

# A Hybrid Approach To Hierarchical Density-based Cluster Selection

Claudia Malzer<sup>1,2,3</sup> and Marcus Baum<sup>2</sup>

<sup>1</sup> HAWK Hochschule für angewandte Wissenschaft und Kunst  
Hildesheim/Holzminden/Göttingen

<sup>2</sup> Data Fusion Group, Institute of Computer Science, University of Göttingen  
{cmalzer,mbaum1}@uni-goettingen.de

<sup>3</sup> Max Planck Institute for Dynamics and Self-Organization, Göttingen

**Abstract.** HDBSCAN is a density-based clustering algorithm that constructs a cluster hierarchy tree and then uses a specific stability measure to extract flat clusters from the tree. We propose an alternative method for selecting clusters from the HDBSCAN hierarchy. Our approach, HDBSCAN( $\hat{\epsilon}$ ), is particularly useful for data sets with variable densities where we require a low minimum cluster size but want to avoid an abundance of micro-clusters in high-density regions. The method uses an additional input parameter  $\hat{\epsilon}$  and acts like a hybrid between DBSCAN\* and HDBSCAN. It can easily be integrated into existing HDBSCAN implementations.

**Keywords:** density-based clustering · hierarchical cluster extraction · HDBSCAN · DBSCAN

## 1 Introduction

Clustering algorithms are used by researchers of various domains to explore and analyze patterns of similarity in their data. While partitioning methods like *k-means* group objects into a predefined number of categories, hierarchical algorithms build a hierarchy of data partitions in order to reveal the intrinsic data structure. For example, single-linkage clustering is a bottom-up hierarchical approach where each object starts as its own cluster, and at each step clusters are merged depending on some distance measure between their closest members.

Density-based clustering is a paradigm where clusters are regarded as data partitions that have a higher density than their surroundings. Hence, a common application scenario is a data distribution where dense concentrations of objects are separated by areas of sparse data. Objects that do not meet a given density criterion are discarded as noise. Those kind of algorithms can be useful in many research fields, but are particularly well suited for spatial data mining [9].

In this paper, we discuss a problem that emerges in data sets with highly variable densities, especially when choosing a low minimum cluster size. In such case, we either completely miss some potentially relevant clusters, or we receive a large number of small clusters in high-density regions that we would have

intuitively regarded as only one or few clusters. We also provide a solution in form of a hierarchical cluster extraction method, called HDBSCAN( $\hat{\epsilon}$ ), which can be viewed as a hybrid between DBSCAN\* (see Section 3) and HDBSCAN. From the HDBSCAN hierarchy we select DBSCAN\* clusters for a fixed user-defined distance threshold, and HDBSCAN clusters from all data partitions not affected by the threshold.

Section 2 below provides a short overview of existing density-based clustering algorithms, with focus on cluster extraction from hierarchical solutions. Section 3 gives a more detailed insight into HDBSCAN, since we will use the hierarchy created by this algorithm as basis for our own cluster extraction method. In Section 4 we illustrate the mentioned problem on a real-life application and discuss how the three density-based clustering algorithms DBSCAN, OPTICS and HDBSCAN handle this case. We then formally introduce our alternative cluster extraction solution for HDBSCAN. Results of experiments with HDBSCAN( $\hat{\epsilon}$ ) on one real and several synthetic data sets are presented and discussed in Section 5. Section 6 concludes with a brief summary and future perspectives.

## 2 Related Work

In the classic density-based algorithm DBSCAN [5], density is defined as having a minimum number of objects (specified by an input parameter *minPts*) within the neighborhood of a certain radius. The size of the radius is specified by the distance threshold parameter  $\epsilon$  (*epsilon*). Connected subsets of objects that fulfill this density criterion are regarded as clusters, all others are discarded as noise. This method allows to detect clusters of arbitrary shape and does not require to specify the number of clusters a priori.

DBSCAN’s major weakness is that its epsilon parameter serves as a *global* density threshold and it is therefore not possible to discover clusters of variable densities. Many DBSCAN variants have been proposed with the aim of overcoming this problem. For example, DECODE [13] is an algorithm for clustering spatial data sets that include point processes of variable densities. A disadvantage of this algorithm, which is based on Markov chain Monte Carlo methods, is its high computational complexity.

A DBSCAN extension that became particularly popular is the algorithm OPTICS by Ankerst et al. [1], which instead follows a hierarchical approach. In contrast to DBSCAN, OPTICS constructs an ordered representation of the data set that allows to explore *all* possible density levels instead of just the data partitions at a single density level. A value for the minimum cluster size is the only required input parameter; specifying an epsilon value is optional and only used to reduce run-time. Visualization techniques such as *reachability plots* can be used for graphical interpretation, and the authors also provide an automatic method for extracting clusters from the ordering. This method requires an input parameter  $\xi$  to control the granularity of separating clusters from each other. Arguing that finding a suitable value for  $\xi$  is not intuitive and mostly a result of trial and error, Sander et al. [14] proposed an alternative method for extracting

clusters from the OPTICS ordering. They removed  $\xi$  and instead rely on a heuristic internal value for cluster separation that is supposed to work well in all experiments.

AUTO-HDS [7], another hierarchical method, was motivated by the analysis of high-dimensional biological data sets such as gene-expression data. Besides a *minPts*-like parameter called  $n_\epsilon$ , the parameter  $r_{shave}$  serves as a smoothing factor to control the number of points to be clustered at each hierarchy level, and the  $n_{part}$  value can further be used to exclude child clusters with less than  $n_{part}$  points. AUTO-HDS relies on a stability measure to extract clusters of variable densities from the hierarchy.

Overall, AUTO-HDS is quite similar to Campello et al.’s HDBSCAN [2]. However, it has been shown that HDBSCAN can outperform both AUTO-HDS and the combination of OPTICS with Sander et al.’s cluster extraction method [2] [10]. HDBSCAN was proposed as an improved extension of DBSCAN and OPTICS for data exploration in diverse research fields. The algorithm requires only a *minPts* value as user input and then simplifies a complex single-linkage hierarchy to a smaller tree of candidate clusters. It also supports a smoothing parameter *min\_samples*, comparable to AUTO-HDS’  $n_{part}$ . A flat clustering solution is extracted based on local cuts through the tree. HDBSCAN’s Python implementation [11] conforms to the widely used *scikit-learn* [12] library and supports a variety of metrics such as cosine similarity and haversine distance.

Because of its many benefits, we use HDBSCAN as basis for our proposed hierarchical cluster extraction method – called HDBSCAN( $\hat{\epsilon}$ ) – and will give a more detailed review of the algorithm in the next section. Note that our HDBSCAN( $\hat{\epsilon}$ ) is not the same as  $\epsilon$ -HDBSCAN by Dockhorn et al. [3], where the authors generated a DBSCAN hierarchy for a fixed epsilon value and then gradually decreased the *minPts* value. The same authors further introduced the *edge quantile* method, which performs local cuts on branches of the DBSCAN hierarchy wherever the 0.95 quantile of edge lengths is exceeded [4]. In contrast, we use a single user-defined cut value that can be viewed as a smoothing factor similar to *min\_samples*, but applies to distance rather than cluster size. Since our method is designed as an extension for the existing HDBSCAN architecture, setting the threshold to 0 always results in the same clustering as the original HDBSCAN.

### 3 The HDBSCAN Algorithm

In DBSCAN, objects with at least *minPts* data points within the epsilon radius are called *core points*. Objects that are no core points themselves, but lie within the epsilon neighborhood of a core point, are called *border points*. HDBSCAN is built on top of a slightly modified version of DBSCAN, DBSCAN\*, which neglects these border points [2]. It can be viewed as a hierarchical DBSCAN\* implementation for all possible epsilon thresholds.

### 3.1 Mutual Reachability Distance

In HDBSCAN, the *core distance*  $d_{core}$  is defined as the distance of an object to its *minPts*-nearest neighbor. However, the constructed hierarchy is based on the *mutual reachability distance*, which for two objects  $x_p, x_q$  is

$$\max\{d_{core}(x_p), d_{core}(x_q), d(x_p, x_q)\}$$

where  $d(x_p, x_q)$  refers to the “normal” distance according to the chosen metric, e.g. Euclidean distance. This approach separates sparse points from other points by at least their core distance. It aims at making the clustering more robust to noise by avoiding strong “single-linkage effects”, i.e. long thin cluster chains that appear when clusters are merged via noise points.

The data set can then be represented by a *mutual reachability graph* with the data objects as vertices and their connections as weighted edges, where the weight of each edge is the mutual reachability distance between the connected pair of points. Using this graph to construct a minimum spanning tree and sorting its edges by mutual reachability distance results in a hierarchical tree structure (*dendrogram*). By choosing an epsilon as horizontal cut value and selecting all clusters with at least *minPts* points at this density level, we could retrieve DBSCAN\* results from the hierarchy.

### 3.2 Condensed Cluster Hierarchy

Since HDBSCAN aims at discovering clusters of variable densities, it instead proceeds to building a simplified version of the complex hierarchy tree, the *condensed cluster tree*. Starting from the root, each cluster split is only regarded as a *true* split if both child clusters contain at least *minPts* objects. If they contain less than *minPts* objects, the cluster is considered as having disappeared at this density level. If only one of the children has less than *minPts* objects, the interpretation is that the parent cluster has simply lost points but still exists. This approach follows the *run pruning* concept by Stuetzle [16] and results in a hierarchy of candidate clusters with variable densities.

### 3.3 Cluster Extraction

Given the condensed cluster tree, one possibility is to simply select all leaf nodes. They represent clusters that cannot be split up any further with respect to *minPts*. This selection method is one of two provided options in HDBSCAN’s Python implementation and results in very fine-grained clusters.

The other option is *eom*, short for *excess of mass*. This method, which refers back to the research by Hartigan [8], is recommended by Campello et al. [2] as the optimal global solution to the problem of finding clusters with the highest *stability*, which they define as

$$S(C_i) = \sum_{x_j \in C_i} (\lambda_{max}(x, C_i) - \lambda_{min}(C_i)) = \sum_{x_j \in C_i} \left( \frac{1}{\epsilon_{min}(x_j, C_i)} - \frac{1}{\epsilon_{max}(C_i)} \right)$$

for a cluster  $C_i$  that appears at density level  $\lambda_{min}(C_i)$  and splits or disappears at density level  $\lambda_{max}(C_i)$ , where  $\lambda = \frac{1}{\epsilon}$  in  $[0, \infty)$ . The authors formalize the optimization problem for maximizing the sum of stabilities as

$$\begin{aligned} \max_{\delta_2, \dots, \delta_k} J &= \sum_{i=2}^k \delta_i S(\mathbf{C}_i) \\ \text{subject to } &\begin{cases} \delta_i \in \{0, 1\}, i = 2, \dots, k \\ \sum_{j \in \mathbf{I}_h} \delta_j = 1, \forall h \in \mathbf{L} \end{cases} \end{aligned}$$

with  $\mathbf{L} = \{h | \mathbf{C}_h \text{ is leaf cluster}\}$  as leaves,  $\mathbf{I}_h$  as the set of clusters on the paths from leaves to the excluded root, and  $\delta_i$  as boolean indicator whether the respective cluster is selected or not.

As a solution to this problem, HDBSCAN's selection algorithm traverses the condensed cluster tree bottom-up and selects the cluster with highest stability on each path.

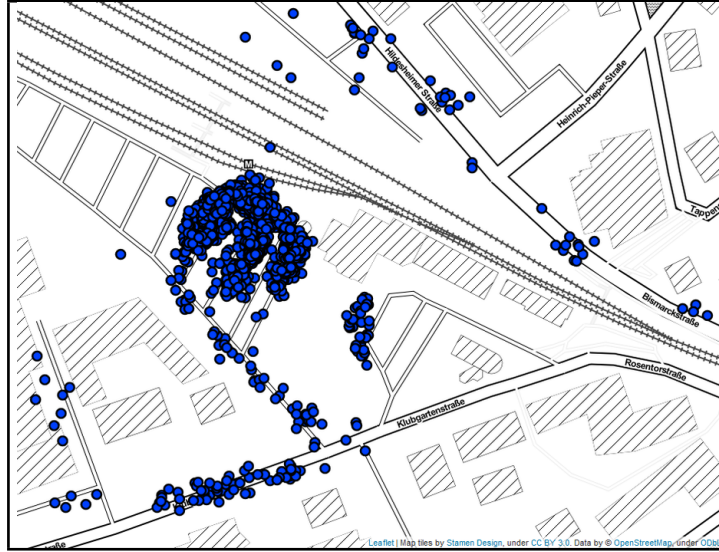
## 4 HDBSCAN( $\hat{\epsilon}$ ): A Threshold for Cluster Splits

In this section, we introduce a new cluster extraction method for the HDBSCAN hierarchy. Our motivation for this approach is given below, followed by a formal definition and algorithmic solution.

### 4.1 Motivation

HDBSCAN is a powerful clustering algorithm for unsupervised data exploration. However, for some applications, the single input parameter *minPts* might not be sufficient to discover the clusters that best represent the underlying data structure. In particular, let us consider a large data set distributed such that there is a high number of very dense objects in some areas, and only few objects in other areas. If we were only interested in the highly populated areas, HDBSCAN would give us good results for a *minPts* value large enough to declare sparse regions as noise and dense regions as clusters. In some cases, however, we do not want all observations in sparse environments to be marked as noise: these areas might naturally contain fewer objects in total, but small yet dense groups of objects that do exist might be just as relevant as the ones in regions with a large amount of data.

Figure 1 demonstrates such a scenario. It shows around 2800 GPS data points on a map extract, representing recorded pick-up and drop-off locations from a door-to-door demand-responsive ride pooling system. Our aim was to assign addresses requested by customers to the closest areas where ride pooling vehicles were actually able to stop in the past, i.e. in compliance with traffic rules and available space. The largest (visual) data cluster can be found around the train station. Smaller clusters are placed along the streets, depending on the requested

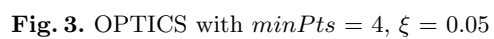


**Fig. 1.** The sample data set. The highest concentration of GPS data points represents the area around a train station.

location in form of a postal address or point of interest. Since we are considering a door-to-door system where customers are not bound to collective pick-up or drop-off locations, even small groups of 4 or 5 points are of interest to us.

Figure 2 presents the clustering result with HDBSCAN’s default selection method *eom*, from now on referred to as HDBSCAN(*eom*). Using  $minPts = 4$ , the algorithm successfully discovers all the small clusters while declaring obvious outliers or groups with less than 4 points as noise (depicted as light gray points). However, in the area with highest concentration of data points – the train station –, it generates a very large number of micro-clusters. In our case, this is not what we want: we would prefer one or only few clusters representing the location. This would be possible by increasing  $minPts$  or the smoothing parameter  $min\_samples$ , but with the trade-off of losing small clusters in less dense areas or merging them into other clusters separated by a relatively large distance.

We clustered the same data set with DBSCAN and OPTICS, both from Python’s *scikit-learn* library. The  $minPts$  parameter was set to 4 in both cases. For OPTICS, we tried  $\xi$  values between 0.03 to 0.05, and for DBSCAN, we tried  $\epsilon$  values between 3 and 10 meters (haversine distance). In each case, we eventually chose a value that we intuitively considered to produce the best results. Figure 3 shows OPTICS clusters for  $\xi = 0.05$ , which are comparable to HDBSCAN. Figure 4 depicts DBSCAN results for  $\epsilon = 5$  meters, which seems clearly better suitable for our application. However, DBSCAN neglects potentially important clusters with densities beyond the chosen epsilon value. In particular, this applies to the two groups of objects on the bottom-left and another one on the far-right.



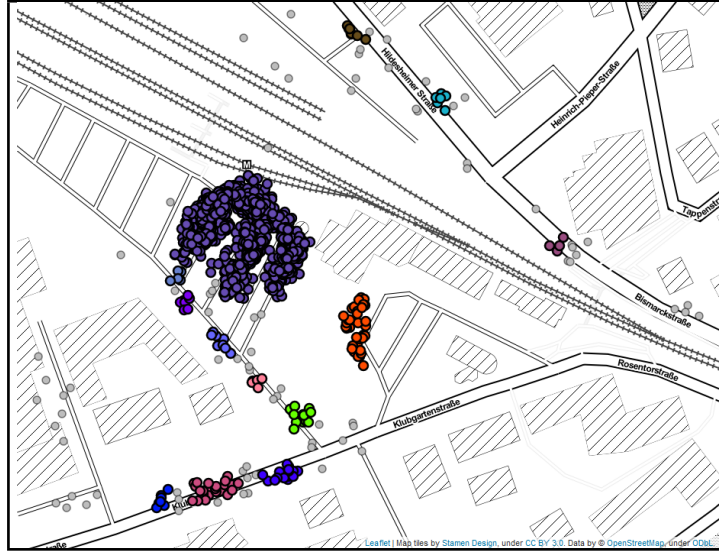


Fig. 4. DBSCAN with  $\minPts = 4$ ,  $\epsilon = 5$  meters

Selecting a larger  $\epsilon$  would cover these objects, but at the same time merge some of the other clusters. Note that this demonstration is based on only a small sample. Applying DBSCAN to a larger data set increases its tendency to single-linkage effects, such as extending the train station cluster down the streets.

What we are looking for is basically a combination between HDBSCAN and DBSCAN. Instead of performing a horizontal cut through the entire HDBSCAN hierarchy and discarding all clusters beyond this line, we just want to prevent clusters below a given threshold from splitting up any further. We could then still select regular HDBSCAN clusters from data partitions not affected by the threshold. All others would be DBSCAN (or, to be precise, DBSCAN\*) clusters, and we are willing to accept limitations like the possibility of overlooking a few relevant subclusters for the benefit of removing many redundant micro-clusters.

## 4.2 Formal Definition

We adjust the definitions used by [2] so that they fit our new requirement. We therefore introduce the notions of *epsilon stable* and *epsilon stability*.

**Definition 1 (Epsilon stable).** A cluster  $C_i$  with  $i = \{2, \dots, k\}$  is called *epsilon stable* if  $\epsilon_{\max}(C_i) > \hat{\epsilon}$  for a given  $\hat{\epsilon} > 0$ .

As explained earlier,  $\lambda_{\min}(C_i) = \frac{1}{\epsilon_{\max}(C_i)}$  is the density level at which cluster  $C_i$  appears. Note that this is equal to the level at which it split off its parent cluster. Hence we call a cluster epsilon stable if the split from its parents occurred at a distance above our threshold  $\hat{\epsilon}$  (or below the level  $\lambda_{\min}(C_i)$ , respectively) and formally define epsilon stability as follows:



**Definition 2 (Epsilon Stability).**

$$ES(C_i) = \begin{cases} \lambda_{min}(C_i) & \text{if } C_i \text{ is epsilon stable} \\ 0 & \text{otherwise} \end{cases}$$

If we select the cluster with highest epsilon stability on each path of the HDBSCAN condensed hierarchy tree, we end up with all the clusters that we do not want to split up any further w.r.t.  $\hat{\epsilon}$  and  $minPts$ . Their parents split up at some distance  $\epsilon_{min} > \hat{\epsilon}$ , which is equal to the  $\epsilon_{max}$  value on the level where their children appear. While those children are still valid clusters, they are not allowed to split up themselves because either they are leaf clusters or the level  $\lambda_{max} = \frac{1}{\epsilon_{min}}$  at which the split would happen is above the threshold.

This leads to the optimization problem of maximizing the sum of epsilon stabilities, i.e. finding the maximum epsilon stable cluster on each path from leaf to root.

**Definition 3 (Optimization Problem).**

$$\begin{aligned} \max_{\delta_2, \dots, \delta_k} J &= \sum_{i=2}^k \delta_i ES(C_i) \\ \text{subject to } &\begin{cases} \delta_i \in \{0, 1\}, i = 2, \dots, k \\ \sum_{j \in I_h} \delta_j = 1, \forall h \in \mathbf{L} \end{cases} \end{aligned}$$

with  $\mathbf{L} = \{h | C_h \text{ is leaf cluster}\}$  as leaves,  $I_h$  as the set of clusters on the paths from leaves to the excluded root, and  $\delta_i$  as boolean indicator whether the respective cluster is selected or not.

**4.3 Optimization Algorithm**

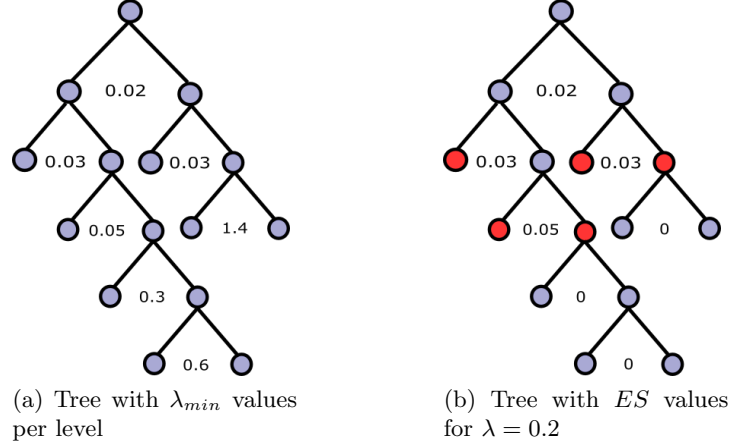
The pseudo code in Algorithm 1 demonstrates how to solve the problem in Definition 3. Initially, we mark all leaves of our HDBSCAN cluster hierarchy as selected. For each leaf, if it either has been previously marked as not being a cluster, or if  $\lambda_{min} \leq \frac{1}{\hat{\epsilon}}$  for input parameter  $\hat{\epsilon}$ , then this node is epsilon stable and we continue with the next leaf. Otherwise, we traverse upwards until we find an ascendant that split off its parent at a density level  $\lambda_{min} \leq \frac{1}{\hat{\epsilon}}$ . If we find one before reaching the root, we select it as a cluster and unselect all of its descendants.

Since  $ES(C_i)$  decreases as we traverse up the tree, this procedure ensures that we are selecting the maximum epsilon stable cluster on each path. This is equal to selecting the first cluster on the path that “was born” at a distance greater than  $\hat{\epsilon}$ , and thus the first cluster that is not allowed to split up.

The concept is illustrated in Figure 5, which shows the HDBSCAN cluster hierarchy for a small sample data set. The tree on the left is annotated with

**Algorithm 1** Solution to the Optimization Problem

- 
1. Initialize  $\delta_2 = \dots = \delta_k = 1$
  2. Do bottom-up from all leaves (excluding the root):
    - 2.1. If  $ES(C_i) > 0$  or  $\delta_{C(i)} = 0$ , continue
    - 2.2. Else if  $ES(C_i) = 0$  and  $ES(C_{PARENT(i)}) > 0$ , set  $\delta_{PARENT(i)} = 1$  and set  $\delta_{(.)} = 0$  for all nodes in  $C_{PARENT(i)}$ 's subtree
- 



**Fig. 5.** The HDBSCAN cluster tree in Figure 5a is annotated with the  $\lambda_{min}$  value per level. Figure 5b shows the same tree annotated with epsilon stability values for a given threshold  $\hat{\epsilon} = 5$  (or  $\lambda = 0.2$ , respectively). On each path, the cluster with maximum epsilon stability is highlighted in red.

$\lambda_{min}$  values. On the right, the same tree is annotated with corresponding epsilon stability values for an input parameter of  $\hat{\epsilon} = 5$  meters, or  $\lambda = \frac{1}{\hat{\epsilon}} = 0.2$ . According to Definition 2, each cluster on a level that is not epsilon stable is set to 0. All others receive their  $\lambda_{min}$  as epsilon stability value.

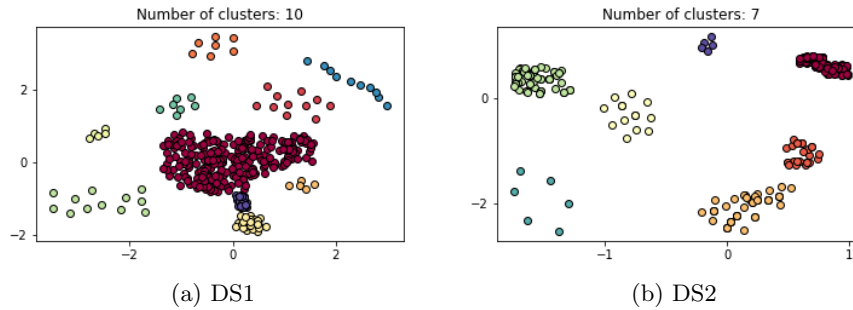
It can be seen that the density levels with  $\lambda_{min}$  values of 0.6, 0.3 and 1.4 exceed the threshold of 0.2. This indicates that the parents of the clusters on these levels split up at a distance lower than 5 meters. By starting from the leaves and selecting the ascendant with maximum epsilon stability value on each path, we receive the final set of clusters (highlighted in red).

Note that Algorithm 1 is an extended version of HDBSCAN's *leaf* selection method. Alternatively, it can be build on top of the *eom* method. To do this, we simply need to start our algorithm with the nodes selected by HDBSCAN(*eom*).

## 5 Experiments and Discussion

We applied HDBSCAN( $\hat{\epsilon}$ ) to two synthetic data sets, illustrated in Figure 6 with different (random) colors representing their true labels. Both data sets contain

clusters of variable shapes and densities and serve as examples for cases where HDBSCAN(eom) leads to an abundance of micro-clusters. For comparison, each data set was also clustered with scikit-learn implementations of OPTICS and DBSCAN. The minimum cluster size was set to a fixed value of 4 in all experiments. The epsilon threshold in DBSCAN and HDBSCAN( $\hat{\epsilon}$ ) was in each case set to a value that results in the most accurate clustering with respect to the ground truth data. The  $\xi$  value for OPTICS was set to 0.05 in both cases. Lower or higher values do not seem to improve the result.

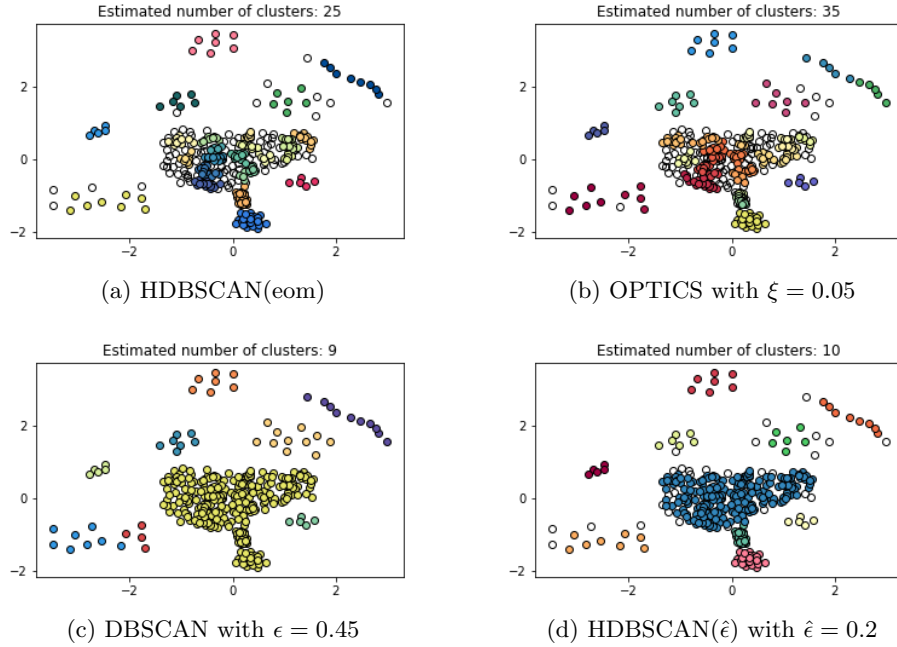


**Fig. 6.** Two synthetic data sets with 10 and 7 clusters, respectively.

As depicted in Figure 7, HDBSCAN(eom) returns 25 clusters for DS1, where most clusters are formed within the high-density region in the center. OPTICS provides a similar result. As long as epsilon is chosen large enough, DBSCAN is able to recognize the center region as a single cluster, plus smaller clusters in the neighborhood. On the downside, two dense data partitions on the bottom are not separated from the large cluster. Unlike DBSCAN, choosing a threshold low enough to separate these partitions does not prevent HDBSCAN( $\hat{\epsilon}$ ) from finding the other clusters, although a few single points are marked as noise.

Clustering results for DS2 are presented in Figure 8. Again, HDBSCAN(eom) successfully discovers clusters of variable densities, but breaks up regions with a large number of dense data points into multiple micro-clusters. OPTICS results are no improvement over HDBSCAN(eom). DBSCAN almost achieves the desired output, but we could not find an epsilon value that recognizes the cluster on the bottom-left while keeping the remaining clusters separated. Figure 8c shows the result of an epsilon value (0.38) that is just small enough not to merge these clusters, which leaves the bottom-left cluster as noise.

HDBSCAN( $\hat{\epsilon}$ ) instead achieves the desired clustering using an  $\hat{\epsilon}$  value of 0.1 (Figure 8d). This result is represented by the Adjusted Rand Index (ARI) as seen in Table 1. Note that the Adjusted Rand Index, a commonly used cluster validation measure, does not consider noise. For this reason, DBSCAN and HDBSCAN( $\hat{\epsilon}$ ) both achieve a perfect ARI score of 1 for DS2, although DBSCAN



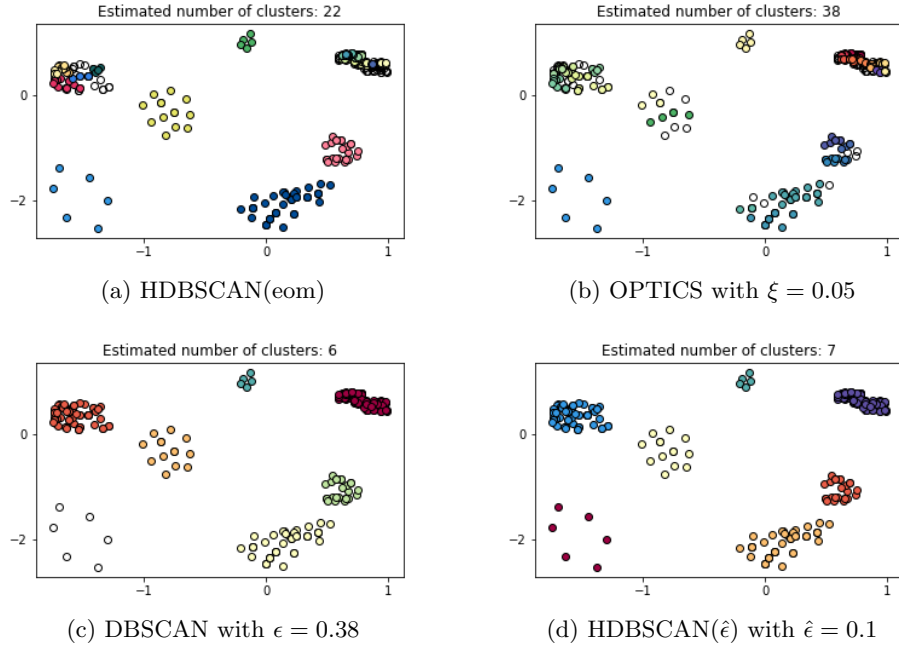
**Fig. 7.** Clustering results for DS1. Colorless points represent noise.

marks the bottom-left cluster as noise. Information about noise is included in the %c value, which refers to the fraction of data points not regarded as noise.

**Table 1.** Clustering results for synthetic data sets DS1 and DS2 in terms of Adjusted Rand Index (ARI) and percentage of data points not marked as noise (%c)

Data Set	HDBSCAN(eom)		OPTICS		DBSCAN		HDBSCAN( $\hat{\epsilon}$ )	
	ARI	%c	ARI	%c	ARI	%c	ARI	%c
DS1	0.15	0.74	0.10	0.75	0.52	1	0.86	0.92
DS2	0.28	0.75	0.11	0.78	1	0.98	1	1

In addition, results for four common toy data sets are shown in Figure 9. The data sets Spiral, Jain and Flame were taken from [6]. The “anisotropically distributed data set” was generated using code from [15], with the only difference that we reduced the number of samples from 1500 to 150 and *minPts* from 20 to 4. With well-chosen epsilon values, DBSCAN shows a good performance on all of these data sets, while HDBSCAN(eom) tends to create too many clusters for *minPts* = 4. Therefore,  $\hat{\epsilon}$  thresholds were chosen to correct this behavior with HDBSCAN( $\hat{\epsilon}$ ). In case of the Flame data set, HDBSCAN(eom) already

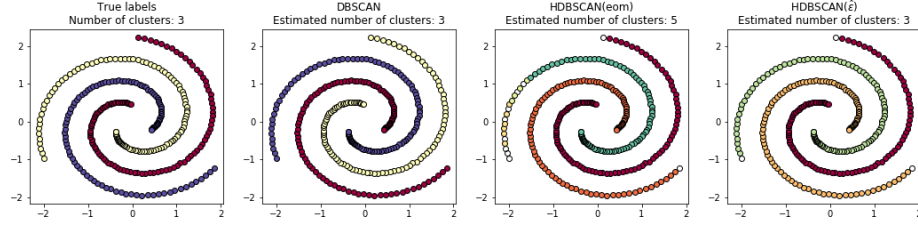


**Fig. 8.** Clustering results for DS2. Colorless points represent noise.

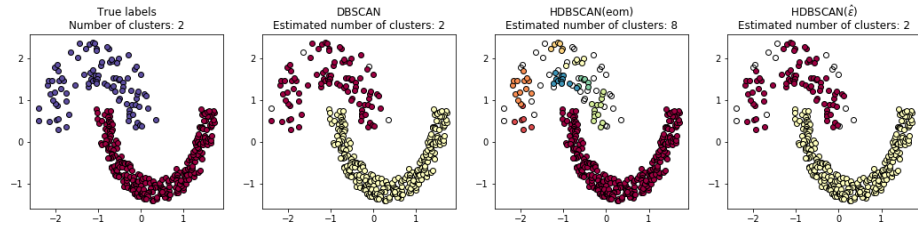
achieves a good result, but the example demonstrates that a threshold value of 0 for  $\text{HDBSCAN}(\hat{\epsilon})$  simply generates  $\text{HDBSCAN}(\text{eom})$  clusters. However, be aware that unlike the data sets DS1 and DS2 as shown above, none of these toy data sets are suitable for demonstrating the biggest advantage of  $\text{HDBSCAN}(\hat{\epsilon})$ , which is the fact that  $\text{HDBSCAN}$  and  $\text{DBSCAN}^*$  results can be combined.

Finally, we applied our  $\text{HDBSCAN}(\hat{\epsilon})$  algorithm to the sample of GPS data introduced in Section 4.1. Figure 10 shows the result for  $\hat{\epsilon} = 5$  meters. Compared to  $\text{DBSCAN}$  in Figure 4 with the same epsilon value, and  $\text{HDBSCAN}(\text{eom})$  in Figure 2, we notice that we indeed receive a combination of both. We no longer lose clusters of variable densities beyond the given epsilon, but at the same time avoid the abundance of micro-clusters in the original  $\text{HDBSCAN}$  clustering, which was an undesired side-effect of having to choose a low  $\text{minPts}$  value. Note that for the given parameter setting, running  $\text{HDBSCAN}(\hat{\epsilon})$  based on  $\text{HDBSCAN}(\text{eom})$  or based on  $\text{HDBSCAN}(\text{leaf})$  would not make any difference: the  $\hat{\epsilon}$  threshold neutralizes the effect of  $\text{HDBSCAN}(\text{eom})$ 's stability calculations. For a lower threshold, e.g.  $\hat{\epsilon} = 3$ , some minor differences can be noticed.

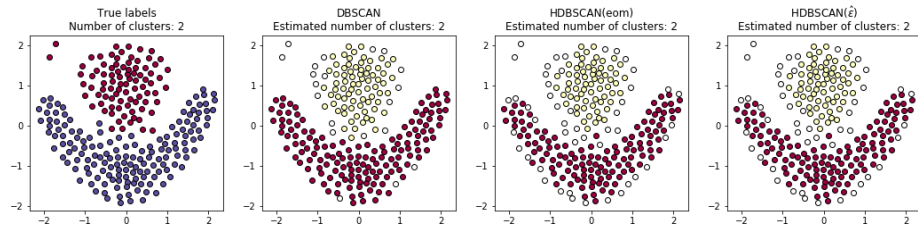
In general, the most suitable  $\hat{\epsilon}$  value is certainly not always easy to choose. For spatial data like GPS points, it is quite intuitive to decide on a distance threshold, but in higher-dimensional data, this becomes more difficult. Another limitation is that cutting the hierarchy at a fixed threshold can neglect meaningful subcluster structures. In other words,  $\text{HDBSCAN}(\hat{\epsilon})$  inherits  $\text{DBSCAN}$ 's



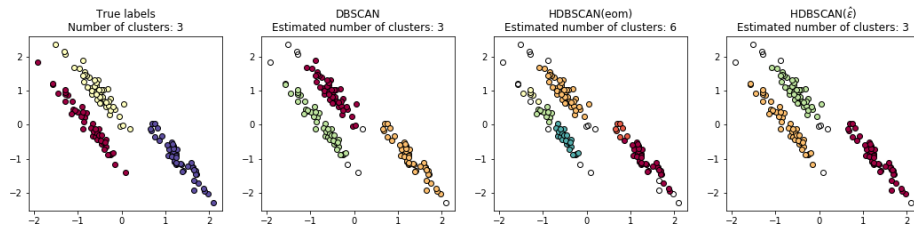
(a) Spiral data set with  $\epsilon/\hat{\epsilon} = 0.3$



(b) Jain data set with  $\epsilon/\hat{\epsilon} = 0.315$



(c) Flame data set with  $\epsilon = 0.28$  (DBSCAN),  $\hat{\epsilon} = 0$  (HDBSCAN( $\hat{\epsilon}$ ))



(d) Anisotropically distributed data set with  $\epsilon/\hat{\epsilon} = 0.3$

**Fig. 9.** Clustering common toy data sets with  $minPts = 4$ .

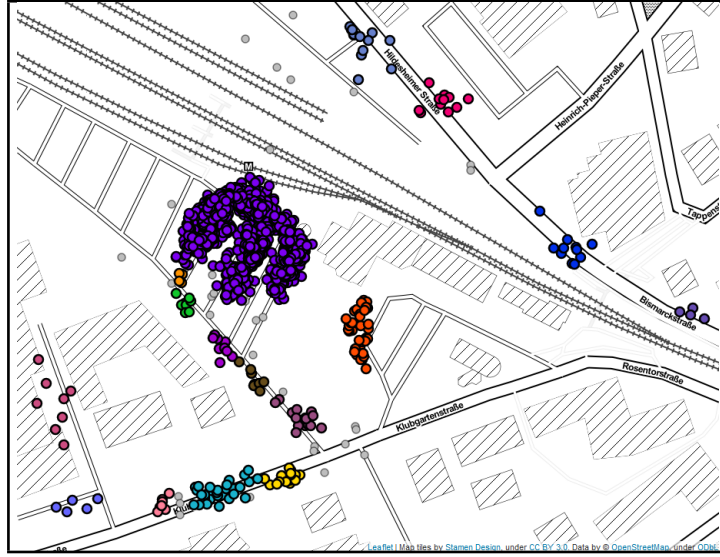


Fig. 10. HDBSCAN( $\hat{\epsilon}$ ) with  $\minPts = 4$ ,  $\hat{\epsilon} = 5$  meters

shortcomings wherever it uses a fixed value to select clusters. However, as the example above shows, there are applications where HDBSCAN can truly benefit from such a threshold. In particular, for scenarios where we are interested in both small and large clusters and therefore choose a low minimum cluster size, it is helpful to have a parameter that can be tuned such that an abundance of micro-clusters in high-density areas is avoided.

## 6 Summary and Conclusion

We introduced a cluster extraction method that acts like a hybrid between DBSCAN\* and HDBSCAN: for data partitions affected by a given distance threshold, we extract DBSCAN\* results, for all others we select HDBSCAN clusters either according to *eom* or *leaf* selection mode. Our method can easily be integrated into existing HDBSCAN implementations. It is already available for use as part of the scikit-learn compatible Python implementation <sup>1</sup>.

We believe that this extension will prove to be valuable particularly in clustering spatial data, but it could be applied to different kind of data as well. Future work might consider alternative clustering approaches for environments where data partitions are very variable in terms of size and density. It might also be worth exploring new semi-supervised clustering methods where the extraction of clusters is influenced by background information such as geographic context.

<sup>1</sup> <https://github.com/scikit-learn-contrib/hdbscan>

## Acknowledgements

The authors are grateful for the insight and remarks provided by Alexander Dockhorn (Otto von Guericke University Magdeburg). We thank Armin Hahn (Max Planck Institute for Dynamics and Self-Organisation), Bernd Stock (HAWK Hochschule für angewandte Wissenschaft und Kunst) and Stephen Boahen Asabere (University of Göttingen) for their useful comments and Michael Schäfer for providing GPS records from the ride pooling system “EcoBus” at Max Planck Institute for Dynamics and Self-Organisation.

## References

1. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure. In: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data. pp. 49–60. SIGMOD ’99, ACM, New York, NY, USA (1999)
2. Campello, R.J.G.B., Moulavi, D., Sander, J.: Density-based clustering based on hierarchical density estimates. In: Pei, J., Tseng, V.S., Cao, L., Motoda, H., Xu, G. (eds.) *Advances in Knowledge Discovery and Data Mining*. pp. 160–172. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
3. Dockhorn, A., Braune, C., Kruse, R.: An alternating optimization approach based on hierarchical adaptations of dbscan. In: 2015 IEEE Symposium Series on Computational Intelligence. pp. 749–755 (Dec 2015).
4. Dockhorn, A., Braune, C., Kruse, R.: Variable density based clustering. 2016 IEEE Symposium Series on Computational Intelligence (SSCI) pp. 1–8 (2016)
5. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. pp. 226–231. KDD’96, AAAI Press (1996)
6. Fránti, P., Sieranoja, S.: K-means properties on six clustering benchmark datasets (2018), <http://cs.uef.fi/sipu/datasets/>
7. Gupta, G., Liu, A., Ghosh, J.: 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 (April 2010)
8. Hartigan, J.A.: Estimation of a convex density contour in two dimensions. *Journal of the American Statistical Association* **82**(397), 267–270 (1987)
9. Kriegel, H.P., Kröger, P., Sander, J., Zimek, A.: Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **1**(3), 231–240 (2011)
10. McInnes, L., Healy, J.: Accelerated hierarchical density based clustering. In: 2017 IEEE International Conference on Data Mining Workshops (ICDMW). pp. 33–42 (Nov 2017)
11. McInnes, L., Healy, J., Astels, S.: hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software* **2**(11), 205 (2017)
12. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011)



13. Pei, T., Jasra, A., Hand, D.J., Zhu, A.X., Zhou, C.: Decode: a new method for discovering clusters of different densities in spatial data. *Data Mining and Knowledge Discovery* **18**(3), 337 (Nov 2008)
14. Sander, J., Qin, X., Lu, Z., Niu, N., Kovarsky, A.: Automatic extraction of clusters from hierarchical clustering representations. In: *Proceedings of the 7th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*. pp. 75–87. PAKDD '03, Springer-Verlag, Berlin, Heidelberg (2003)
15. Scikit-learn: Comparing different clustering algorithms on toy datasets (2019), [https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_cluster\\_comparison.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html)
16. Stuetzle, W.: Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. *Journal of Classification* **20**(1), 025–047 (May 2003)