorrelated) |
| UG6d\_10k | 10,000 | 6 | Synthetic (Uncorrelated) |
| UG6d\_15k | 15,000 | 6 | Synthetic (Uncorrelated) |
| CG12d\_5k | 5,000 | 12 | Synthetic (Correlated) |
| CG12d\_10k | 10,000 | 12 | Synthetic (Correlated) |
| CG12d\_15k | 15,000 | 12 | Synthetic (Correlated) |
| UG12d\_5k | 5,000 | 12 | Synthetic (Uncorrelated) |
| UG12d\_10k | 10,000 | 12 | Synthetic (Uncorrelated) |
| UG12d\_15k | 15,000 | 12 | Synthetic (Uncorrelated) |
| CG18d\_5k | 5,000 | 18 | Synthetic (Correlated) |
| CG18d\_10k | 10,000 | 18 | Synthetic (Correlated) |
| CG18d\_15k | 15,000 | 18 | Synthetic (Correlated) |
| UG18d\_5k | 5,000 | 18 | Synthetic (Uncorrelated) |
| UG18d\_10k | 10,000 | 18 | Synthetic (Uncorrelated) |
| UG18d\_15k | 15,000 | 18 | Synthetic (Uncorrelated) |

Table 2: Datasets used for comparison.

algorithm ([Karypis, Han, and Kumar](#_bookmark64) [1999](#_bookmark64)), along with a variety of shape datasets commonly found in clustering validation papers ([Gionis, Mannila, and Tsaparas](#_bookmark52) [2007](#_bookmark52); [Zahn](#_bookmark87) [1971](#_bookmark87); [Chang](#_bookmark40) [and Yeung](#_bookmark40) [2008](#_bookmark40); [Jain and Martin](#_bookmark60) [2005](#_bookmark60); [Fu and Medico](#_bookmark50) [2007](#_bookmark50)). To test higher-dimensional data, synthetic data was generated from both uncorrelated and correlated Gaussian sources (correlation has been shown to affect the performance of *k*-d trees in higher dimensions ([Arya](#_bookmark31) [and Mount](#_bookmark31) [1993](#_bookmark31))). The names of these datasets are abbreviated with whether they were generated from a correlated Gaussian (CG) or uncorrelated Gaussian (UG) source, followed by the dimensionality of the dataset, and the size of the dataset. For example, the dataset UG6d\_10k refers to a 6-dimensional 10,000 point dataset generated from an uncorrelated multivariate Gaussian source distribution. The correlation between each dimension within the correlated datasets was randomly (uniformly) chosen between -1 and 1.

We perform a comparison with ELKI version 0.7, PyClustering 0.6.6, fpc 2.1-10, dbscan 0.9-8,

SPMF v2.10, WEKA 3.8.0, SciKit-Learn 0.17.1 on a MacBook Pro equipped with a 2.5 GHz

DBSCAN Benchmarks

Log scale

Library

**dbscan spmf scikit elki weka fpc**

**pycluster**

1000

Time (ms)

10

**jain**

**D31**

**s4**

Dataset

**pathbased**

**spiral**

**flame**

**Compound**

**R15**

**Aggregation**

**synth1**

**synth2**

**synth3**

**s1**

**t4.8k**

Figure 17: Runtime of DBSCAN in milliseconds (*y*-axis, logarithmic scale) vs. the name of the dataset tested (*x*-axis).

OPTICS Benchmarks

Log scale

10000

Library

**dbscan elki weka spmf**

**pycluster**

Time (ms)

100

Dataset

**pathbased**

**spiral flame**

**jain**

**Compound**

**R15**

**Aggregation**

**synth2**

**synth1**

**synth3**

**D31**

**s1**

**s4**

**t4.8k**

Figure 18: Runtime of OPTICS in milliseconds (*y*-axis, logarithmic scale) vs. the name of the dataset tested (*x*-axis).

Intel Core i7 processor, running OS X El Capitan 10.11.6. All datasets are normalized to the unit interval, [0*,* 1], per dimension to standardize neighbor queries. For all datasets we use *minPts* = 2 and *E* = 0*.*10 for DBSCAN. For OPTICS, *minPts* = 2 with a large *E* = 1 is used. These parameters are explicitly chosen to be fairly large to test the scalability of the algorithms runtime performance. We replicate each run for each dataset 15 times and report the average runtime here. When possible, then we use the same parameters settings for libraries that support spatial-indexing acceleration (e.g., if it is possible to use a *k*-d tree,

we use the same leaf size of 10). In total, *≈* 3*,* 045 tests were run for DBSCAN and *≈* 2*,* 175

tests were run for OPTICS. Figures [17](#_bookmark25) and [18](#_bookmark26) show the runtime results. The datasets are sorted from fastest to slowest, averaged across all libraries tested. The results show that the average runtime varies by many orders of magnitude between implementations, and that dbscan compares very favorably with other implementations.

# Concluding remarks

The dbscan package offers a set of scalable, robust, and complete implementation s of popular density-based clustering algorithms from the DBSCAN family. The main features of dbscan are a simple interface to fast clustering and cluster extraction algorithms, extensible data structures and methods for both density-based clustering visualization and representation including efficient conversion algorithms between OPTICS ordering and dendrograms.

The density-based clustering field is still developing, and as new algorithms are introduced, we will try to incorporate them into dbscan. Initial versions of several algorithms related to density-based clustering have been added to dbscan recently. These algorithms include Hier- archical DBSCAN (HDBSCAN, [Campello, Moulavi, and Sander](#_bookmark36) [2013a](#_bookmark36)), Local Outlier Factor (LOF, [Breunig, Kriegel, Ng, and Sander](#_bookmark35) [2000](#_bookmark35)), Global-Local Outlier Scores from Hierarchies (GLOSH, [Campello *et al.*](#_bookmark38)[2015](#_bookmark38)), the Framework for the Optimal Selection of Clusters (FOSC, [Campello, Moulavi, Zimek, and Sander](#_bookmark37) [2013b](#_bookmark37)), and Jarvis-Patrick and Shared Nearest Neigh- bors clustering ([Jarvis and Patrick](#_bookmark62) [1973](#_bookmark62)). dbscan will continue to incorporate state-of-the-art algorithms and methods used for density-based clustering and related problems.

# Acknowledgments

This work is partially supported by industrial and government partners at the Center for Surveillance Research, a National Science Foundation I/UCRC.

# References

Aggarwal CC, Reddy CK (2013). *Data Clustering: Algorithms and Applications*. 1st edition.

Chapman & Hall/CRC.

Ankerst M, Breunig MM, Kriegel HP, Sander J (1999). “OPTICS: Ordering Points to Iden- tify the Clustering Structure.” In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pp. 49–60.

Arbelaitz O, Gurrutxaga I, Muguerza J, Pérez JM, Perona I (2013). “An Extensive Com- parative Study of Cluster Validity Indices.” *Pattern Recognition*, **46**(1), 243–256. [doi:](https://doi.org/10.1016/j.patcog.2012.07.021) [10.1016/j.patcog.2012.07.021](https://doi.org/10.1016/j.patcog.2012.07.021).

Arya S, Mount DM (1993). “Algorithms for Fast Vector Quantization.” In *Data Compression Conference DCC’93*, pp. 381–390.

Benaglia T, Chauveau D, Hunter D, Young D (2009). “mixtools: An R Package for Analyzing Finite Mixture Models.” *Journal of Statistical Software*, **32**(6), 1–29. [doi:10.18637/jss.](https://doi.org/10.18637/jss.v032.i06) [v032.i06](https://doi.org/10.18637/jss.v032.i06).

Bentley JL (1975). “Multidimensional Binary Search Trees Used for Associative Searching.”

*Communications of the ACM*, **18**(9), 509–517. [doi:10.1145/361002.361007](https://doi.org/10.1145/361002.361007).

Birant D, Kut A (2007). “ST-DBSCAN: An Algorithm for Clustering Spatial-Temporal Data.”

*Data & Knowledge Engineering*, **60**(1), 208–221. [doi:10.1016/j.datak.2006.01.013](https://doi.org/10.1016/j.datak.2006.01.013).

Breunig MM, Kriegel HP, Ng RT, Sander J (2000). “LOF: Identifying Density-Based Local Outliers.” In *Proceedings of the 2000 ACM SIGMOD International Conference on Man- agement of Data*, pp. 93–104.

Campello RJGB, Moulavi D, Sander J (2013a). “Density-Based Clustering Based on Hierar- chical Density Estimates.” In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 160–172.

Campello RJGB, Moulavi D, Zimek A, Sander J (2013b). “A Framework for Semi-Supervised and Unsupervised Optimal Extraction of Clusters from Hierarchies.” *Data Mining and Knowledge Discovery*, **27**(3), 344–371. [doi:10.1007/s10618-013-0311-4](https://doi.org/10.1007/s10618-013-0311-4).

Campello RJGB, Moulavi D, Zimek A, Sander J (2015). “Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection.” *ACM Transactions on Knowledge Discovery from Data*, **10**(1), 5. [doi:10.1145/2733381](https://doi.org/10.1145/2733381).

Celebi ME, Aslandogan YA, Bergstresser PR (2005). “Mining Biomedical Images with Density-Based Clustering.” In *International Conference on Information Technology: Coding and Computing (ITCC’05)*, volume II, pp. 163–168.

Chang H, Yeung DY (2008). “Robust Path-Based Spectral Clustering.” *Pattern Recognition*,

**41**(1), 191–203. [doi:10.1016/j.patcog.2007.04.010](https://doi.org/10.1016/j.patcog.2007.04.010).

Chen W, Ji MH, Wang JM (2014). “T-DBSCAN: A Spatiotemporal Density Clustering for GPS Trajectory Segmentation.” *International Journal of Online Engineering*, **10**(6), 19–24. [doi:10.3991/ijoe.v10i6.3881](https://doi.org/10.3991/ijoe.v10i6.3881).

Chowdhury AKMR, Mollah ME, Rahman MA (2010). “An Efficient Method for Subjec- tively Choosing Parameter ‘*k*’ automatically in VDBSCAN (Varied Density Based Spa- tial Clustering of Applications with Noise) Algorithm.” In *The 2nd International Con- ference on Computer and Automation Engineering (ICCAE)*, volume 1, pp. 38–41. [doi:](https://doi.org/10.1109/ICCAE.2010.5452004) [10.1109/ICCAE.2010.5452004](https://doi.org/10.1109/ICCAE.2010.5452004).

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001). *Introduction to Algorithms*. 2nd edition. The MIT Press.

Duan L, Xu L, Guo F, Lee J, Yan B (2007). “A Local-Density Based Spatial Clustering Algorithm with Noise.” *Information Systems*, **32**(7), 978–986. [doi:10.1016/j.is.2006.](https://doi.org/10.1016/j.is.2006.10.006) [10.006](https://doi.org/10.1016/j.is.2006.10.006).

Eddelbuettel D, François R (2011). “Rcpp: Seamless R and C++ Integration.” *Journal of Statistical Software*, **40**(8), 1–18. [doi:10.18637/jss.v040.i08](https://doi.org/10.18637/jss.v040.i08).

Ertöz L, Steinbach M, Kumar V (2003). “Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data.” In *Proceedings of the 3rd SIAM International Conference on Data Mining*, pp. 47–58.

Ester M, Kriegel HP, Sander J, Xu X, *et al.* (1996). “A Density-Based Algorithm for Dis- covering Clusters in Large Spatial Databases with Noise.” In *KDD’96 Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, volume 96, pp. 226–231.

Fournier-Viger P, Gomariz A, Gueniche T, Soltani A, Wu CW, Tseng VS, *et al.* (2014). “SPMF: A Java Open-Source Pattern Mining Library.” *Journal of Machine Learning Re- search*, **15**(1), 3389–3393.

Fränti P, Virmajoki O (2006). “Iterative Shrinking Method for Clustering Problems.” *Pattern Recognition*, **39**(5), 761–765. [doi:10.1016/j.patcog.2005.09.012](https://doi.org/10.1016/j.patcog.2005.09.012).

Fu L, Medico E (2007). “FLAME, a Novel Fuzzy Clustering Method for the Analysis of DNA Microarray Data.” *BMC Bioinformatics*, **8**(1), 1. [doi:10.1186/1471-2105-8-3](https://doi.org/10.1186/1471-2105-8-3).

Ghanbarpour A, Minaei B (2014). “EXDBSCAN: An Extension of DBSCAN to Detect Clusters in Multi-Density Datasets.” In *2014 Iranian Conference on Intelligent Systems (ICIS)*, pp. 1–5.

Gionis A, Mannila H, Tsaparas P (2007). “Clustering Aggregation.” *ACM Transactions on Knowledge Discovery from Data*, **1**(1), 4. [doi:10.1145/1217299.1217303](https://doi.org/10.1145/1217299.1217303).

Gosling J, Joy B, Steele G, Bracha G (2000). *The Java Language Specification*. Addison- Wesley Professional.

Gupta G, Liu A, Ghosh J (2010). “Automated Hierarchical Density Shaving: A Ro- bust Automated Clustering and Visualization Framework for Large Biological Data Sets.” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **7**(2), 223–237. [doi:10.1109/tcbb.2008.32](https://doi.org/10.1109/tcbb.2008.32).

Hahsler M, Hornik K, Buchta C (2008). “Getting Things in Order: An Introduction to the R Package seriation.” *Journal of Statistical Software*, **25**(3), 1–34. [doi:10.18637/jss.v025.](https://doi.org/10.18637/jss.v025.i03) [i03](https://doi.org/10.18637/jss.v025.i03).

Hahsler M, Piekenbrock M (2019). *dbscan: Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms*. R package version 1.1-5, URL [https://CRAN.](https://CRAN.R-project.org/package%3Ddbscan) [R-project.org/package=dbscan](https://CRAN.R-project.org/package%3Ddbscan).

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009). “The WEKA Data Mining Software: An Update.” *ACM SIGKDD Explorations Newsletter*, **11**(1), 10–18. [doi:10.1145/1656274.1656278](https://doi.org/10.1145/1656274.1656278).

Hennig C (2019). *fpc: Flexible Procedures for Clustering*. R package version 2.2-3, URL

[https://CRAN.R-project.org/package=fpc](https://CRAN.R-project.org/package%3Dfpc).

Hubert L, Arabie P (1985). “Comparing Partitions.” *Journal of Classification*, **2**(1), 193–218.

[doi:10.1007/bf01908075](https://doi.org/10.1007/bf01908075).

Jain AK, Martin HC (2005). “Law, Data Clustering: A User’s Dilemma.” In *Proceedings of the First International Conference on Pattern Recognition and Machine Intelligence*.

Jain AK, Murty MN, Flynn PJ (1999). “Data Clustering: A Review.” *ACM Computuing Surveys*, **31**(3), 264–323. [doi:10.1145/331499.331504](https://doi.org/10.1145/331499.331504).

Jarvis RA, Patrick EA (1973). “Clustering Using a Similarity Measure Based on Shared near Neighbors.” *IEEE Transactions on Computers*, **100**(11), 1025–1034. [doi:10.1109/t-c.](https://doi.org/10.1109/t-c.1973.223640) [1973.223640](https://doi.org/10.1109/t-c.1973.223640).

Kailing K, Kriegel HP, Kröger P (2004). “Density-Connected Subspace Clustering for High- Dimensional Data.” In *Proceedings of the 2004 SIAM International Conference on Data Mining*, volume 4.

Karypis G, Han EH, Kumar V (1999). “Chameleon: Hierarchical Clustering Using Dynamic Modeling.” *Computer*, **32**(8), 68–75. [doi:10.1109/2.781637](https://doi.org/10.1109/2.781637).

Kaufman L, Rousseeuw PJ (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, New York. [doi:10.1002/9780470316801](https://doi.org/10.1002/9780470316801).

Kisilevich S, Mansmann F, Keim D (2010). “P-DBSCAN: A Density Based Clustering Algo- rithm for Exploration and Analysis of Attractive Areas Using Collections of Geo-Tagged Photos.” In *Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application*, p. 38.

Kriegel HP, Schubert E, Zimek A (2016). “The (Black) Art of Runtime Evaluation: Are We Comparing Algorithms or Implementations?” *Knowledge and Information Systems*, pp. 1–38. [doi:10.1007/s10115-016-1004-2](https://doi.org/10.1007/s10115-016-1004-2).

Liu P, Zhou D, Wu N (2007). “VDBSCAN: Varied Density Based Spatial Clustering of Applications with Noise.” In *2007 International Conference on Service Systems and Service Management*, pp. 1–4.

MacQueen J, *et al.* (1967). “Some Methods for Classification and Analysis of Multivariate Observations.” In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pp. 281–297.

Microsoft Academic Search (2017). “Top Publications in Data Mining.” [https:](https://academic.microsoft.com/%23/search?iq=%40data%20mining%40&q=data%20mining&filters&from=56&sort=3)

[//academic.microsoft.com/#/search?iq=@data%20mining@&q=data%20mining&](https://academic.microsoft.com/%23/search?iq=%40data%20mining%40&q=data%20mining&filters&from=56&sort=3) [filters=&from=56&sort=3](https://academic.microsoft.com/%23/search?iq=%40data%20mining%40&q=data%20mining&filters&from=56&sort=3). Accessed on 2017-10-02.

Mount DM, Arya S (2010). *ANN: Library for Approximate Nearest Neighbour Searching*.

URL <http://www.cs.umd.edu/~mount/ANN/>.

Novikov A (2019). “PyClustering: Data Mining Library.” *Journal of Open Source Software*,

**4**(36), 1230. [doi:10.21105/joss.01230](https://doi.org/10.21105/joss.01230).

Patwary MMA, Blair J, Manne F (2010). “Experiments on Union-Find Algorithms for the Disjoint-Set Data Structure.” In *International Symposium on Experimental Algorithms*, pp. 411–423.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, *et al.* (2011). “SciKit-Learn: Machine Learn- ing in Python.” *Journal of Machine Learning Research*, **12**(Oct), 2825–2830. [doi:](https://doi.org/10.3389/fninf.2014.00014) [10.3389/fninf.2014.00014](https://doi.org/10.3389/fninf.2014.00014).

Pei T, Jasra A, Hand DJ, Zhu AX, Zhou C (2009). “DECODE: A New Method for Discovering Clusters of Different Densities in Spatial Data.” *Data Mining and Knowledge Discovery*, **18**(3), 337–369. [doi:10.1007/s10618-008-0120-3](https://doi.org/10.1007/s10618-008-0120-3).

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Founda- tion for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.

Rehman SU, Asghar S, Fong S, Sarasvady S (2014). “DBSCAN: Past, Present and Future.” In *2014 Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, pp. 232–238.

Roy S, Bhattacharyya DK (2005). “An Approach to Find Embedded Clusters Using Density Based Techniques.” In *International Conference on Distributed Computing and Internet Technology*, pp. 523–535.

Sander J (2011). “Density-Based Clustering.” In *Encyclopedia of Machine Learning*, pp.

270–273. Springer-Verlag.

Sander J, Qin X, Lu Z, Niu N, Kovarsky A (2003). “Automatic Extraction of Clusters from Hierarchical Clustering Representations.” In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 75–87.

Schubert E, Koos A, Emrich T, Züfle A, Schmid KA, Zimek A (2015). “A Framework for Clustering Uncertain Data.” *Proceedings of the VLDB Endowment – Proceedings of the 41st International Conference on Very Large Data Bases, Kohala Coast, Hawaii*, **8**(12), 1976–1979. [doi:10.14778/2824032.2824115](https://doi.org/10.14778/2824032.2824115).

Scrucca L, Fop M, Murphy TB, Raftery AE (2016). “mclust 5: Clustering, Classification and Density Estimation Using Gaussian Finite Mixture Models.” *The R Journal*, **8**(1), 205–233. [doi:10.32614/RJ-2016-021](https://doi.org/10.32614/RJ-2016-021).

SIGKDD (2014). “SIGKDD News: 2014 SIGKDD Test of Time Award.” [http://www.kdd.](http://www.kdd.org/News/view/2014-sigkdd-test-of-time-award) [org/News/view/2014-sigkdd-test-of-time-award](http://www.kdd.org/News/view/2014-sigkdd-test-of-time-award). Accessed on 2016-10-10.

Van Rossum G, *et al.* (2011). *Python Programming Language*. URL [https://www.python.](https://www.python.org/) [org/](https://www.python.org/).

Veenman CJ, Reinders MJT, Backer E (2002). “A Maximum Variance Cluster Algorithm.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(9), 1273–1280. [doi:](https://doi.org/10.1109/tpami.2002.1033218) [10.1109/tpami.2002.1033218](https://doi.org/10.1109/tpami.2002.1033218).

Verma M, Srivastava M, Chack N, Diswar AK, Gupta N (2012). “A Comparative Study of Various Clustering Algorithms in Data Mining.” *International Journal of Engineering Research and Applications*, **2**(3), 1379–1384. [doi:10.1109/icdse.2016.7823946](https://doi.org/10.1109/icdse.2016.7823946).

Zahn CT (1971). “Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters.”

*IEEE Transactions on Computers*, **100**(1), 68–86. [doi:10.1109/t-c.1971.223083](https://doi.org/10.1109/t-c.1971.223083).

# A. Technical note on OPTICS cluster extraction

Of the two cluster extraction methods outlined in this paper, the flat DBSCAN-type ex- traction method seems to remain the de facto clustering method implemented in most sta- tistical software for OPTICS. However, this method produces clusters which are quite sim- ilar to the DBSCAN. To the best of the authors’ knowledge, the only (other) library that has implemented the Extract-*ξ* method for finding *ξ*-clusters is the Environment for De- veloping KDD-Applications Supported by Index Structures (ELKI, [Schubert *et al.*](#_bookmark81)[2015](#_bookmark81)). Perhaps much of the complication as to why nearly every statistical computing framework has neglected the Extract-*ξ* cluster method stems from the fact that the original specifi- cation (Figure 19 in [Ankerst *et al.*](#_bookmark29)[1999](#_bookmark29)), while mostly complete, lacks important correc- tions that otherwise produce artifacts when clustering data. In the original specification of the algorithm, the dents in the ordering structure OPTICS produces are scanned for sig- nificant changes in reachability (specified by the *ξ* threshold). Clusters are identified as

ranges of consecutive points separated by 1 *− ξ* density-reachability changes in the reacha-

bility plot. It is possible, however, after the recursive completion of the update algorithm (Figure 7 in [Ankerst *et al.*](#_bookmark29)[1999](#_bookmark29)) that the next point processed in the ordering is not actually within the reachability distance of other members of the cluster being currently processed. To account for the missing details described above, a number of supplemental processing steps were added in the ELKI framework, which are mentioned in ELKI’s release notes (see <https://elki-project.github.io/releases/release_notes_0.7>). These steps cor- rect artifacts through the addition of a small filtering step, thus improving the *ξ*-cluster method from the original implementation mentioned in the original OPTICS paper. This correction was not introduced until 2015, 16 years after the original publication of OPTICS and the Extract-*ξ* method. dbscan has incorporated these important changes in extractXi() via the option correctPredecessors which is by default enabled.

## Affiliation:

Michael Hahsler

Department of Engineering Management, Information, and Systems Bobby B. Lyle School of Engineering, SMU

P. O. Box 750123, Dallas, TX 75275, United States of America E-mail: [mhahsler@lyle.smu.edu](mailto:mhahsler@lyle.smu.edu)

URL: <http://michael.hahsler.net/>

Matt Piekenbrock, Derek Doran

Department of Computer Science and Engineering Wright State University

3640 Colonel Glenn Hwy, Dayton, OH, 45435, United States of America E-mail: [piekenbrock.5@wright.edu](mailto:piekenbrock.5@wright.edu), [derek.doran@wright.edu](mailto:derek.doran@wright.edu)

URL: <http://www.wright.edu/~piekenbrock.5>, <http://knoesis.org/doran>

*Journal of Statistical Software* <http://www.jstatsoft.org/>

published by the Foundation for Open Access Statistics <http://www.foastat.org/>

October 2019, Volume 91, Issue 1 *Submitted:* 2017-02-06

[doi:10.18637/jss.v091.i01](https://doi.org/10.18637/jss.v091.i01) *Accepted:* 2018-06-07