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 algorithm DBSCAN and the augmented ordering algorithm OPTICS. Compared to other implementations, **dbscan** offers open-source implementations using C++ and advanced data structures like k-d trees to speed up computation. An important advantage of this implementation is that it is up-to-date with several primary advancements that have been added since their original publications, including artifact corrections and dendrogram extraction methods for OPTICS. Experiments with **dbscan**'s implementation of DBSCAN and OPTICS compared and other libraries such as FPC, ELKI, WEKA, PyClustering, SciKit-Learn and SPMF suggest that **dbscan** provides a very efficient implementation.

Keywords: DBSCAN, OPTICS, Density-based Clustering, Hierarchical Clustering.

1. Introduction

Clustering is typically described as the process of finding structure in data by grouping similar objects together, where the resulting set of groups are called clusters. Many clustering algorithms directly apply the idea that clusters can be formed such that objects in the same cluster should be more similar to each other than to objects in other clusters. The notion of similarity (or distance) stems from the fact that objects are assumed to be data points embedded in a data space in which a similarity measure can be defined. Examples are methods based on solving the k-means problem or mixture models which find the parameters of a parametric generative probabilistic model from which the observed data are assumed to arise. Another approach is hierarchical clustering, which uses local heuristics to form a hierarchy of nested grouping of objects. Most of these approaches (with the notable exception of single-link hierarchical clustering) are biased towards clusters with convex, hyper-spherical shape. A detailed review of these clustering algorithms is provided in Kaufman and Rousseeuw (1990), Jain, Murty, and Flynn (1999), and the more recent review by Aggarwal and Reddy (2013).

Density-based clustering approaches clustering differently. It simply posits that clusters are contiguous 'dense' regions in the data space (i.e., regions of high point density), separated by areas of low point density (Kriegel, Kröger, Sander, and Arthur 2011; Sander 2011). Density-based methods find such high-density regions representing clusters of arbitrary shape and

typically have a structured means of identifying noise points in low-density regions. These properties provide advantages for many applications compared to other clustering approaches. For example, geospatial data may be fraught with noisy data points due to estimation errors in GPS-enabled sensors (Chen, Ji, and Wang 2014) and may have unique cluster shapes caused by the physical space the data was captured in. Density-based clustering is also a promising approach to clustering high-dimensional data (Kailing, Kriegel, and Kröger 2004), where partitions are difficult to discover, and where the physical shape constraints assumed by model-based methods are more likely to be violated.

Several density-based clustering algorithms have been proposed, including DBSCAN algorithm (Ester, Kriegel, Sander, Xu *et al.* 1996), DENCLUE (Hinneburg and Keim 1998) and many DBSCAN derivates like HDBSCAN (Campello, Moulavi, Zimek, and Sander 2015). These clustering algorithms are widely used in practice with applications ranging from finding outliers in datasets for fraud prevention (Breunig, Kriegel, Ng, and Sander 2000), to finding patterns in streaming data (Chen and Tu 2007; Cao, Ester, Qian, and Zhou 2006), noisy signals (Kriegel and Pfeifle 2005; Ester *et al.* 1996; Tran, Wehrens, and Buydens 2006; Hinneburg and Keim 1998; Duan, Xu, Guo, Lee, and Yan 2007), gene expression data (Jiang, Pei, and Zhang 2003), multimedia databases (Kisilevich, Mansmann, and Keim 2010), and road traffic (Li, Han, Lee, and Gonzalez 2007).

This paper focuses on an efficient implementation of the DBSCAN algorithm (Ester *et al.* 1996), one of the most popular density-based clustering algorithms, whose consistent use earned it the SIGKDD 2014's Test of Time Award (SIGKDD 2014), and OPTICS (Ankerst, Breunig, Kriegel, and Sander 1999), often referred to as an extension of DBSCAN. While surveying software tools that implement various density-based clustering algorithms, it was discovered that in a large number of statistical tools, not only do implementations vary significantly in performance (Kriegel, Schubert, and Zimek 2016), but may also lack important components and corrections. Specifically, for the statistical computing environment R (Team *et al.* 2013), only naive DBSCAN implementations without speed-up with spatial data structures are available (e.g., in the well-known Flexible Procedures for Clustering package (Hennig 2015)), and OPTICS is not available. This motivated the development of a R package for density-based clustering with DBSCAN and related algorithms called **dbscan**. The **dbscan** package contains complete, correct and fast implementations of DBSCAN and OPTICS. The package currently enjoys thousands of new installations from the CRAN repository every month.

This article presents an overview of the R package **dbscan** focusing on DBSCAN and OPTICS, outlining its operation and experimentally compares its performance with implementations in other open-source implementations. We first review the concept of density-based clustering and present the DBSCAN and OPTICS algorithms in Section 2. This section concludes with a short review of existing software packages that implement these algorithms. Details about **dbscan**, with examples of its use, are presented in Section 3. A performance evaluation is presented in Section 4. Concluding remarks are offered in Section 5.

2. Density-based clustering

Density-based clustering is now a well-studied field. 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 2011). An important distinction between density-based clustering and alternative approaches to cluster analysis, such as the use of *(Gaussian) mixture models* (see Jain *et al.* 1999), is that the latter represents a *parametric* approach in which the observed data are assumed to have been produced by mixture of either Gaussian or other parametric families of distributions. While certainly useful in many applications, parametric approaches naturally assume clusters will exhibit some type convex (generally 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 of 'minimum variance', where the underlying assumption is made that ideal clusters are found by minimizing some measure of intra-cluster variance (often referred to as cluster cohesion) and maximizing the inter-cluster variance (cluster separation) (Arbelaitz, Gurrutxaga, Muguerza, Pérez, and Perona 2013). Conversely, the label density-based clustering is used for methods which do not assume parametric distributions, 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 k. This methodology is best expressed in the DBSCAN algorithm, which we discuss next.

2.1. DBSCAN: Density Based Spatial Clustering of Applications with Noise

As one of the most cited of the density-based clustering algorithms (Microsoft Academic Search 2016), DBSCAN (Ester *et al.* 1996) is likely the best known 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 will go through the most important definitions used in DBSCAN and related algorithms. The definitions and the presented pseudo code follows the original by Ester *et al.* (1996), but are adapted to provide a more consistent presentation with the other algorithms discussed in the paper.

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

Definition 1. ϵ-Neighborhood. *The ϵ-neighborhood, $N_\epsilon(p)$, of a data point p is the set of points within a specified radius ϵ around p.*

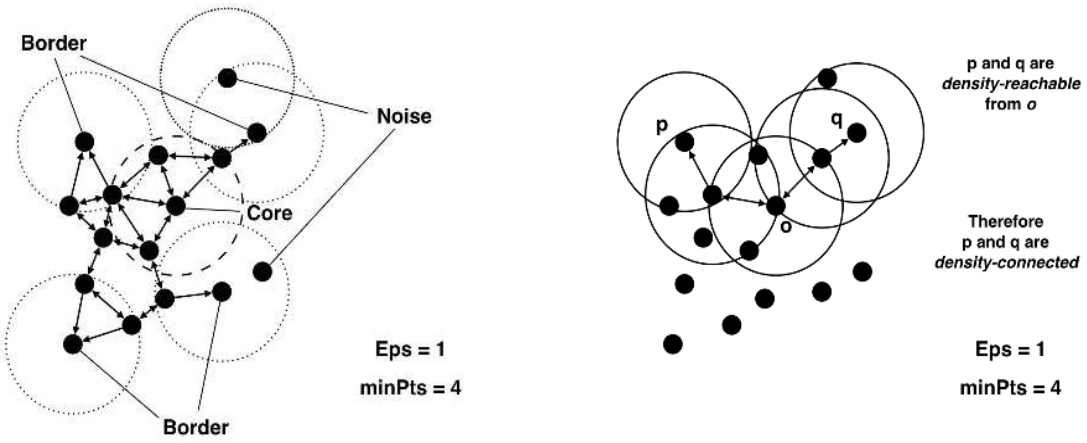
$$N_\epsilon(p) = \{q \mid d(p,q) < \epsilon\}$$

where d is some distance measure and $\epsilon \in \mathbb{R}^+$. Note that the point p is always in its own ϵ-neighborhood, i.e., $p \in N_\epsilon(p)$ always holds.

Following this definition, the size of the neighborhood $|N_\epsilon(p)|$ can be seen as a simple unnormalized kernel density estimate around p using a uniform kernel and a bandwidth of ϵ. DBSCAN uses $N_\epsilon(p)$ and a threshold called *minPts* to detect dense regions and to classify the points in a data set into **core**, **border**, or **noise** points.

Definition 2. Point classes. *A point $p \in D$ is classified as*

- *a* **core point** *if $N_\epsilon(p)$ has high density, i.e., $|N_\epsilon(p)| \geq minPts$ where $minPts \in \mathbb{Z}^+$ is a user-specified density threshold,*



(a) (b)

Figure 1: Concepts used 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.

- *a* **border point** *if p is not a core point, but it is in the neighborhood of a core point $q \in D$, i.e., $p \in N_\epsilon(q)$, or*
- *a* **noise point**, *otherwise.*

A visual example is shown in Figure 1(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 $q \in D$ is directly density-reachable from a point $p \in D$ with respect to ϵ and minPts if, and only if,*

1. $|N_\epsilon(p)| \geq minPts$, *and*
2. $q \in N_\epsilon(p)$.

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

Definition 4. Density-reachable. *A point p is density-reachable from q if there exists in D an ordered sequence of points $(p_1, p_2, ..., p_n)$ with $q = p_1$ and $p = p_n$ such that $p_i + 1$ directly density-reachable from p_i $\forall$ $i \in \{1, 2, ..., n-1\}$.*

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

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

Definition 6. Cluster. *A cluster C is a non-empty subset of D satisfying the following conditions:*

1. **Maximality**: *If $p \in C$ and q is density-reachable from p, then $q \in C$; and*
2. **Connectivity**: *$\forall\ p, q \in C$, p is density-connected to q.*

The DBSCAN algorithm identifies all such clusters by finding all core points and expanding each to all density-reachable points. The algorithm begins with an arbitrary point p and retrieves its ϵ-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 are considered noise.

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 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 *et al.* (2015) suggest a modification called DBSCAN* which considers all border points as noise instead and leaves them unassigned.

2.2. OPTICS: Ordering Points To Identify Clustering Structure

There are many instances where it would be useful to detect clusters of varying density. From identifying causes among similar seawater characteristics (Birant and Kut 2007), to network intrusion detection systems (Ertöz, Steinbach, and Kumar 2003), point of interest detection using geo-tagged photos (Kisilevich *et al.* 2010), classifying cancerous skin lesions (Celebi, Aslandogan, and Bergstresser 2005), the motivations for detecting clusters among varying densities are numerous. The inability to find clusters of varying density is a notable drawback of DBSCAN resulting 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.

In 1999, some of the original DBSCAN authors developed OPTICS (Ankerst *et al.* 1999) 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 ϵ and *minPts* parameters as DBSCAN, however, the ϵ 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 an additional concepts called core-distance and reachability-distance. All used distances are calculated using the same metric (often Euclidean distance) used for the neighborhood calculation.

Definition 7. Core-distance. *The core-distance of a point $p \in D$ with respect to minPts and ϵ is defined as*

$$\text{core-dist}(p; \epsilon, \mathit{minPts}) = \begin{cases} \mathit{UNDEFINED} & \mathit{if}\ |N_\epsilon(p)| < \mathit{minPts}, \mathit{and} \\ \text{minPts-dist}(p) & \mathit{otherwise.} \end{cases}$$

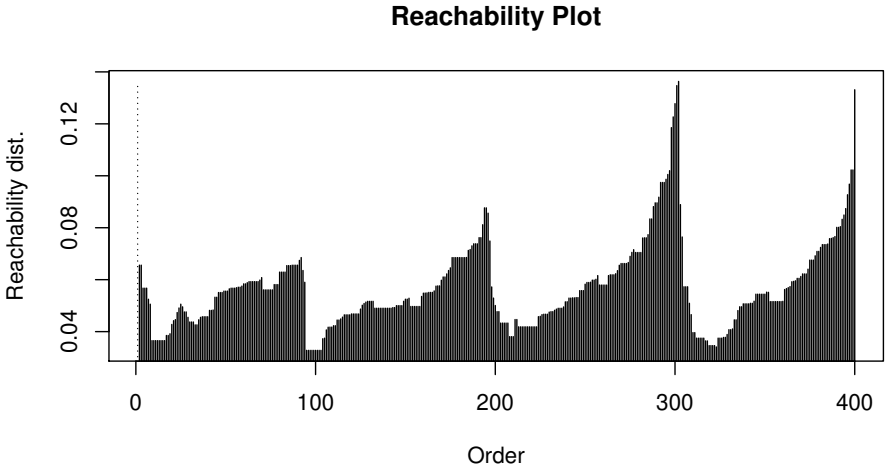


Figure 2: OPTICS reachability plot example for a data set with four clusters of 100 data points each.

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* $p \in D$ *to a point* $q \in D$ *parameterized by* ϵ *and* $minPts$ *is defined as*

$$\text{reachability-dist}(p, q; \epsilon, minPts) = \begin{cases} UNDEFINED & \text{if } |N_\epsilon(p)| < minPts, \text{ and} \\ \max(\text{core-dist}(p), d(p, q)) & \text{otherwise.} \end{cases}$$

The reachability-distance of a core point p with respect to object q is the smallest neighborhood radius such that p would be directly density-reachable from q. Note that ϵ is typically set very large compared to DBSCAN. Therefore, $minPts$ behaves differently for OPTICS: more points will be considered core points and it affects how many nearest neighbors are considered in the core-distance calculation, where larger values will lead to larger and more smooth reachability distributions. This needs to be kept in mind when choosing appropriate parameters.

OPTICS provides an augmented ordering. The algorithm starting with a point and expands it's neighborhood like DBSCAN, but it explores the new point 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 of the order and the resulting reachability-distance is shown in the form of a reachability plot in Figure 2. Low reachability-distances shown as valleys represent clusters separated by peaks representing points with larger distances. This density representation essentially conveys the same information as the often used dendrogram or 'tree-like' structure. This is why OPTICS is often also noted as a visualization tool. Sander, Qin, Lu, Niu, and Kovarsky (2003) showed how the output of OPTICS can be converted into an equivalent dendrogram, and that under certain conditions, the dendrogram produced by the well known hierarchical clustering with single linkage is identical to running OPTICS with the parameter $minPts = 2$

From the order discovered by OPTICS, two ways to group points into clusters was discussed in Ankerst *et al.* (1999), one which we will refer to as the **ExtractDBSCAN** method and

one which we will refer to as the **Extract-ξ** method summarized below:

1. **ExtractDBSCAN** uses a single global reachability-distance threshold ϵ' to extract a clustering. This can be seen as a horizontal line in the reachability plot in 2. Peaks above the cut-off represent noise points and separate the clusters.

2. **Extract-ξ** identifies clusters *hierarchically* by scanning through the ordering that OPTICS produces to identify significant, relative changes in reachability-distance. The authors of OPTICS noted that clusters can be thought of as identifying 'dents' in the reachability plot.

The ExtractDBSCAN method extracts a clustering equivalent to DBSCAN* (i.e., DBSCAN where border points stay unassigned). Because this method extracts clusters like DBSCAN, it cannot 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, which we call Extract-ξ[1], identifies a cluster hierarchy and replaces the data dependent global ϵ 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. The hierarchical representation Extract-ξ can, as opposed to the ExtractDBSCAN method, produce clusters of varying densities.

With its two ways of extracting clusters from the ordering, whether through either the global ϵ' or relative ξ threshold, OPTICS can be seen as a generalization of DBSCAN. In contexts where one wants to find clusters of similar density, OPTICS's ExtractDBSCAN yields a DBSCAN-like solution, while in other contexts Extract-ξ can generate a hierarchy representing 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 Sarasvady 2014), OPTICS has received decidedly less attention. Perhaps one of the reasons for this is because the Extract-ξ method for grouping points into clusters has gone largely unnoticed, as it is not implemented in most open-source software packages that advertise an implementation of OPTICS. This includes implementations in WEKA (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten 2009), SPMF (Fournier-Viger, Gomariz, Gueniche, Soltani, Wu, Tseng *et al.* 2014), and the PyClustering (Novikov 2016) and Scikit-learn (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg *et al.* 2011) libraries for Python. To the best of our knowledge, the only other open-source library sporting a complete implementation of OPTICS is ELKI (Schubert, Koos, Emrich, Züfle, Schmid, and Zimek 2015), written in Java.

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 2014; Chowdhury, Mollah, and Rahman 2010; Gupta, Liu, and Ghosh 2010; Duan *et al.* 2007). Along the same line of thought, others cite OPTICS as

[1] In the original OPTICS publication Ankerst *et al.* (1999), 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 as the Extract-ξ method.

Library	DBSCAN	OPTICS	ExtractDBSCAN	Extract-ξ
dbscan	✓	✓	✓	✓
ELKI	✓	✓	✓	✓
SPMF	✓	✓	✓	
PyClustering	✓	✓	✓	
WEKA	✓	✓	✓	
SCIKIT-LEARN	✓			
FPC	✓			

Library	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. A ✓ symbol denotes availability.

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 experimentation (Verma, Srivastava, Chack, Diswar, and Gupta 2012; Roy and Bhattacharyya 2005; Liu, Zhou, and Wu 2007; Pei, Jasra, Hand, Zhu, and Zhou 2009). However, OPTICS fundamentally returns an ordering of the data which can be post-processed to extract either 1) a flat clustering 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.

2.3. Current implementations of DBSCAN and OPTICS

Implementations of DBSCAN and/or OPTICS are available in many statistical software packages. We focus here on open-source solutions. These include the Waikato Environment for Knowledge Analysis (WEKA) (Hall *et al.* 2009), the Sequential Pattern Mining Framework (SPMF) (Fournier-Viger *et al.* 2014), the Environment for Developing KDD-Application supported by Index Structures (ELKI) (Schubert *et al.* 2015), the Python library scikit-learn (Pedregosa *et al.* 2011), the PyClustering Data Mining library (Novikov 2016), the Flexible Procedures for Clustering R package (Hennig 2015), and the **dbscan** package (Hahsler and Piekenbrock 2016) introduced in this paper.

Table 1 presents a comparison of the features offered by these packages. All packages support DBSCAN and most use index acceleration to speed up the ϵ-neighborhood queries involved in both DBSCAN and OPTICS algorithms, the known bottleneck that typically dominates the runtime and is essential for processing larger data sets. **dbscan** is the first R implementation offering this improvement. OPTICS with ExtractDBSCAN is also widely implemented, but the Extract-ξ method, as well as the use of dendrograms with OPTICS, is only available in

dbscan and ELKI. A small experimental runtime comparison is provided in Section 4.

3. 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, François, Allaire, Chambers, Bates, and Ushey (2011)) using the k-d tree data structure implemented in the C++ library ANN (Mount and Arya 2010) 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 data set in form of a `data.frame` or a `matrix`. The implementations 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 ϵ-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 just 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 just 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.* (2015) suggest for DBSCAN* to consider 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 below.

Clusters can be extracted from the linear order produced by OPTICS. The **dbscan** implementation of the cluster extraction methods for ExtractDBSCAN and Extract-ξ are:

```
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 ϵ 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.* (1999) by pruning the steep up area for points that have predecessors not in the cluster (see Technical Note in Appendix A for details).

3.1. Nearest Neighbor Search

The density based algorithms in **dbscan** rely heavily on forming neighborhoods, i.e., finding all points belonging to an ϵ-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 $O(n^2)$ runtime complexity. A convenient way in R 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 $O(n^2)$, and becomes very large and slow to compute for medium to large data sets.

In order to avoid computing the complete distance matrix, **dbscan** relies on a space-partitioning data structure called a k-d trees (Bentley 1975). 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 $O(n\,log(n))$. 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 of neighbors in `x` for each point in `x`. `sort` controls if the returned points are sorted by 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 data sets, 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 2010) 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. Only nearest neighbors up to a distance of a factor of (1 + approx)eps will be returned, but some actual neighbors may be omitted potentially leading to spurious clusters and noise points. However, the algorithm will enjoy a significant speedup. For more details, we refer the reader to the documentation of the ANN library (Mount and Arya 2010). `dbscan()` and `optics()` use internally `frNN()` and the additional arguments in `...` are passed on to the nearest neighbor search method.

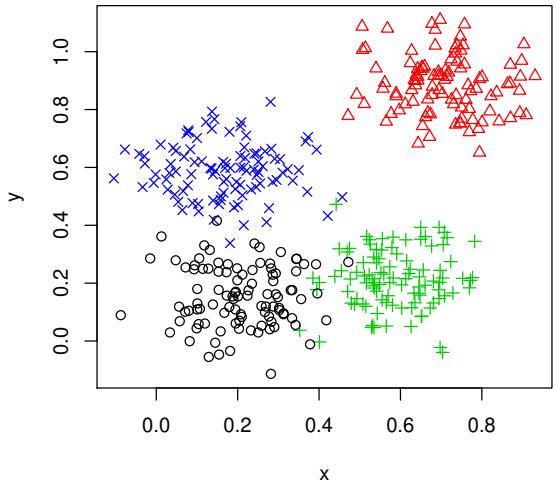


Figure 3: The sample dataset, consisting of 4 noisy Gaussian distributions with slight overlap.

3.2. Clustering with DBSCAN

We use a very simple artificial data set of four slightly overlapping Gaussians in two-dimensional space with 100 points each. We load **dbscan**, set the random number generator to make the results reproducible and create the data set.

```
> library("dbscan")
> set.seed(2)
> n <- 400
> x <- cbind(
+   x = runif(4, 0, 1) + rnorm(n, sd = 0.1),
+   y = runif(4, 0, 1) + rnorm(n, sd = 0.1)
+   )
> true_clusters <- rep(1:4, time = 100)

> plot(x, col = true_clusters, pch = true_clusters)
```

The resulting data set is shown in Figure 3.

To apply DBSCAN, we need to decide on the neighborhood radius `eps` and the density threshold `minPts`. The rule of thumb for minPts is to use at least the number of dimensions of the data set plus one. In our case, this is 3. For eps, we can plot the points' kNN distances (i.e., the distance to the kth 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 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.

```
> kNNdistplot(x, k = 3)
> abline(h=.05, col = "red", lty=2)
```

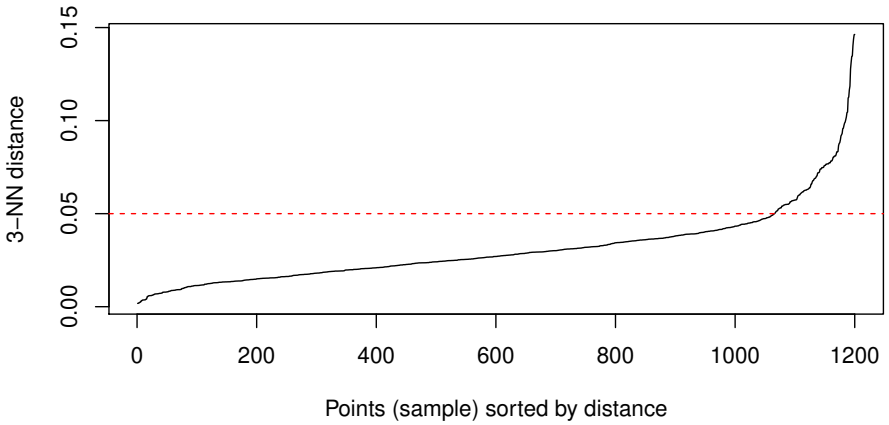


Figure 4: k-Nearest Neighbor Distance plot.

The kNN distance plot is shown in Figure 4. A knee is visible at around a 3-NN distance of 0.05. We have manually added a horizontal line for reference.

Now we can perform the clustering with the chosen parameters.

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

The resulting clustering identified one large cluster with 185 member points and 2 medium size clusters of between 87 and 89 points, three very small clusters and 30 noise points (represented 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.

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

The scatter plot in Figure 5 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 isolated points. **dbscan** also provides a plot that adds convex cluster hulls to the scatter plot shown in Figure 6.

```
> hullplot(x, res)
```

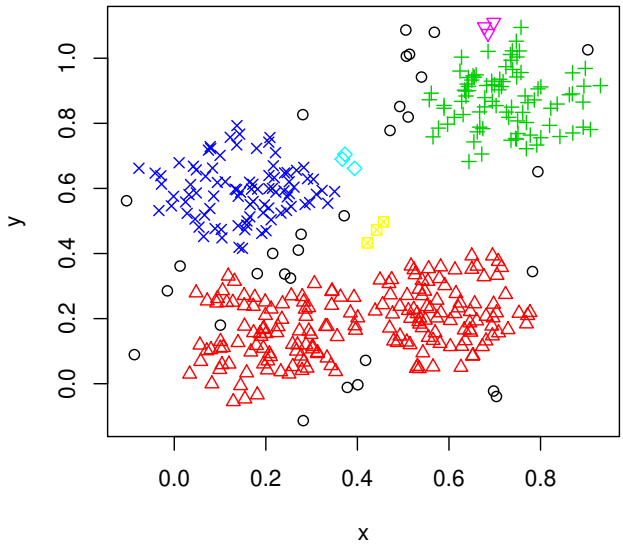


Figure 5: Result of clustering with DBSCAN. Noise is represented as black circles.

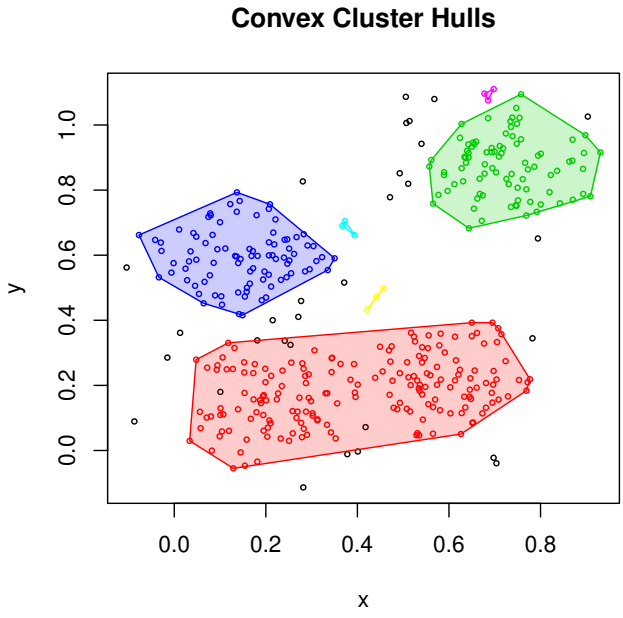


Figure 6: Convex hull plot of the DBSCAN clustering. Noise points are black. Note that noise points and points of another cluster may lie within the convex hull of a different cluster.

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 close enough to a core point.

```
> 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
```

3.3. 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 10 here) and `minPts` mostly affects core and reachability-distance calculation, where larger values have a smoothing effect. We use also 10, i.e., the core-distance is defined as the distance to the 9th nearest neighbor (spanning a neighborhood of 10 points).

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

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.

```
> 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 data set 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.

```
> plot(res)
```

The reachability plot in Figure 7 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 data sets we can plot a line connecting the points in order.

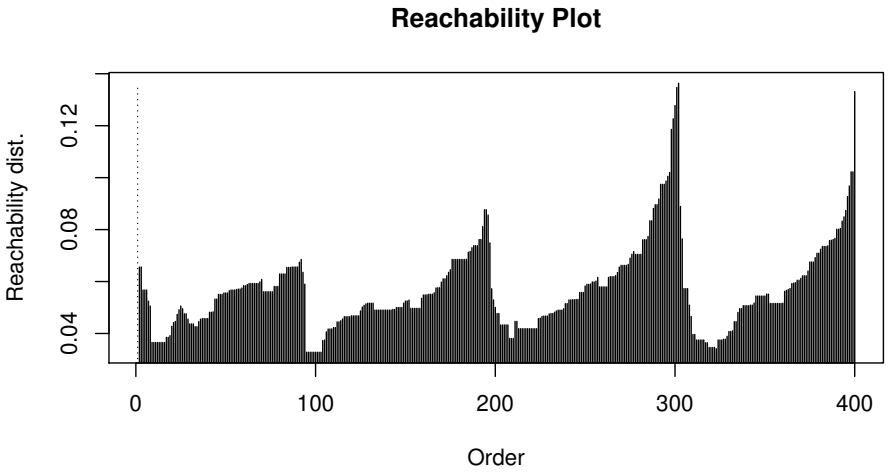


Figure 7: OPTICS reachability plot. Note that the first reachability value is always UNDEFINED.

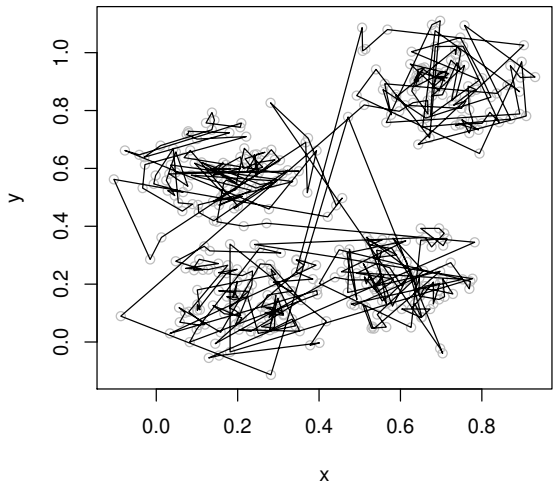


Figure 8: OPTICS order of data points represented as a line.

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

Figure 8 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, 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 eps parameter. The reachability plot in figure 7 shows four peaks, i.e., points with a high reachability-distance. These points indicate boundaries between clusters four clusters. An `eps` threshold that separates the four clusters can be visually determined. In this case we use `eps_cl` of 0.065.

```
> res <- extractDBSCAN(res, eps_cl = .065)
> plot(res)

> hullplot(x, res)
```

The resulting reachability and corresponding clusters are shown in Figures 9 and 10. The clustering resembles closely the original structure of the four clusters with which the data were generated, 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.

```
> res <- extractXi(res, xi = 0.05)
> 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`.

```
> 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
```

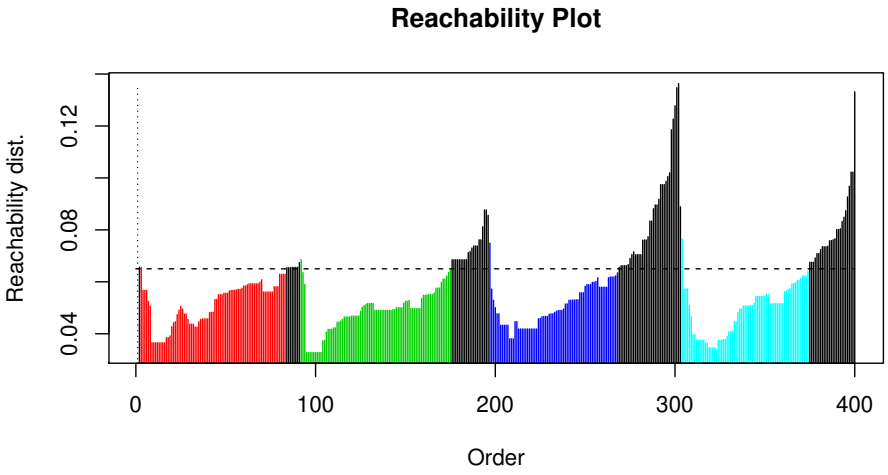


Figure 9: Reachability plot for a DBSCAN-type clustering extracted at global $\epsilon = 0.065$ results in four clusters.

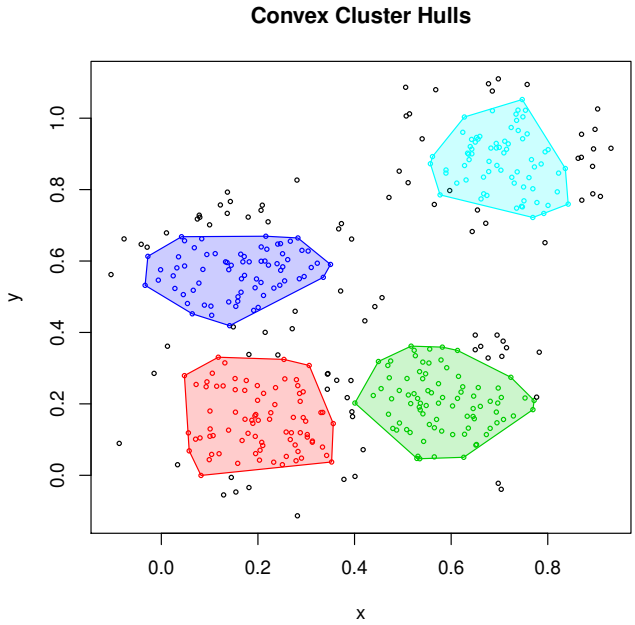


Figure 10: Convex hull plot for a DBSCAN-type clustering extracted at global $\epsilon = 0.065$ results in four clusters.

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

```
> plot(res)
```

```
> hullplot(x, res)
```

Figure 11 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 shown in Figure 12. 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.

3.4. Reachability and Dendrograms

Reachability plots can be converted into equivalent dendrograms (Sander *et al.* 2003). **dbscan** contains a fast implementation of the reachability-to-dendrogram conversion algorithm through the use of a disjoint-set data structure (Cormen, Leiserson, Rivest, and Stein 2001; Patwary, Blair, and Manne 2010), allowing the user to choose which hierarchical representation they prefer. The conversion algorithm can be directly called for OPTICS objects using the coercion method `as.dendrogram()`.

```
> dend <- as.dendrogram(res)
> dend
```

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

The dendrogram can be plotted using the standard plot method.

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

Note how the dendrogram in Figure 13 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.

4. Performance Comparison

Finally, we evaluate the performance of **dbscan**'s implementation of DBSCAN and OPTICS against other open-source implementations. This is not a comprehensive evaluation study, but is used to demonstrate the performance of **dbscan**'s DBSCAN and OPTICS implementation on datasets of varying sizes as compared to other software packages. A comparative test was performed using both DBSCAN and OPTICS algorithms, where supported, for the libraries listed in Table 1 on page 8. The used datasets and their sizes are listed in Table 2. The data sets tested include s1 and s2, the randomly generated but moderately-separated Gaussian clusters often used for agglomerative cluster analysis (Fränti and Virmajoki 2006), the R15 validation data set used for maximum variance based clustering approach by Veenman, Reinders, and Backer (2002), the well-known spatial data set t4.8k used for validation of the

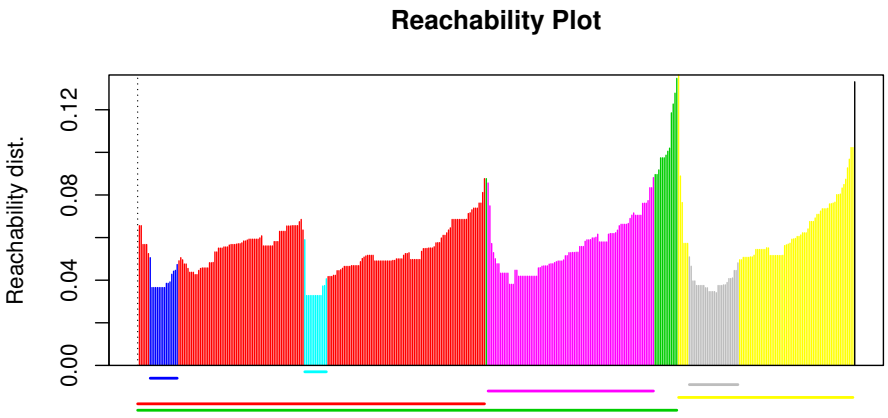


Figure 11: Reachability plot of a hierarchical clustering extracted with Extract-ξ.

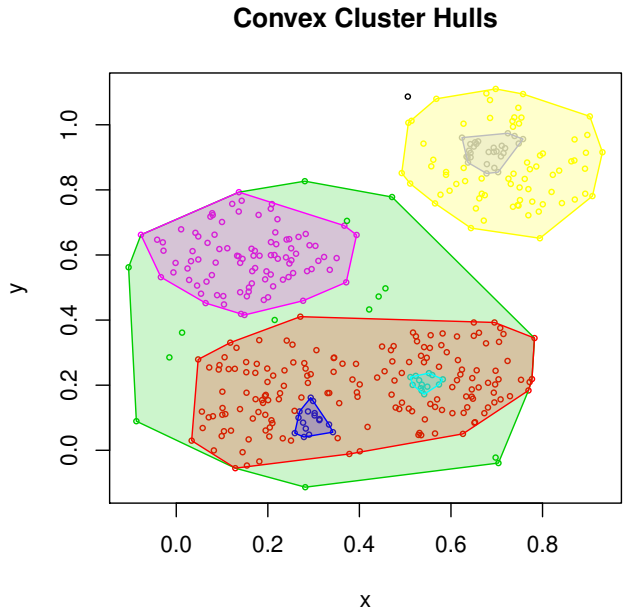


Figure 12: Convex hull plot of a hierarchical clustering extracted with Extract-ξ.

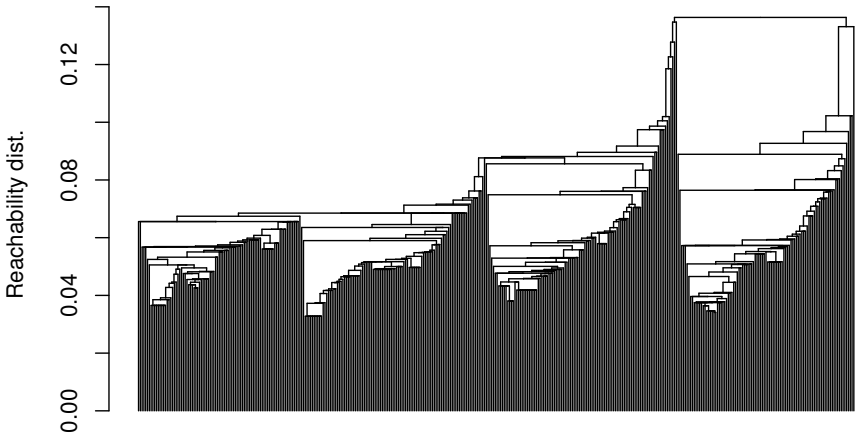


Figure 13: Dendrogram structure of OPTICS reordering.

CHAMELEON algorithm (Karypis, Han, and Kumar 1999), along with a variety of shape data sets commonly found in clustering validation papers (Gionis, Mannila, and Tsaparas 2007; Zahn 1971; Chang and Yeung 2008; Jain and Martin 2005; Fu and Medico 2007).

We performed the 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 an 2.5 GHz Intel Core i7 processor, running OS X El Capitan 10.11.6. All data sets where normalized to the unit interval, [0, 1], per dimension to standardize neighbor queries. For all data sets we used $minPts = 2$ and $\epsilon = 0.10$ for DBSCAN. For OPTICS, $minPts = 2$ with a large $\epsilon = 1$ was used. We replicated each run for each data set 15 times and report the average runtime here. Figures 14 and 15 shows the runtimes. The datasets are sorted from easiest to hardest and the algorithm in the legend are sorted from on average fastest to slowest. The results show that the implementation in *dbscan* compares very favorably to the other implementations.

5. Concluding Remarks

The **dbscan** package offers a set of scalable, robust, and complete implementations 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. In addition to DBSCAN and OPTICS discussed in this paper, **dbscan** also contains a fast version of the local outlier factor (LOF) algorithm (Breunig *et al.* 2000) and an implementation of HDBSCAN (Campello *et al.* 2015) is under development.

Data set	Size	Dimension
Aggregation	788	2
Compound	399	2
D31	3,100	2
flame	240	2
jain	373	2
pathbased	300	2
R15	600	2
s1	5,000	2
s4	5,000	2
spiral	312	2
t4.8k	8,000	2
synth1	1000	3
synth2	1000	10
synth3	1000	100

Table 2: Datasets used for comparison.

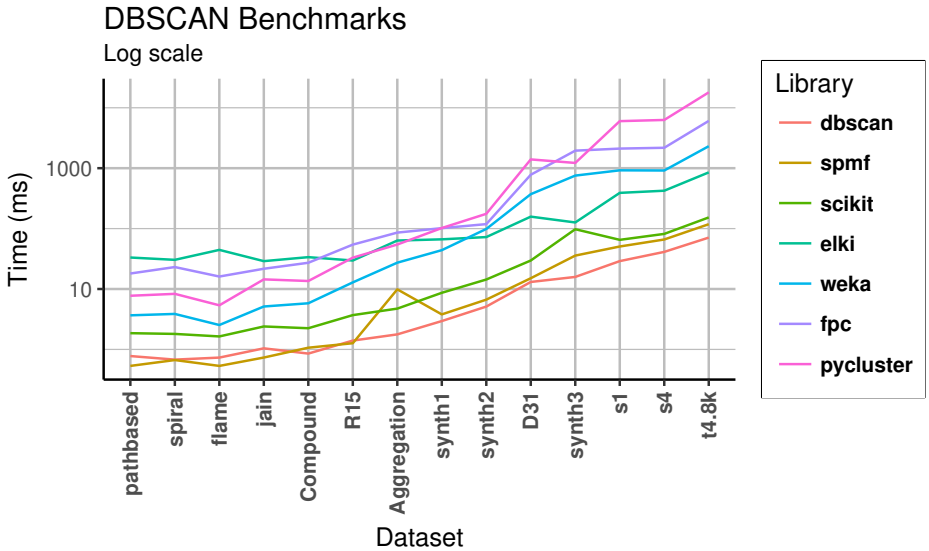


Figure 14: Runtime of DBSCAN in milliseconds (y-axis, logarithmic scale) vs. the name of the data set tested (x-axis).

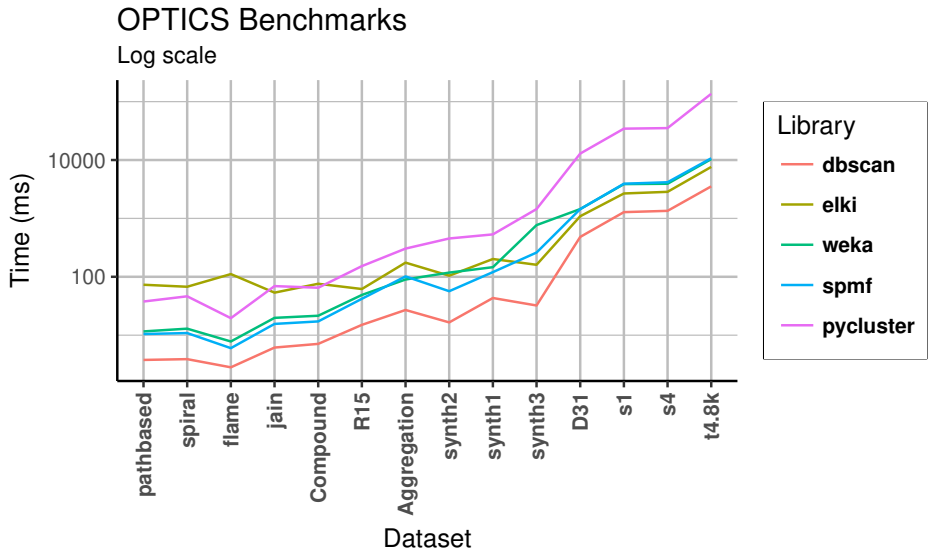


Figure 15: Runtime of OPTICS in milliseconds (y-axis, logarithmic scale) vs. the name of the data set tested (x-axis).

6. 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. ISBN 1466558210, 9781466558212.

Ankerst M, Breunig MM, Kriegel HP, Sander J (1999). "OPTICS: ordering points to identify the clustering structure." In *ACM Sigmod Record*, volume 28, pp. 49–60. ACM.

Arbelaitz O, Gurrutxaga I, Muguerza J, Pérez JM, Perona I (2013). "An extensive comparative study of cluster validity indices." *Pattern Recognition*, **46**(1), 243–256.

Bentley JL (1975). "Multidimensional binary search trees used for associative searching." *Communications of the ACM*, **18**(9), 509–517.

Birant D, Kut A (2007). "ST-DBSCAN: An algorithm for clustering spatial–temporal data." *Data & Knowledge Engineering*, **60**(1), 208–221.

Breunig MM, Kriegel HP, Ng RT, Sander J (2000). "LOF: identifying density-based local outliers." In *ACM sigmod record*, volume 29, pp. 93–104. ACM.

Campello RJ, Moulavi D, Zimek A, Sander J (2015). "Hierarchical density estimates for data clustering, visualization, and outlier detection." *ACM Transactions on Knowledge Discovery from Data (TKDD)*, **10**(1), 5.

Cao F, Ester M, Qian W, Zhou A (2006). "Density-Based Clustering over an Evolving Data Stream with Noise." In *SDM*, volume 6, pp. 328–339. SIAM.

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*, volume 1, pp. 163–168. IEEE.

Chang H, Yeung DY (2008). "Robust path-based spectral clustering." *Pattern Recognition*, **41**(1), 191–203.

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. ISSN 18612121. doi:10.3991/ijoe.v10i6.3881.

Chen Y, Tu L (2007). "Density-based clustering for real-time stream data." In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133–142. ACM.

Chowdhury AMR, Mollah ME, Rahman MA (2010). "An efficient method for subjectively choosing parameter 'k'automatically in VDBSCAN (Varied Density Based Spatial Clustering of Applications with Noise) algorithm." In *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, volume 1, pp. 38–41. IEEE.

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001). "Introduction to algorithms second edition."

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.

Eddelbuettel D, François R, Allaire J, Chambers J, Bates D, Ushey K (2011). "Rcpp: Seamless R and C++ integration." *Journal of Statistical Software*, **40**(8), 1–18.

Ertöz L, Steinbach M, Kumar V (2003). "Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data." In *SDM*, pp. 47–58. SIAM.

Ester M, Kriegel HP, Sander J, Xu X, *et al.* (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." In *Kdd*, 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 Research*, **15**(1), 3389–3393.

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

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

Ghanbarpour A, Minaei B (2014). "EXDBSCAN: An extension of DBSCAN to detect clusters in multi-density datasets." In *Intelligent Systems (ICIS), 2014 Iranian Conference on*, pp. 1–5. IEEE.

Gionis A, Mannila H, Tsaparas P (2007). "Clustering aggregation." *ACM Transactions on Knowledge Discovery from Data (TKDD)*, **1**(1), 4.

Gupta G, Liu A, Ghosh J (2010). "Automated hierarchical density shaving: A robust automated clustering and visualization framework for large biological data sets." *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **7**(2), 223–237. ISSN 15455963. doi:10.1109/TCBB.2008.32.

Hahsler M, Piekenbrock M (2016). *dbscan: Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms*. R package version 0.9-8.2.

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.

Hennig C (2015). *fpc: Flexible Procedures for Clustering*. R package version 2.1-10, URL https://CRAN.R-project.org/package=fpc.

Hinneburg A, Keim DA (1998). "An efficient approach to clustering in large multimedia databases with noise." In *KDD*, volume 98, pp. 58–65.

Jain AK, Martin H (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. ISSN 0360-0300. doi:10.1145/331499.331504. URL http://doi.acm.org/10.1145/331499.331504.

Jiang D, Pei J, Zhang A (2003). "DHC: a density-based hierarchical clustering method for time series gene expression data." In *Bioinformatics and Bioengineering, 2003. Proceedings. Third IEEE Symposium on*, pp. 393–400. IEEE.

Kailing K, Kriegel HP, Kröger P (2004). "Density-connected subspace clustering for high-dimensional data." In *Proc. SDM*, volume 4. SIAM.

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

Kaufman L, Rousseeuw PJ (1990). *Finding groups in data : an introduction to cluster analysis*. Wiley series in probability and mathematical statistics. Wiley, New York. ISBN 0-471-87876-6.

Kisilevich S, Mansmann F, Keim D (2010). "P-DBSCAN: a density based clustering algorithm 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. ACM.

Kriegel HP, Kröger P, Sander J, Arthur Z (2011). "Density-based clustering." *Wires Data and Knowledge Discovery*, **1**, 231–240.

Kriegel HP, Pfeifle M (2005). "Density-based clustering of uncertain data." In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 672–677. ACM.

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.

Li X, Han J, Lee JG, Gonzalez H (2007). "Traffic density-based discovery of hot routes in road networks." In *International Symposium on Spatial and Temporal Databases*, pp. 441–459. Springer.

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. IEEE.

Microsoft Academic Search (2016). "Top publications in data mining." http://academic.research.microsoft.com/RankList?entitytype=1&topDomainID=2&subDomainID=7&last=0&start=1&end=100. (Accessed on 08/29/2016).

Mount DM, Arya S (2010). *ANN: library for approximate nearest neighbour searching*. URL http://www.cs.umd.edu/~mount/ANN/.

Novikov A (2016). "PyClustering: PyClustering library." http://pythonhosted.org/pyclustering/. V.0.6.6.

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. Springer.

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 learning in Python." *Journal of Machine Learning Research*, **12**(Oct), 2825–2830.

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.

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

Roy S, Bhattacharyya D (2005). "An approach to find embedded clusters using density based techniques." In *International Conference on Distributed Computing and Internet Technology*, pp. 523–535. Springer.

Sander J (2011). "Density-based clustering." In *Encyclopedia of Machine Learning*, pp. 270–273. Springer.

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. Springer.

Schubert E, Koos A, Emrich T, Züfle A, Schmid KA, Zimek A (2015). "A Framework for Clustering Uncertain Data." *PVLDB*, **8**(12), 1976–1979. URL http://www.vldb.org/pvldb/vol8/p1976-schubert.pdf.

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

Team RC, *et al.* (2013). "R: A language and environment for statistical computing."

Tran TN, Wehrens R, Buydens LM (2006). "KNN-kernel density-based clustering for high-dimensional multivariate data." *Computational Statistics & Data Analysis*, **51**(2), 513–525.

Veenman CJ, Reinders MJT, Backer E (2002). "A maximum variance cluster algorithm." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(9), 1273–1280.

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 (IJERA)*, **2**(3), 1379–1384.

Zahn CT (1971). "Graph-theoretical methods for detecting and describing gestalt clusters." *IEEE Transactions on computers*, **100**(1), 68–86.

A. Technical Note on OPTICS cluster extraction

Of the two cluster extraction methods outlined in the publication, the flat DBSCAN-type extraction method seems to remain the defacto clustering method implemented across most statistical software for OPTICS. However, this method does not provide any advantage over the original DBSCAN method. 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 Developing KDD-Applications Supported by Index Structures (ELKI) (Schubert *et al.* 2015). 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 specification (Figure 19 in Ankerst *et al.* (1999)), while mostly complete, lacks important corrections that otherwise produce artifacts when clustering data. In the original specification of the algorithm, the 'dents' of the ordering structure OPTICS produces are scanned for significant changes in reachability (hence the ξ threshold), where clusters are represented by contiguous ranges of points that are distinguished by $1-\xi$ density-reachability changes in the reachability plot. It is possible, however, after the recursive completion of the `update` algorithm (Figure 7 in Ankerst *et al.* (1999)) that the next point processed in the ordering is not actually within the reachability distance of other members of cluster being currently processed. To account for the missing details described above, Erich Schubert introduces a number of supplemental processing steps, added in the ELKI framework. These changes are only briefly mentioned in the ELKI's release notes[2], and details can only be found by reading the source code. These steps do correct the artifacting 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 version 0.7.0 of the ELKI framework, released in 2015, 16 years after the original publication of OPTICS and the Extract-ξ metho. **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
E-mail: mhahsler@lyle.smu.edu
URL: http://michael.hahsler.net/

Matt Piekenbrock
Department of Computer Science and Engineering
Dept. of Computer Science and Engineering, Wright State University
3640 Colonel Glenn Hwy, Dayton, OH, 45435
E-mail: piekenbrock.5@wright.edu
URL: http://wright.edu/

[2] see https://elki-project.github.io/releases/release_notes_0.7

Derek Doran
Department of Computer Science and Engineering
Dept. of Computer Science and Engineering, Wright State University
3640 Colonel Glenn Hwy, Dayton, OH, 45435
E-mail: derek.doran@wright.edu
URL: http://knoesis.org/doran