Noname manuscript No.
(will be inserted by the editor)

Scalable Density-Based Clustering with Quality Guarantees using Random Projections

Johannes Schneider · Michail Vlachos

the date of receipt and acceptance should be inserted later

Abstract Clustering offers significant insights in data analysis. Density-based algorithms have emerged as flexible and efficient techniques, able to discover high-quality and potentially irregularly shaped clusters. Here, we present scalable density-based clustering algorithms using random projections. Our clustering methodology achieves a speedup of two orders of magnitude compared with equivalent state-of-art density-based techniques, while offering analytical guarantees on the clustering quality. Moreover, it does not introduce difficult to set parameters. We provide a comprehensive analysis of our algorithms and comparison with existing density-based algorithms.

1 Introduction

Clustering is an important operation for knowledge extraction. Its objective is to assign objects to groups such that objects within a group are more similar than objects across different groups. Subsequent inspection of the groups can provide important insights, with applications to pattern discovery [27], data summarization/compression [17] and data classification [6]. In the field of clustering, computationally light techniques, such as k-Means, are typically of heuristic nature, may require non-trivial parameters, such as the number of clusters, and often rely on stringent assumptions, such as the cluster shape. Density-based clustering algorithms have emerged as both high-quality and efficient clustering techniques with solid theoretical foundations on density estimation [12]. They can discover clusters with irregular shapes and only require

J. Schneider
ABB Corporate Research, Switzerland
E-mail: johannes.schneider@ch.abb.com

M. Vlachos
IBM Research - Zurich
E-mail: mvl@zurich.ibm.com

parameters that are relatively easy to set (e.g., minimum number of points per cluster). They can also help to assess important dataset characteristics, such as the *intrinsic* density of data, which can be visualized via reachability plots.

In this work, we extend the state of the art in density-based clustering techniques by presenting algorithms that significantly improve runtime, while providing analytical guarantees on the preservation of cluster quality. Furthermore, we highlight weaknesses of current clustering algorithms with respect to parameter dependency. Our performance gains and our quality guarantees are achieved through the use of random projections. A key theoretical result of random projections is that, in expectation, distances are preserved. We exploit this in a pre-processing phase to partition objects into sets that should be examined together. The resulting sets are used to compute a new type of density estimate through sampling.

Our algorithm requires the setting of only a single parameter, namely, the minimum number of points in a cluster, which is customarily required as input in density-based techniques. In general, we make the following **contributions**:

- We show how to use random projections to improve the performance of existing density-based algorithms, such as OPTICS and its performance-optimized version DeLi-Clu without the need to set any parameters. We introduce a new density estimate based on computing average distances. We also provide guarantees on the preservation of cluster quality and runtime.
- The algorithm is evaluated extensively and yields performance gains of two orders of magnitude with *provable* degree of distortion on the clustering result compared with prevalent density-based approaches, such as OPTICS.

2 Background and Related Work

The majority of density-based clustering algorithms follow the ideas presented in DBSCAN [10], OPTICS [3] and DENCLUE [13]. Our methodology is more similar in spirit to OPTICS, but relaxes several notions, such as the construction of neighborhood. The end result is a scalable density-based algorithm even without parallelization.

DBSCAN was the first influential approach for density-based clustering in the data-mining literature. Among its shortcomings are flat (not hierarchical) clustering, large complexity, and the need for several parameters (cluster radius, minimum number of objects). OPTICS overcame several of these weaknesses by introducing a variable density and requiring the setting of only one parameter (density threshold). OPTICS does not explicitly produce a data clustering but only a cluster ordering, which is visualized through *reachability plots*. Such a plot corresponds to a linear list of all objects examined, augmented by additional information, i.e., the reachability distance, that represents the intrinsic hierarchical cluster structure. Valleys in the reachability plot can be considered as indications of clusters. OPTICS has a complexity

that is on the order of $O(N \cdot |\text{neighbors}|)$, which can be as high as $O(N^2)$ in the worst case, or $O(N \log N)$ in the presence of an index (discounting for the cost of building and maintaining the actual index). A similar complexity analysis applies for DBSCAN.

DENCLUE capitalizes on kernel density estimation techniques. Performance optimizations have been implemented in DENCLUE 2.0 [12] but its asymptotic complexity is still quadratic.

Other approaches to expedite the runtime of density-based clustering techniques involve implementations in Hadoop or using Graphical Processing Units (GPUs). For example, Cludoop [28] is a Hadoop-based density-based algorithm that reports an up to fourfold improvement in runtime. Boehm et al. [5] presented CUDA-DClust, which improves the performance of DBSCAN using GPUs. They report an improvement in runtime of up to 15 times. G-DBSCAN [2] and CudaSCAN [19] are recent GPU-driven implementations of DBSCAN, and report an improvement in runtime of 100 and 160 times, respectively. Our approach uses random projections to speed up the execution, while at the same time having provable cluster quality guarantees. It exhibits an equivalent or more speedup than the above parallelization approaches, without the need for distributed execution, thus lending itself to a simpler implementation.

Random-projection methodologies have been also used to speed up density-based clustering. For example, the algorithm in [25] leverages the observation that for high-dimensional data and a small number of clusters, it is possible to identify clusters based on the density of the projected points on a randomly chosen line. We do not capitalize on this observation. We attempt to determine neighborhood information using recursively applied random projections. The protocol in [25] is of "heuristic nature", as attested by the original authors, so it does not provide any quality guarantees. It runs in time $O(n^2 \cdot N + N \cdot \log N)$, where N points are projected onto n random lines. It requires the specification of several parameters, whereas our approach is parameter-light.

Randomly projected k-d-trees were introduced in [7]. A k-d-tree is a spatial data structure that splits data points into cells. The algorithm in [7] uses random projections to partition points recursively into two sets. Our algorithm `Partition` shares this methodology, but uses a simpler splitting rule. We just select a projected point uniformly at random, whereas the splitting rule in [7] requires finding the median of all projected points and using a carefully crafted jitter. Furthermore, we perform multiple partitionings. The purpose of [7] is to serve as an indexing structure. Retrieving the k-nearest neighbors in a k-d tree can be elaborate and suffers heavily from the curse of dimensionality. To find the nearest neighbor of a point, it may require to look at several branches (cells) in the tree. The cardinality of the branches searched grows with the dimensions. Even worse, computing k-nearest neighbors only with respect to OPTICS would mean that all information between cluster distances would be lost. More precisely, for any point of a cluster, the k-nearest neighbors would always be from the same cluster. Therefore, only knowing the k-nearest neighbors is not sufficient for OPTICS. To the best of our knowledge, there is no indexing structure that supports finding distances between two clusters

(as we do). Nevertheless, an indexing structure such as [7] or the one used in [1] can prove valuable, but might come with significant overhead compared with our approach. We get neighborhood information directly using small sets. Furthermore, we are also able to obtain distances between clusters.

Random projections have also been applied to hierarchical clustering [22], i.e., single and average linkage clustering (more precisely, Ward's method). To compute a single linkage clustering it suffices to maintain the nearest neighbor that is not yet in the same cluster for each point. To do so, [22] runs the same partitioning algorithm as the one used here. In contrast to this work, it computes all pairwise distances for each final set of the partitioning. Because this requires quadratic time in the set size, it is essential to keep the maximum possible set size, $minPts$, as small as possible. In contrast to this work, where $minPts$ is related to the density parameter used in OPTICS, there is no relation to any clustering parameter. In fact, $minPts$ might be increased during the execution of the algorithm. If a shortest edge with endpoints A,B is "unstable", meaning that the two points A and B do not co-occur in many final sets, then $minPts$ is increased. In this work there is no notion of neighborhood stability.

Multiple random projections onto one-dimensional spaces have also been used for SVM [20]. Note that for SVMs a hyperplane can be defined by a vector. The naive approach tries to guess the optimal hyperplane for an SVM using random projections. The more sophisticated approach uses local search to change the hyperplane, coordinate by coordinate.

Projection-indexed nearest neighbors have been proposed in [9] for outlier detection. First, they identify potential k-nearest-neighbor candidates in a reduced dimensional space (spanned by several random projections). Then, they compute distances to this nearest-neighbor candidates in the original space to select the k-nearest-neighbors. In contrast, we perform (multiple) recursive partitionings of points using random projections to identify potential nearest neighbors.

Locality Sensitive Hashing (LSH) [8] also employs random projections. LSH does not perform a recursive partitioning of the dataset as we do, but splits the entire data set into bins of fixed width. It conducts multiple of these partitionings. Furthermore, in contrast to our technique, it requires several parameters, such as width of a bin, number of hash tables, and number of projections per hash value. These parameters typically require knowledge of the dataset for proper tuning.

3 Our approach

In density-based clustering, a key step is to discover the *neighborhood* of each object to estimate the local density. Traditional density-based clustering algorithms, such as OPTICS, may exhibit limited scalability partially because of the expensive computation of neighborhood. Other approaches, such as DeLi-Clu [1], use indexing techniques to speed up the neighborhood discovery

process. As discussed in the related work, spatial indexing approaches are not tailored towards the scenario of OPTICS and require fast k-nearest neighbor retrieval but also distance computation between (distant) points of different clusters.

Our approach capitalizes on a random-projection methodology to create a partitioning of the space from which the neighborhood is produced. We explain this step in Section 4. The intuition is that if an object resides in the neighborhood of another object across multiple projections, then it belongs to that object's neighborhood. The majority of random-projection methodologies project high-dimensional data into a lower dimensionality d that does not depend on the original dimensionality, but is logarithmic to the dataset size [16]. In contrast, we run computations directly on multiple one-dimensional projections. This allows us to work on a very reduced space, in which operations, such as neighborhood construction, can be executed very efficiently. Our neighborhood construction is fast because it only requires *linear time* in the number of points for each projection. Note that a naive scheme looking at pairs of neighboring points would require quadratic time.

After the candidate neighboring points of each object are computed, see Section 5, the local density is estimated. This is described in Section 6. We prove that the local density computed using our approach is an $O(1)$-approximation of the core density calculated by the algorithm used in OPTICS given weak restrictions on the neighborhood size (depending on the distance). This essentially allows us to compute reachability plots equivalent to those of OPTICS, but at a substantially lower cost. Finally, in Section 8, we show empirically that our approach is significantly faster than existing density-based clustering algorithms.

This work represents an extension of [21]. We augment our previous work by formally stating the proofs for the theorems presented and including additional comparisons with existing density-based clustering techniques. We also make the source-code for our approach available in the public domain.

Naturally, our approach and the related proofs are focused on Euclidean distances, because random projections conserve the Euclidean distance.

3.1 Preliminaries

We are given a set of N points $\mathcal{P}$ in the d-dimensional Euclidean space, i.e., for a point $P \in \mathcal{P}$ it holds $P \in \mathbb{R}^d$. We use the term *whp*, i.e., with high probability, to denote probability $1 - 1/N^c$ for an arbitrarily large constant c. The constant c (generally) also occurs as a factor hidden in the big O-notation. We often use the following Chernoff bound:

Theorem 1 *The probability that the number X of occurred independent events $X_i \in \{0, 1\}$, i.e., $X := \sum_i X_i$, is not in $[(1 - c_0)\mathbb{E}[X], (1 + c_1)\mathbb{E}[X]]$ with $c_0 \in]0, 1]$ and $c_1 > 0$ can be bounded by*

$$p(X \leq (1 - c_0)\mathbb{E}[X] \vee X \geq (1 + c_1)\mathbb{E}[X]) < 2e^{-\mathbb{E}[X] \cdot \min(c_0, c_1)^2/3}$$

Symbol	Explanation
P, T, Q	Points in Euclidean space $\mathbb{R}^d$
$\mathcal{P}, \mathcal{C}, \mathcal{S}$	Set of points
L	Randomly chosen line
$\mathcal{L}$	Sequence of lines $(L_0, L_1, ...)$
$\mathfrak{S}, \mathfrak{W}$	Set of sets of points
$Davg(A)$	Average distance of A to nearby points, see Definition 2
$\mathcal{N}(A)$	Neighbors of A, see Algorithm 3
$\mathcal{N}_f(A)$	subset of neighbors of A, see Definition 1
$\mathcal{N}(A, r)$	All points $B \in \mathcal{P}$ within distance r from A
$\tilde{D}_k(A)$	Distance to k nearest point in $\mathcal{N}(A)$ computed using Algorithm 3
$D_k(A)$	Distance to k nearest point among all points $\mathcal{S}$
$minPts$	Parameter of OPTICS stating the number of points used for the density estimate, i.e., the distance to the $minPts$ nearest
$\overline{D}_{minPts}(A)$	Denote the average distance of the $minPts$ nearest points to a point A as $\overline{D}_{minPts}(A) := \sum_{B \in \mathcal{N}(A, D_{minPts}(A))} D(A,B)/\lvert\mathcal{N}(A, D_{minPts}(A))\rvert$. Generally, $\lvert\mathcal{N}(A, D_{minPts}(A))\rvert = minPts$, except if there are multiple points at the same distance $D_{minPts}(A)$ from A.
$minSize$	Parameter of the partitioning process stating the minimum size for which a set is split
c_m	Fixed analysis constant; It relates the partitioning parameter $minSize$ for splitting the point set and the density parameter $minPts$: $minSize = c_m \cdot minPts$; $c_m \geq 1$
r	Distance to the $c_m minPts$ nearest neighbor, i.e., $r := D_{c_m \cdot minPts}(A)$ in the analysis; otherwise, just the distance to the $minPts$ closest neighbor.
$c_0, ..., c_3$	Constants in basic probability bounds
c_L	Fixed analysis constant; for a partitioning of the entire point set, we need up to $c_L \log N$ random lines
c_d	Fixed analysis constant; a point is far away if it is a factor $c_d > 1$ further away than the $minSize$ nearest point
c_p	Fixed analysis constant; we perform $c_p(\log N)$ partitions of the point set
c_s	A small constant close to zero, used for technical reasons.
f_d, f_g	Factors used for the upper bound on the neighborhood size, see Equation (1).

Table 1: Notation and constants used in the paper

If an event occurs whp for a point (or edge) it occurs for all whp. This can be proved using Boole's inequality (or, alternatively, consider [23]).

Theorem 2 *For n^{c_2} (dependent) events E_i with $i \in [0, n^{c_2}-1]$ and constant c_2 such that each event E_i occurs with probability $p(E_i) \geq 1-1/n^{c_3}$ for $c_3 > c_3+2$, the probability that all events occur is at least $1 - 1/n^{c_3-c_2-2}$.*

4 Pre-process: Data Partitioning

Our density-based clustering algorithm consists of two phases: the first partitions the data so that close points are placed in the same partition. The second uses these partitions to compute distances or densities only within pairs of the same partition. This enables much faster execution.

The partitioning phase splits the dataset into smaller sets (`Partition` algorithm). We perform multiple of these partitions by using different random

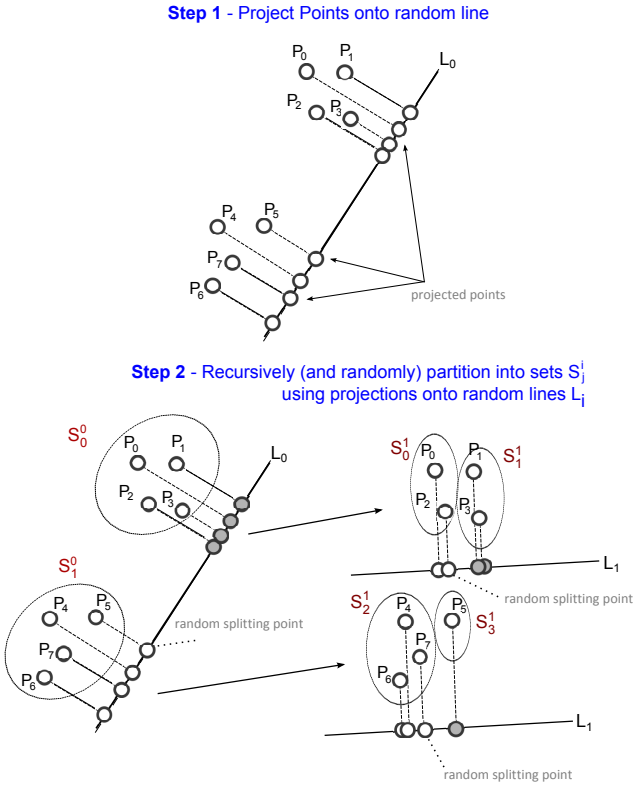

Fig. 1: A single partitioning of points using random projections. The splitting point is chosen uniformly at random in-between the two most distant points on the projection line.

projections (`MultiPartition` algorithm). Intuitively, if the projections $P \cdot L$ and $Q \cdot L$ of two points P, Q onto line L are of similar value then the points should be close. Thus, they are likely kept together whenever the points are divided. The process is illustrated in Figure 1.

For a single partition, we start with the entire point set. We split it recursively into two parts until the size of the point set is at most $minSize+1$, where $minSize$ is a parameter of the algorithm. To split the points, the points are projected onto a random line, and a point that has been projected onto the line is chosen uniformly at random. All points with a projected value smaller than that of the point chosen constitute one part and the remainder the other part. In principle, one could also split based on distance, i.e., pick a point randomly on the projection line that lies between the projected point of minimum and maximum value. However, this might create sets that only contain points of one cluster. This yields infinite distances between clusters, because no distance will be computed for points stemming from different clusters. For example, if there are three very dense clusters on one line, then using

Algorithm 1 Algorithm Partition(points $\mathcal{S}$, line j, sequence of random lines $\mathcal{L}$,$minSize$) return set of sets $\mathfrak{S}$

if $|S| > minSize$ **then**
  $r_s :=$ value chosen uniformly at random from $\{Q \cdot L_j | Q \in \mathcal{S}, L_j \in \mathcal{L}\}$
  $\mathcal{S}_0 := \{Q \in \mathcal{S} | Q \cdot L_j \leq r_s, L_j \in \mathcal{L}\}$
  $\mathcal{S}_1 := \mathcal{S} \setminus \mathcal{S}_0$
  $Partition(\mathcal{S}_0, j+1, \mathcal{L}, minSize)$
  $Partition(\mathcal{S}_1, j+1, \mathcal{L}, minSize)$
else
  $\mathfrak{S} := \mathfrak{S} \cup \{\mathcal{S}\}$
end if

Algorithm 2 MultiPartition(points $\mathcal{P}$, minimum set size $minSize$) return set of point sets $\mathfrak{S}$

for $i = 1..c_p(\log N)$ **do**
  Choose sequence of random lines $\mathcal{L} := (L_0, L_1, ..., L_{c_L \log N})$ for constant c_L with $L_j \in \mathbb{R}^d$ being a random vector of unit length
  $\mathfrak{W} :=$ result of $Partition(\mathcal{P}, 0, \mathcal{L}, minSize)$
  $\mathfrak{S} := \mathfrak{S} \cup \mathfrak{W}$
end for

a distance-based splitting criterion will give the following: The first random projection will likely yield one set containing all points of one cluster and one set containing all points of the other two clusters. It is probable that these two clusters being in one set are split into two separate sets in the second projection. From then on, all further partitionings are within the same cluster. Thus, all clusters are assumed to have infinite distances from each other, although all clusters are on the same line and might have rather different distances to each other. Using our splitting criterion for this scenario yields that (most likely) some pair of points from different clusters will be considered.

More formally, the `MultiPartition` algorithm chooses in total $c_p \log N$ sequences (for constant c_p), where each sequence $\mathcal{L} := (L_0, L_1, ...)$ consists of $c_L \log N$ random lines for a constant c_L with $L_j \in \mathbb{R}^d$. The Partition algorithm is called for each sequence $\mathcal{L}$. The points $\mathcal{S}$ are projected onto each line $L_j \in \mathcal{L}$. First, after the projection onto L_0, the points $\mathcal{S}$ are split into two disjoint sets $\mathcal{S}_0^0 \subseteq P$ and $\mathcal{S}_1^0$ using the value $r_s := L_0 \cdot A$ of a randomly chosen point $A \in \mathcal{S}$. The set $\mathcal{S}_0^0$ contains all points $P \in \mathcal{P}$ with smaller projected value than the number r_s chosen, i.e., $Q \cdot L_0 \leq r_s$, and the other points $\mathcal{P} \setminus \mathcal{S}_0^0$ end up in $\mathcal{S}_1^0$. Afterwards, recurse on sets $\mathcal{S}_0^0$ and $\mathcal{S}_1^0$, that is, for line L_1 we first consider set $\mathcal{S}_0^0$ and split it into sets $\mathcal{S}_0^1$ and $\mathcal{S}_1^1$. Then, a similar process is used on $\mathcal{S}_1^0$ to obtain sets $\mathcal{S}_2^1$ and $\mathcal{S}_3^1$. For line L_2, we consider all four sets $\mathcal{S}_0^1$, $\mathcal{S}_1^1$, $\mathcal{S}_2^1$ and $\mathcal{S}_3^1$. The recursion ends once a set $\mathcal{S}$ contains fewer than $minSize+1$ points. We compute the union of all sets of points resulting from any partitioning for any of the projection sets $\mathcal{L} \in \mathfrak{L}$. Techniques equivalent to algorithm `Partition` have been used in the RP-tree [7].

Theorem 3 *For a d-dimensional dataset, algorithm Partition runs in* $O(dN \log N)$ *time whp.*

Essentially, the theorem says that we need to compute $O(\log N)$ projections of all points, because each projection has to project N points of d dimensions taking dN time. If we were to split a set of N points into two sets of equal size $N/2$ then it is clear that $\log N$ projections would be sufficient, because after that many splits the resulting sets are only of size 1, i.e., $N/2/2/2... = N/2^{\log N} = 1$. Therefore, the proof deals mainly with showing that this also holds when splitting points are chosen randomly.

Proof For each random line $L_j \in \mathcal{L}$, all N points from the d-dimensional space are projected onto the random line L_j, which takes time $O(dN)$. The number of random lines required until a point P is in a set of size smaller than *minSize* is bounded as follows: In each recursion, the given set $\mathcal{S}$ is split into two sets $\mathcal{S}_0, \mathcal{S}_1$. By $p(E_{|\mathcal{S}|/4})$ we denote the probability of event $E_{|\mathcal{S}|/4} := \min(|\mathcal{S}_0|, |\mathcal{S}_1|) \geq |\mathcal{S}|/4$ that the size of both sets is at least $1/4$ than that of the total set. As the splitting point is chosen uniformly at random, we have $p(E_{|\mathcal{S}|/4}) = 1/2$. Put differently, the probability that a point P is in a set of size at most $3/4$ of the overall size $|\mathcal{S}|$ is at least $1/2$ for each random line L. When projecting onto $|\mathcal{L}| = c_L \cdot \log N$ lines, we expect $E_{|\mathcal{S}|/4}$ to occur $c_L \cdot \log N/2$ times. Using Theorem 1, the probability that there are fewer than $c_L \cdot \log N/4$ occurrences is

$$e^{-c_L \cdot \log N/48} = 1/N^{c_L/48}.$$

For a suitable constant c_L, we have

$$N \cdot (3/4)^{c_L \cdot \log N/4} < 1.$$

Therefore, the number of recursions until point P is in a set $\mathcal{S}$ of size less than *minSize* is at most $c_L \cdot \log N$ whp. Using Theorem 2 this holds for all N points whp. A single projection takes time $O(dN)$. Thus, the time to compute $|\mathcal{L}| = c_L \cdot \log N$ projections is $O(dN \log N)$ whp. □

Algorithm `MultiPartition` calls Algorithm `Partition` $c_p(\log N)$ times, thus using Theorem 2 we get the following corollary.

Corollary 1 *Algorithm MultiPartition runs in* $O(dN \log^{2+2\log(c_d)^3/c_d} N)$ *time whp.*

5 Neighborhood

Using the data partitioning described above, we compute, for each point, a neighborhood consisting of nearby points and an estimate of density. Each set resulting from the data partitioning consists of nearby points. Thus, potentially, all points in a set are neighbors of each other. However, looking at a set

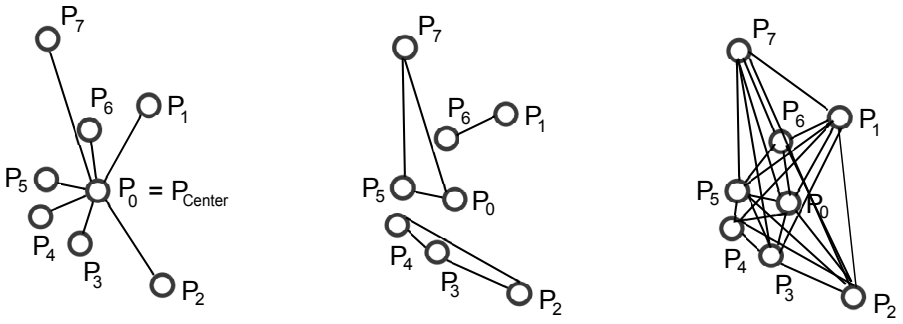

Fig. 2: Picking a center and adding all points to its neighborhood (left panel) results in a connected graph. Picking the same number of edges at random (center panel) likely results in several non-connected components. Picking all pairwise distances (right panel) is computationally expensive.

as a clique of points results in an excessive computation and memory overhead, because the distances for all pairs of neighboring points must be computed.

OPTICS (see Section 7.1) uses the idea of core points. If a core point has sufficiently large density then the core point and all its closest neighbors NC form a cluster, irrespective of the neighborhood of the points NC near the core point. This motivates the idea to pick only a single point per set, call it *center*, and add the other points of the set to the neighborhood of the center (and the center to the neighborhood of all points). If the center is dense enough, it and its neighbors are in the same cluster. Another motivation to pick a single point and add all points to its neighborhood is that this gives a connected component with the minimum number of edges (see Figure 2). More precisely, the single point picked from a set S of points has $|S|$-1 edges. Picking $|S|$-1 edges randomly reduces the probability that the graph is connected, e.g., picking edges randomly results in the creation of triangles of nearby nodes.

To reduce run-time, one may consider to evaluate all pairwise distances only for a single random projection and (potentially) perform fewer random projections overall. Although this seems feasible, further reducing the number of projections to asymptotically below $\log n$ (i.e., $o(\log n)$) poses a high risk of obtaining inaccurate results, because a single random projection only preserves distances in expectation and, therefore, a minimum number of projections is necessary to obtain stable and accurate neighborhoods. More precisely, using only a few random projections likely creates neighborhoods that consist of points that are actually far from each other, i.e., that should not be considered neighbors, and points that are not in the same neighborhood although they are close to each other.

To summarize the neighborhood creation process: A sequence $\mathcal{S} \in \mathfrak{S}$ is an ordering of points projected onto a random line (see Figure 1). For each sequence $\mathcal{S} \in \mathfrak{S}$, we pick a random point, i.e., a center point P_{Center}. For this point, we add all other points $\mathcal{S} \setminus P_{Center}$ to its neighborhood $\mathcal{N}(P_{Center})$. The center

P_{Center} is added to the neighborhood $\mathcal{N}(P)$ of all points $P \in \mathcal{S} \setminus P_{Center}$.The pseudocode is given in Algorithm 3.

Algorithm 3 Neighbors(set of set of points $\mathfrak{S}$) return for each point A neighbor set $\mathcal{N}(A)$

```
1: for all P ∈ S ∈ 𝔖 do N(P) := {} end
2: for all S ∈ 𝔖 do
3:    P_Center := random point in set S
4:    N(P_Center) := N(P_Center) ∪ (S \ P_Center)
5:    for all P ∈ S \ P_Center do
6:       N(P) := N(P) ∪ {P_Center}
7:    end for
8: end for
```

The next theorem elaborates on the size of the neighborhood created. In the current algorithm, we only ensure that the size of the neighborhood is at least $\Omega(\min(minSize, (\log N)))$. Thus, for a large parameter $minSize$, i.e., $minSize \gg \log N$, the size of the neighborhood might be smaller than $minSize$. In this situation, the neighborhood would be a sample of size roughly $\log N$ of close points. To get a larger neighborhood, it is possible to pick more than one center per set in Algorithm 3.

Theorem 4 *For the size $|\mathcal{N}(A)|$ of a neighborhood $\mathcal{N}(A)$ for every point A holds $|\mathcal{N}(A)| \in \Omega(\min(minSize, (\log N)))$ whp and $|\mathcal{N}(A)| \in O(\log N \cdot minSize)$.*

Proof The size of the neighborhood of a point A can be bounded by keeping in mind that the entire point set is split $c_p(\log N)$ times into sets of size at most $minSize$. For each final set of size at most $minSize$ a point may receive $minSize - 1$ new neighbors. This yields the upper bound. A point A gets at least one neighbor for the first set. From then on, for every final set that is the result of the partitioning process, a new point might either be added to the neighborhood or a point chosen might already be in the neighborhood. Algorithm MultiPartition performs $c_p(\log N)$ calls to algorithm Partition. For each call, we obtain a smallest set $\mathcal{S}_A$ containing A. Define $\mathfrak{S}_A \subset \mathfrak{S}$ to be the union of all sets $A \in \mathcal{S}_A \in \mathfrak{S}$ containing A. Before the last split of a set $\mathcal{S}_A$ resulting in the sets $\mathcal{S}_{1,A}$ and $\mathcal{S}_2$, the set $\mathcal{S}$ must be of size at least $c_m \cdot minSize$; the probability that splitting it at a random point results in a set $\mathcal{S}_A$ with $|\mathcal{S}_A| < c_m/2 \cdot minSize$ is at most 1/2. Thus, using a Chernoff bound 1, at least $c_p/8 \log N$ sets $\mathcal{S}_A \in \mathfrak{S}_A$ are of size at least $c_m/2 \cdot minSize$ whp. Assume that the current number of distinct neighbors $\mathcal{N}(A)$ for A is smaller than $\min(c_m/4 \cdot minSize, c_p/16(\log N))$. Then, for each of the $c_p/8(\log N)$ final sets $\mathcal{S}_A$ the probability that a new point is added to $\mathcal{N}(A)$ is at least 1/2. (Note that we cannot guarantee that final sets resulting from the partitioning are different.) Thus, we expect that at least $\min(c_m/4 \cdot minSize, c_p/16(\log N))$ points are added (given $|\mathcal{N}(A)| < c_m/4 \cdot minSize$). The probability that we

deviate by more than $1/2$ of the expectation is $1/N^{c_p/96}$ using Theorem 1 for point A and $1/N^{c_p/96-2}$ for all points using Theorem 2. Therefore, for every neighborhood, it is at least $\min(c_m/8 \cdot minSize, c_p/32(\log N))$ whp. □

The time complexity is dominated by the time it takes to compute the partitioning of points (see Theorem 1). Once we have the partitioning, the time is linear in the number of points per set: For example, hash tables requiring O(1) for inserts and finds (to check whether a neighbor is already stored) can be used to implement the neighborhood construction in lines 4 and 6 in Algorithm 3.

Corollary 2 *The neighborhood of all points $A \in \mathcal{P}$ can be computed in time $O(dN \log^{2+1/c_d} N)$.*

The neighborhood may not contain some close points, but rather some more distant points as shown in Figure 3. We discuss the details and guarantees in Section 7.3.

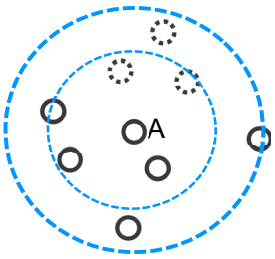

Fig. 3: Point A and its 8 closest points. Potentially, the computed neighborhood $\mathcal{N}(A)$ of a point A using 5 points might miss some of the closest points, e.g., the points with dashed circles.

6 Density Estimate

To compute the density at point A one needs to measure the volume containing a fixed amount of points. This volume is roughly r^d, where d is the dimension and r is the radius of a ball that is required to include a fixed number of points $minPts$. More precisely, the radius r is the distance to the $minPts$-th point. The density is then a function of $1/r^d$.

Many of the existing density-based clustering algorithms, such as OPTICS, use $1/r$ as a measure for density rather than $1/r^d$. Reasons for this are:

- It is computationally faster.
- For large d, even small changes in r would yield large differences in density. Therefore, a transformation would be required when visualizing densities of points.

A key question is how the number of points $minPts$ used to compute the volume relates to the minimum size threshold $minSize$ such that a set is split in Algorithm `MultiPartition`. Because a point is chosen uniformly at random as center to compute the neighborhood from a final set, the final set should be of order $|\mathcal{N}(A)|$. Thus, the minimum split size should also be about $2\mathcal{N}(A)|$, because in expectation a split reduces the size by a factor of 2, i.e., $minSize \approx 2 \cdot |\mathcal{N}(A)| \approx minPts$.

Algorithm 4 states the density and neighborhood computation. For the theoretical analysis of Algorithm 4 (given later), we fix $minSize$ to $c_m \cdot minPts$ for some constant c_m determined in the analysis.

6.1 Smoother Density Estimate

Let us illustrate the downside of using $1/r$ as a density estimate. The computation of r is indifferent to the distribution of the $minPts$ closest datapoints within the ball of radius r from a point A, as well as to that of points that are further away than the $minPts$ point from A. Also, the distribution of points in datasets is often not very homogeneous – otherwise the outcome of clustering would not be meaningful. Furthermore, the value of parameter $minPts$ is typically small, e.g., choosing $minPts$ between 10 to 100 is a reasonable choice. As a consequence, using the radius r to the $minPts$-closest point might yield unnatural density estimates, as shown in Figure 4 because of the high sensitivity on the number of fixed points $minPts$. For all three cases points A, B have circles of equal radius (and, thus, equal density) in Figure 4. For the distribution of points on the left, this seems plausible. In the middle distribution, A should be of larger density, because all points except one are very near to A, thus changing $minPts$ by one has a large impact. On the right-hand side, A also appears denser because it has many points that are just at marginally larger distance than the $minPts$ closest point.

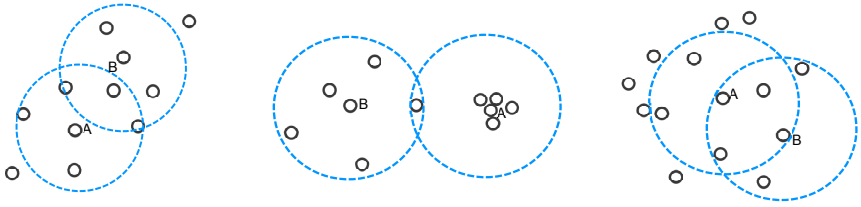

Fig. 4: Three examples of a distribution of 10 points together with the circles for two points A, B covering the 5 closest points. For a standard definition of density, all densities for A and B are the same although an intuitive consideration might not suggest this.

This shortcoming motivates the computation of an average involving several points, e.g., the radius r can be computed using $(1-f)\cdot minPts$ to $(1+f)\cdot minPts$ closest points for constant $f \in [0,1]$, yielding a less sensitive density estimate. We define the density of a point and the average distance $Davg(A)$ of a point A, which depends on the neighbors $\mathcal{N}(A)$.

Definition 1 The set of neighbors $\mathcal{N}_f(A)$ is a subset of neighbors $\mathcal{N}(A)$ given by all points with distances between the $(1-f)minPts$-closest point C_0 and the $(1+f)minPts$-closest point C_1 in $\mathcal{N}(A)$ for constant $f \in [0,1]$.

$$\mathcal{N}_f(A) := \{B \in \mathcal{N}(A) | D(A,C_0) \leq D(A,B) \leq D(A,C_1)\}$$

Note, for f close to 1, we use A as C_0, i.e., the 0th nearest neighbor of A is A itself.

Definition 2 The average distance $Davg(A)$ of a point A is the average of the distances from A to each point $P \in \mathcal{N}_f(A)$:

$$Davg(A) := \sum_{B \in \mathcal{N}_f(A)} D(A,B)/|\mathcal{N}_f(A)|$$

Definition 3 The density at a point A is the inverse of the average distance, i.e., $1/Davg(A)$.

This definition yields significantly better results for many cases, in which $1/r$ is not appropriate, as shown in Figure 4. Algorithm 4 states the density and neighborhood computation for $f = 1$.

Algorithm 4 DensityEstimateAndNeighbors(points $\mathcal{P}$, distance in points $minPts$) return for each point A its neighborhood $\mathcal{N}(A)$ and density estimate $\tilde{D}_{minPts}(A)$ (or average distance $Davg(A)$)

1: $f := 1$; $minSize := c_m \cdot minPts$; {with constant c_m for theoretical analysis; For implementation $f := 0.2$; $minSize := minPts$;}
2: $\mathfrak{S} := MultiPartition(\mathcal{P}, minSize)$
3: $\mathcal{N} := Neighbors(\mathfrak{S})$
4: **for all** $A \in \mathcal{P}$ **do**
5: $\tilde{D}_{minPts}(A)$:= Distance to $minPts$-closest point in $\mathcal{N}(A)$ {OPTICS density}
6: Alternatively, a smoother density estimate $Davg(A)$ instead of $\tilde{D}_{minPts}(A)$ can be used:
7: —— $C_0 := (1-f)minPts$-closest point in $\mathcal{N}(A)$
8: —— $C_1 := (1+f)minPts$-closest point in $\mathcal{N}(A)$
9: —— $\mathcal{N}_f(A) := \{B \in \mathcal{N}(A) | D(A,C_0) \leq D(A,B) \leq D(A,C_1)\}$
10: —— $Davg(A) := \sum_{B \in \mathcal{N}_f(A)} D(A,B)/|\mathcal{N}_f(A)|$
11: **end for**

7 Density-Based Clustering using Reachability

We apply our ideas to speed up the computation of an ordering of points, i.e., OPTICS [3].

7.1 OPTICS

Ordering points to identify the clustering structure (OPTICS) [3] defines a sequence of all points and a distance for each point. This enables an easy visualization to identify clusters. Similarity between two points A, B is measured by computing a reachability distance. This distance is the maximum of the Euclidean distance between A and B and the core distance (or density around a point), i.e., the distance of A to the $minPts$-th points, where $minPts$ corresponds to the minimum size of a cluster. Thus, any point A is equally close (or equally dense) to B if A is among the $minPts$-nearest neighbors of B. If this is not the case then the distance between the two points matters. The algorithm comes with a parameter ϵ that impacts performance. Parameter ϵ states the maximum distance for which we look for the $minPts$-closest neighbors. Using ϵ equal to the maximum distance of a point therefore requires the computation of all pair-wise distances without the use of sophisticated data structures. Choosing ϵ very small may not cluster any points, as the neighborhood of any point is empty, i.e., all points have zero density. A core point is a point that contains at least $minPts$ points within distance ϵ.

The algorithm maintains a list of point pairs sorted by their reachability distance. It chooses a point A with minimum reachability distance (if the list is non-empty, otherwise it chooses an arbitrary point and uses "undefined" as reachability distance. It marks the point as processed and updates the list of point-wise distances by computing the reachability distance from A to each neighbor. It updates or inserts pairs of points (with their corresponding distance) consisting of A and the neighbors of A if a pair of points has not already been processed or if the newly computed distance is smaller.

7.2 *S-OPTICS*: Speedy OPTICS

Our algorithm for density-based clustering, SOPTICS, introduces a fast version of OPTICS which exploits the pre-processing elaborated previously to discover the neighborhood of each point.[1] The processing of points is the same as for OPTICS, aside from the neighborhood computation as shown in Algorithm 5 (line 4). A key difference is that we do not need a parameter ϵ as in OPTICS. Optionally, one might also use the smoothed density estimate introduced in Section 6.1 as discussed in the next subsection.

Algorithm 5 provides a pseudocode for SOPTICS. Essentially, we maintain an updatable heap, in which each entry consists of a reachability distance of a point A and the point A itself. The heap is sorted by reachability distance. For initialization an arbitrary point is put on the heap with undefined reachability distance. Afterwards, repeatedly a point (with shortest reachability distance) is polled from the heap and marked as processed before the reachability distance of all its non-processed neighbors is computed and either inserted into the heap or an existing entry for that point is updated.

[1] SOPTICS presents small differences to the Fast OPTICS (FOPTICS) presented in [21].

Algorithm 5 SOPTICS(points $\mathcal{P}$, distance in points $minPts$) return $ordereminPts$ of points and reachability distance $reachdist$ for points

$\forall P \in \mathcal{P} : processed(P) := false, reachDist(P) := \infty$
$Heap := ()$ {Updatable heap of pairs (reachability distance, point) sorted by reach. dist.}
$ordereminPts := ()$
$\tilde{D}_{minPts}, \mathcal{N} := DensityEstimateAndNeighbors(\mathcal{P}, minPts)$ {or for smoother density $Davg$}
while $\exists P \in \mathcal{P} : processed(P) == false$ **do**
  Heap.Add((undefined,P))
  while Heap not empty **do**
    $P_{curr} := Heap.Poll$
    $ordereminPts.Append(P_{curr})$
    $processed(P_{curr}) := true$
    for all $A \in \mathcal{N}(P_{curr})$ with $processed(A) == false$ **do**
      $Dreach := \max(\tilde{D}_{minPts}(A), D(A, P_{curr}))$ {or for smoother density $\max(Davg(A), D(A, P_{curr}))$}
      $reachdist(A) := \min(reachdist(A), Dreach)$
      Heap.AddOrUpdate(($reachdist(A)$,A))
    end for
  end while
end while

We have discussed the neighborhood computation in Section 5. Thus, let us now discuss in more detail how we deal with parameter ϵ. OPTICS requires to set a parameter ϵ that balances performance and accuracy. A small parameter results in the core distance being undefined for many points. Therefore, the clustering result would not be very meaningful. For OPTICS, a realistic value for ϵ is the maximum distance to the $minPts$-nearest neighbor. Such an approach represents a good compromise between performance and accuracy. However, this can result in drastic performance penalties in the case of uneven point densities. Let's see it with an example: Assume that there is a dense area of points of diameter 10, for which $\epsilon = 1$ would suffice for optimal accuracy and a significantly less dense area, which requires $\epsilon = 10$. Choosing $\epsilon = 10$ means that OPTICS computes all pairwise distances of points within the dense cluster, whereas for $\epsilon = 1$ it might compute only a small fraction of all pairwise distances. Therefore, it would be even better to define ϵ depending on a point A, i.e., $\epsilon(A)$ is the distance to the $minPts$-nearest neighbor. Using our random projection-based approach, we do not define ϵ directly, but we set $minPts$, determining the size of a set of points that is used for the computation of the core distance. Intuitively, for each point, we would like to know its $minPts$-closest neighbor. Assuming a set computed by our random projections indeed contains nearest neighbors, we have $minSize \approx minPts$ (see discussion after definition 3). Specifying the number of points per set presents a more intuitive approach than using a fixed distance for all points, because it can be set to a fixed (i.e., optimal) value for all points to yield maximal performance, while maintaining the best possible accuracy.

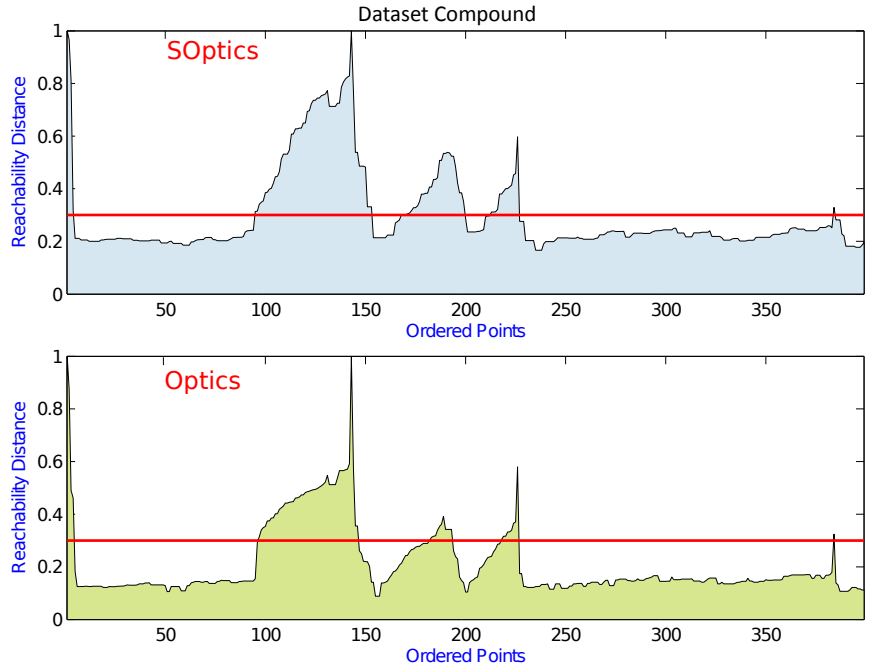

Fig. 5: Reachability plots of SOPTICS and OPTICS for the `Compound` dataset. Note that both exhibit the same hills and valleys and hence discover the same clusters.

Theorem 5 *Algorithm SOPTICS runs in* $O(dN \log^2 N)$ *time whp. It requires* $O(N(d + \log N \cdot minSize))$ *memory.*

Proof Computing all neighbors requires $O(dN \log^2 N)$ time whp according to Theorem 2. The average number of neighbors is at most $\log N$ per point. The size of the heap is at most N. For each point, we consider each of the most $\log N$ neighbors at most once. Thus, we perform $O(N \log N)$ heap operations and also compute the same number of distances. This takes time $O(N \log N \cdot (d + \log N))$. Storing all d-dimensional points in memory requires $N \cdot d$. Storing the neighborhoods of all points requires $\mathrm{O}(N \log N \cdot minSize)$. □

In Figure, 5 we provide a visual illustration of the reachability plots computed on one dataset for OPTICS and SOPTICS. It is apparent that both techniques reveal the same cluster structure.

7.2.1 Smoother density estimate

When using the smoother density estimate from Section 6.1, we get different notions for reachability and core distance.

Definition 4 The core distance of a point A equals the average distance $Davg(A)$.

For the core distance in the original OPTICS algorithm only the distance to a single point, i.e., the $minPts$-nearest point, matters. Using our probabilistic approach, we compute a smoother estimate: points that are closer (or further) than the $minPts$-nearest point also matter.

Two points are reachable if they are neighbors, i.e., one of the two points must be in the neighborhood of the other.

The definition of the reachability distance for a point A and a reachable point B from A is the same as for OPTICS. However, for a point A, we only compute the reachability distance to all neighbors $B \in \mathcal{N}(A)$.

Definition 5 The reachability distance $Dreach(A, B)$ is the maximum of the core distance of A and the distance of A and B, i.e., $Dreach(A, B) := \max(Davg(A), D(A, B))$.

Note that the reachability distance is non-symmetric, i.e., in general $Dreach(A, B) \neq Dreach(B, A)$.

7.3 Theoretical Analysis

Now we state our main theorems regarding the complexity of the techniques presented. We also show that our suggested smoother density measure (Section 6.1) approximates the core distance used in OPTICS. Our algorithm strongly relies on the well-known Johnson–Lindenstrauss Lemma, which states that, if two points are projected on a randomly chosen line, the distance of the projected points on the line corresponds to the scaled distance of the non-projected points, in expectation. Higher-dimensional spaces can in general not be embedded in one dimension without distortion, so the above only holds in expectation. The scaling factor is the same for all points: $1/\sqrt{d}$, i.e., one over the squared root of the original data dimensionality.

We state and prove theorems that show that we retrieve at least some close neighbors for every point, but not necessarily all nearest neighbors. We state a bound for the $minPts$-nearest neighbor distance. This allows us to relate the core distances of OPTICS and SOPTICS. It also helps to get a bound for our smoothened density estimate $Davg(A)$ for a point in $\mathbb{R}^d$, so we relate $Davg(A)$ and the average of the distance to the $minPts$-nearest neighbors of a point A.

Specifically, in Theorem 7, we prove that only a small fraction of points have a projected length that is much longer or shorter than its expectation. This enables us to bound the probability that a projection and a splitting-up of points will keep close points together and separate distant points as shown in Theorem 8. Therefore, after a sequence of partitionings, we can pick a point randomly for each set containing A to include in the neighborhood $\mathcal{N}(A)$ such that at least some of the points $\mathcal{N}(A)$ are close to A (Theorem 9) given that there are some more points near A than just $minPts$. The latter condition stems from the fact that we split a set by the number of points in

the set and not by distance. We discuss it in more detail before the theorem. Theorem 10 explicitly gives a bound on the distance to the $minPts$-nearest neighbor. Finally, in Theorem 11, we relate our computed density measure and the one of OPTICS by showing that they differ at most by a constant factor.

We require a "mild" upper bound on the neighborhood size of a point. The reason being that for a given point, points distant from it are very likely removed compared with very near points when splitting a set. But points that are somewhat near are not removed much more likely than very close points. Thus, if there are too many of them, we need many splits to remove them all and the odds that we remove also a lot of nearby points becomes large.[2]
More mathematically, we require an upper bound on the number of points that are within a certain distance of A. In fact, for point A the bound depends on the distance to the nearest neighbor distances, i.e., r. Roughly speaking, the number of points that are within some factor $f_g \cdot r$ of the distance r cannot be more than exponential in f_g. More precisely, with all details being clarified during this section, consider the distance $(f_g)^{3/2+c_s} \cdot (f_d \cdot r)$ for an arbitrary integer f_g, a value $f_d > 1$ and a small constant $c_s > 0$. The number of points within that interval for $f_g \geq 1$ is allowed to be at most $2^{\sqrt{f_g}} \cdot |\mathcal{N}(A, f_d \cdot r)|$ points. More formally, we require for point A

$$|\{B \in \mathcal{S} | D(A,B) \leq (f_g)^{3/2+c_s} \cdot f_d \cdot r\}| \leq 2^{\sqrt{f_g}} \cdot |\mathcal{N}(A, f_d \cdot r)| \qquad (1)$$

Let $\mathcal{S}_A$ be a set of points containing point A, i.e., $A \in \mathcal{S}_A$, being projected onto a random line L. We distinguish between three point sets:

i) Points close to A, i.e., within radius r, i.e, $\mathcal{S}_A \cap \mathcal{N}(A, r)$, where $\mathcal{N}(A, r)$ are the points within radius r from point A.
ii) Points distant from A, i.e, $\mathcal{S}_A \setminus \mathcal{N}(A, c_d r)$, for some constant $c_d > 1$ (defined later).
iii) Points $\mathcal{N}(A, c_d r)$ that consist of close and somewhat close points.

For these three sets, we prove in Theorems 6 and 7 that only for a few close points will the distance of their projections onto a random line be much larger than the expectation, quantified by random variable X_A^{long}, and that for only few distant points will their projections be much smaller than the expected projection, quantified by X_A^{short}.

Let event $E_A^{long}(C)$ be the event that for a randomly chosen line L the projected length $(C - A) \cdot L$ of a close point $C \in \mathcal{N}(A, r)$ is more than a factor $\log(c_d)$ of the expected projected length $\mathbb{E}[(C - A) \cdot L]$. Let X_A^{long} be the random variable giving the number of all occurred events $E_A^{long}(C)$ for all points $C \in \mathcal{N}(A, r)$.

Let event $E_A^{short}(C)$ be the event that for a randomly chosen line L the projected length $(C-A) \cdot L$ of a distant point $C \in \mathcal{S}_A \setminus \mathcal{N}(A, c_d r)$ is less than a

[2] This condition could be removed for low-dimensional spaces, i.e., assuming d is constant.

factor $2\log(c_d)/c_d$ of the expected projected length $\mathbb{E}[(C-A)\cdot L]$. Let X_A^{short} be the random variable giving the number of all occurred events $E_A^{short}(C)$ for all points $C \in \mathcal{S}_A \setminus \mathcal{N}(A, c_d r)$.

Theorem 6 *For every point C holds $p(E_A^{short}(C)) \leq 3\log(c_d)/c_d$ and $p(E_A^{long}(C)) \leq 2/\log(c_d) e^{-\log(c_d)^2/2}$ for some value c_d.*

Proof The probability of event $E_A^{short}(C)$ can be bounded using Lemma 5 [7]: $p(E_A^{short}(C)) \leq 3\log(c_d)/c_d$. Using again Lemma 5 from [7] for $E_A^{long}(C)$ we have $p(E_A^{long}(C)) \leq 2/\log(c_d) e^{-\log(c_d)^2/2}$. □

Theorem 7 states that for most points there is not too much deviation from the expectation. More precisely, for most close points (as well as for most distant points) the distances of projected close (as well as distant) points is not much longer (shorter) than the expectation.

Theorem 7 *For points $\mathcal{S}_A$ projected onto a random line L define event $E' := (X_A^{short} < |\mathcal{S}_A \setminus \mathcal{N}(A, c_d r)|/\log(c_d)) \wedge (X_A^{long} < |\mathcal{S}_A \cap \mathcal{N}(A, r)|/(c_d)^{\log(c_d)/3})$ We have*

$$p(E') \geq (1 - 2\log(c_d)^2/c_d)^2.$$

Proof Can be found in the appendix. □

The proof works in the same fashion for X_A^{short} and X_A^{long}. We discuss the main ideas using X_A^{short}. The proof computes a bound on the expectation of X_A^{short} by using linearity of expectation to express the expectation of X_A^{short} in terms of the expectation of individual events that are upper-bounded using Theorem 6. To bound the probability that X_A^{short} does not exceed the upper bound of the expectation, we use Markov's inequality.

The next theorem shows that a set resulting from the partitioning is likely to contain some nearby points. The proof starts by looking at a single random projection and assumes that there are only relatively few non-distant points left in the set containing A. It shows that it is likely that distant points from A are removed whenever a set is split, whereas it is unlikely that points near A are removed. Therefore, for a sequence of random projections, we can prove that some nearby points will remain and many more distant points are removed. On the technical side, the proof uses elementary probability theory.

Theorem 8 *For each point A, for at least $c_p/16(\log N)$ sets $\mathcal{S}_A$ resulting from a call to algorithm Partition*

$$|\mathcal{S}_A \cap \mathcal{N}(A, r)|/|\mathcal{N}(A, r)| > 2/c_d$$

Proof Can be found in the appendix. □

The next theorem shows that the computed neighborhood for a point A contains at least some points "near" A. It contains a restriction on the parameter *minPts* that is mainly due to neighborhood construction but could

be eliminated by using a larger parameter $minSize$. The proof bounds the number of sets resulting from the partitioning that are at least of a certain size using a Chernoff bound. Then we compute the probability that for a point A a new nearby point is chosen as a neighbor.

Theorem 9 *The neighborhood $\mathcal{N}_A$ computed by Algorithm 3 for a point A contains at least $2 \cdot minPts$ points for $minPts < c_m \log N$ that are within distance $D_{c_m \cdot minPts}(A)$ from A.*

Proof Can be found in the appendix. □

Let us bound the approximation of the $minPts$-nearest-neighbor distance when using the neighborhood computed by Algorithm 3.

Theorem 10 *Let $D_{minPts}(A)$ be the distance to the $minPts$-nearest neighbor. For the distance $\tilde{D}_{minPts}(A)$ to the $minPts$-nearest neighbor in the neighborhood $\mathcal{N}(A)$ computed by Algorithm 3 holds $D_{minPts}(A) \leq \tilde{D}_{minPts}(A) \leq D_{c_m minPts}(A)$ for suitable constant c_m whp.*

Proof The lower bound follows from bounding the minimum size of $\mathcal{N}(A, r)$. Using Theorem 9, $2minPts$ points are contained within distance $D_{c_m minPts}(A)$. The smallest value of $\tilde{D}_{minPts}(A)$ is reached when $\mathcal{N}(A)$ contains all $minPts$-closest points to A, which implies $\tilde{D}_{minPts}(A) = D_{minPts}(A)$. For the upper bound due to Theorem 9, $\mathcal{N}_A$ contains at least $2minPts$ within distance $D_{c_m \cdot minPts}(A)$. Thus, the $minPts$-nearest point in $\mathcal{N}_A$ is at most at distance $D_{c_m \cdot minPts}(A)$. □

Assume that the k-nearest-neighbor distance is not decreasing very rapidly, when increasing the number of points considered, i.e., k. More formally, assume that there exists a sufficiently large constant $c > 1$, such that $D_{c \cdot minPts}(A) \leq c \cdot D_{minPts}(A)$. Then, we compute a constant approximation of the nearest-neighbor distance.

Next, we relate the core distance of OPTICS (see Section 7.1), i.e., the distance to the $minPts$-nearest neighbors of a point A, and of SOPTICS, i.e., $Davg(A)$.

Theorem 11 *For every point $A \in \mathbb{R}^d$, $D_{minPts}(A) \leq Davg(A) \leq D_{c_m minPts}(A))$ holds for constant c_m and $f = 1$ whp.*

Proof To compute $Davg(A)$ with $f = 1$, we consider the $(1 + f) \cdot minPts = 2minPts$ closest points to A from $\mathcal{N}(A)$. Using Theorem 9, $2minPts$ points are contained in $\mathcal{N}(A)$ with distance at most $D_{c_m minPts}(A)$. This yields $Davg(A) \leq D_{c_m minPts}(A)$. Thus, the upper bound follows. To compute $Davg(A)$, we average the distance using the $2 \cdot minPts$-closest points to A. The smallest value of $Davg(A)$ is reached when $\mathcal{N}(A)$ contains all $2 \cdot minPts$-closest points to A, which implies $Davg(A) \geq D_{2 \cdot minPts}(A) \geq D_{minPts}(A)$ for any set of neighbors $\mathcal{N}(A)$. □

Assume that the average distance to the $minPts$-nearest neighbor is an equivalently valid density measure compared with the distance of the $minPts$-th neighbor used by OPTICS. Typically, the cluster size is significantly larger than $minPts$ and the density within clusters is not varying very rapidly when looking at a point and some nearest neighbors. In this case, we compute an $O(1)$ approximation of the density, i.e., core distance, used by OPTICS. This is fulfilled if the distances to the $minPts^{th}$ up to the $(c_m minPts)^{th}$ point do not increase by more than a constant factor compared with the $minPts$-closest point. More technically, we require the existence of a (sufficiently large) constant c such that $\forall A \in \mathcal{P} : D_{minPts \cdot c}(A) = c \cdot D_{minPts}(A)$.

8 Empirical Evaluation

Here we evaluate the runtime and clustering quality of the proposed random-projection-based technique. The SOPTICS algorithm has been implemented in Java[3]. We compare its performance with that of OPTICS with and without LSH index [8] and DeLi-Clu, from the Elki Java Framework [24] [4] using version 0.7.0 (2015, November 27). DeLi-Clu represents an improvement of OPTICS that leverages indexing structures (e.g., R*-trees) to improve performance. OPTICS with an LSH index it also a good baseline comparison, because it allows one to support very fast nearest-neighbor queries. All experiments have been conducted on a 2.5GHz Intel[5] CPU with 8GB RAM.

8.1 Datasets

We use a variety of two-dimensional datasets typically used for evaluating density-based algorithms as well as high-dimensional data sets to compare the performance of the algorithms for increasing data dimensionality. A summary of the datasets is given in Table 2. We did not apply any particular preprocessing to the datasets; for all algorithms, we measured the time, once the data set had been read into memory.

Implementation details: The source code of SOPTICS is available at the second author's website[6], and can also be found in the ELKI Framework, as of version 0.7.0 [24][7]. Algorithm `Multipartition` chooses $O(\log^{2+1/c_d} N)$ random projections lines and performs the same number of projections of the entire point sets onto each of these lines. In practice, it suffices to choose a

[3] Java is a registered trademark of Oracle and/or its affiliates.

[4] `elki.dbs.ifi.lmu.de/`

[5] Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries. Other product or service names may be trademarks or service marks of other companies

[6] `http://alumni.cs.ucr.edu/~mvlachos/erc/projects/density-based/`

[7] The latest optimizations are not included in version 0.7.0

Dataset	Objects	Dim	Clusters	Adjusted Rand Index	Time (sec)			
				SOPTICS vs OPTICS	SOPTICS (no index)	OPTICS	DeLiClu	OPTICS (with LSH)
Magic04[4]	19020	10	2	0.981	4.3	43.2	27	30.1
Gas Sens. Arr.[4]	13910	128	6	0.979	4.4	192.3	75.4	25.3
EEG Eye State[4]	14980	15	2	0.971	2.8	43.3	26.3	87.4
Musk [4]	6598	168	2	0.981	4.9	82.3	72.4	60
Svmguide1 [18]	3089	4	2	0.944	0.5	2.6	2.1	1.5
Aggregation[11]	788	2	7	1	0.3	0.3	0.3	0.3
Diabetes [18]	768	8	5	0.994	0.2	0.2	0.2	0.2
R15[26]	600	2	15	0.991	0.1	0.1	0.1	0.1
Jain[15]	373	2	2	0.935	0.0	0.0	0.1	0.1
Iris[4]	150	4	3	0.983	0.0	0.0	0.0	0.0

Table 2: Characteristics of the datasets used in the experiments (first four columns). The Adjusted Rand Index shows how similar the OPTICS and SOPTICS clustering are. Finally, the last four columns show the runtime for SOPTICS, OPTICS without index, DeLi-Clu and OPTICS with LSH. SOPTICS closely preserves the clustering of OPTICS while exhibiting a very fast runtime.

set $\mathcal{L}$ of $O(\log N)$ random lines, project all points onto this lines and permute them for each call of algorithm `Partition`. Thus, the practical runtime of algorithm `Multipartition` is $O(dN \log N)$ and that of SOPTICS $O(N \log N \cdot (d + \log^{1+1/c_d} N))$.

Parameter setting: OPTICS requires the parameters ϵ and $minPts$. When not using an index, ϵ is set to infinity, which provides the most accurate results; $minPts$ depends on the dataset. When using an LSH index, setting the parameters is non trivial. For parameter ϵ of OPTICS, we used the smallest ϵ that is needed to get an accurate result, i.e., the maximum distance to the $minPts$-nearest neighbor of a point of a dataset. The LSH index requires three main parameters: number of projections per hash value (k), number of hash tables (l) and the width of the projection (r). For parameters k and l, we performed a grid search using values between 10 and 40. We kept both parameters at a value of 20 because it returned the best results. The width r should be related to the distance of the maximum $minPts$-nearest neighbor, i.e., ideally a bin contains the $minPts$-nearest neighbor of a point and it should also depend on the dimension, because distances are scaled by the square root of the dimension. Thus, in principle, the maximal distance of any point to the $minPts$-nearest neighbor should roughly suffice to get the same results as for OPTICS with $\epsilon = \infty$ (up to some constant factor, i.e., values below 8 did not yield good results; for the "Musk" benchmark we used 64). DeLi-Clu requires only the $minPts$ parameter. SOPTICS uses the same parameter value $minPts$ as OPTICS (and DeLi-Clu) and we set $minSize = minPts$. We set $f = 0.2$. We performed $20 \log(Nd)$ partitionings, i.e., calls to algorithm `Partition` from `MultiPartition` of the entire dataset for SOPTICS.

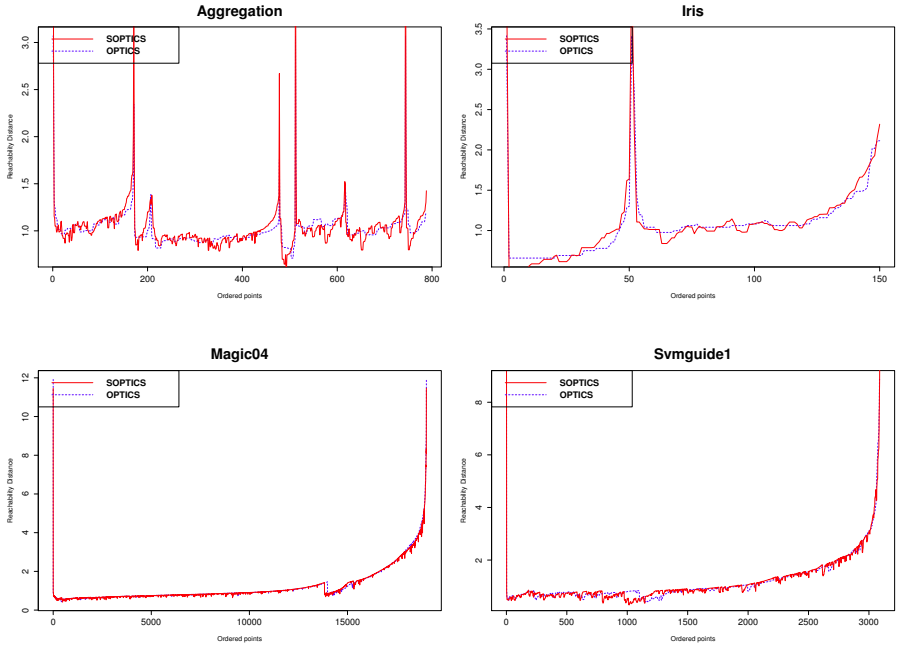

Fig. 6: Reachability plots for various datasets (note that clusters might be permuted)

8.2 Cluster Quality

The original motivation of our work was to provide faster versions of existing density-based techniques while not compromising their accuracy. To compare the cluster results, we use the adjusted Rand index [14]. It returns a value between 0 and 1, where 1 indicates identical clustering. The adjusted Rand index corrects for the chance grouping of elements. The ordering of points as computed by OPTICS does not yield a clustering. Therefore, we defined a threshold that gives a horizontal line in the ordering plot. Whenever the threshold is exceeded, a new cluster begins. We chose the threshold for OPTICS and SOPTICS to match the actual clusters as well as possible and compared the clusters found by OPTICS and SOPTICS. The results are shown in Table 2. Note that the metric consistently exceeds the 0.95 value, suggesting that SOPTICS provides clustering results that are indeed very close to those of OPTICS. More importantly, SOPTICS delivers these results significantly faster than both OPTICS and DeLi-Clu, as also shown in Table 2.

Figure 6 provides visual examples of the high similarity of the reachability plots for SOPTICS and OPTICS. The reachability plot of SOPTICS is less smooth. At first sight, one might expect the opposite as by definition the

average distance $Davg$ is a smoother density estimate. The reason is that the computation of the $minPts$-nearest neighbor is not perfectly accurate. Thus, for some points, our approximation might be accurate, for others the computed neighborhood might consist of points not being part of the $minPts$-nearest neighborhood. In our computation of the $Davg$, we used all points in the set resulting from the partitioning of points. Thus, for a point A, $Davg$ is not necessarily computed using its $minPts$-nearest neighbors, but potentially might miss some nearest neighbors and incorporate some points further away. The random partitioning induces a larger variance in $Davg$. In principle, one could filter outliers to reduce the variance, e.g., for a set S resulting from a partitioning, one could discard points that are far from the mean of all points in S. However, as the reachability plot and the extracted clusters matched very well for OPTICS and SOPTICS, we refrained from additionally filtering any outliers.

8.3 Runtime

Table 2 already shows the clear performance advantages of SOPTICS. Not surprisingly, OPTICS without an index is much slower. However, even using the LSH index does generally not yield satisfactory results. Our splitting of the entire point set is based on the number of points within a region. LSH splits the entire point set according to a fixed bin width, i.e., in a distance-based manner. This distance must inevitably be chosen rather large (e.g., close to maximum) among all points to get accurate results. Therefore, bins are generally (much) too large and contain many points, resulting in slow performance. DeLiClu is generally significantly faster than OPTICS, but still much slower than SOPTICS.
In addition to the experiments discussed above, we conduct scalability experiments using synthetically generated datasets according to a Gaussian distribution. Each Gaussian cluster consists of 1000 points. We use more than 120,000 objects and a dimensionality of 10 to evaluate scalability with respect to the number of objects. In a similar manner we generate synthetic datasets having up to 1200 dimensions to assess scalability with regard to dimensionality. The performance comparison between the various density-based techniques is shown in Figure 7. It suggests a drastic improvement of SOPTICS compared with OPTICS and DeLi-Clu. SOPTICS is more than 500 times faster than OPTICS and more than 20 times faster than DeLi-Clu. Note that DeLi-Clu uses an R*-tree structure to speed up various operations. Our approach bases its runtime improvements on random projections, thus is simpler to implement and maintain.

Figure 8 highlights the runtime for increasing data dimensionalities. Note that the performance gap between OPTICS and DeLi-Clu *diminishes* for higher dimensions. In fact, for more than 500 dimensions, OPTICS is faster than DeLi-Clu. This is due to the use of indexing techniques by DeLi-Clu. It is well understood that the performance of space-partitioning indexing struc-

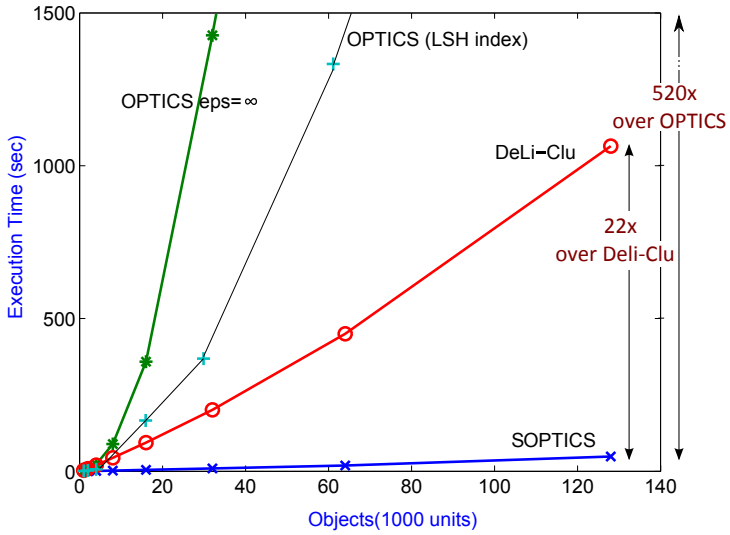

Fig. 7: Runtime comparison of SOPTICS with OPTICS and DeLi-Clu, for increasing number of data objects.

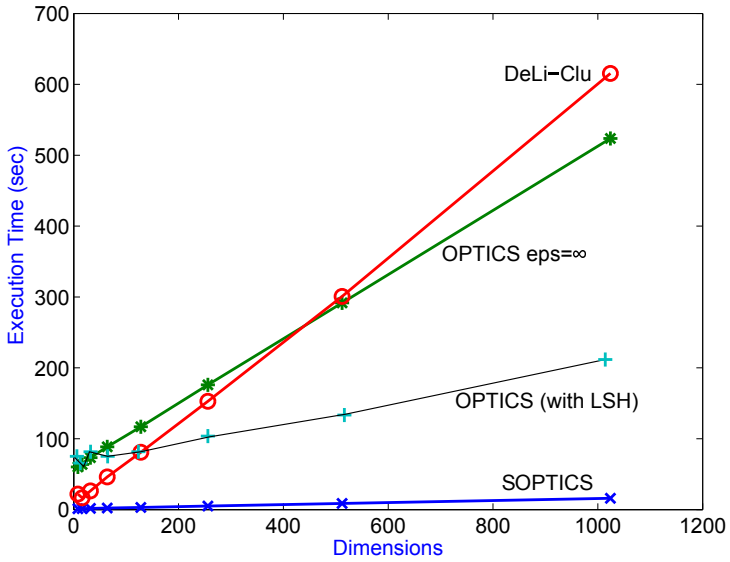

Fig. 8: Evaluating the approaches for increasing data dimensionality. The performance of DeLi-Clu diminishes for higher dimensions, because of its use of indexing techniques.

tures, like R-trees, diminishes for increasing dimensionalities. The performance improvements of SOPTICS compared with OPTICS range from 47 times (at low dimensions) to 32 times (for high dimensions). A different trend is sug-

gested in the runtime improvement against DeLi-Clu, which ranges from 17 times (at low dimensions) to 38 times (at high dimensions). Using OPTICS with an LSH index improves performance at higher dimensionalities, but the approach is still much slower than SOPTICS. Therefore, when dealing with high-dimensional datasets, it is preferable to use techniques based on random projections.

9 Conclusion

Density-based techniques can provide the building blocks for efficient clustering algorithms. Our work contributes to density-based clustering by presenting SOPTICS, which is a random-projection-based version of the popular OPTICS algorithm. Not only is it orders of magnitude faster than OPTICS, but it also comes with analytical clustering preservation guarantees. In the spirit of reproducibility, we have also made available the source code of our approach.

Acknowledgements: The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 259569.

References

1. E. Achtert, C. Böhm, and P. Kröger. DeLi-Clu: Boosting Robustness, Completeness, Usability, and Efficiency of Hierarchical Clustering by a Closest Pair Ranking. In *Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD)*, pages 119–128, 2006.
2. G. Andrade, G. Ramos, D. Madeira, R. Sachetto, R. Ferreira, and L. Rocha. G-DBSCAN: A GPU Accelerated Algorithm for Density-based Clustering, 2013.
3. M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proc. ACM Intl Conf. on Management of Data (SIGMOD)*, pages 49–60, 1999.
4. A. Asuncion and D. Newman. UCI machine learning repository. `http://archive.ics.uci.edu/ml/datasets.html`, 2007.
5. C. Böhm, R. Noll, C. Plant, and B. Wackersreuther. Density-based clustering using graphics processors. In *Proc. Intl. Conf. Information and Knowledge Management (CIKM)*, pages 661–670, 2009.
6. R. Chitta and M. N. Murty. Two-level k-means clustering algorithm for k-tau relationship establishment and linear-time classification. *Pattern Recognition*, 43(3):796–804, 2010.
7. S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proc. Symposium on Theory of Computing (STOC)*, pages 537–546, 2008.
8. M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. Annual Symposium on Computational Geometry*, pages 253–262, 2004.
9. T. de Vries, S. Chawla, and M. E. Houle. Density-preserving projections for large-scale local anomaly detection. *Knowledge and Information Systems*, 32(1):25–52, 2012.
10. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. ACM Conf. Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.
11. A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *ACM Trans. Knowledge Discovery from Data*, 1(1), 2007.

12. A. Hinneburg and H.-H. Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. In *Advances in Intelligent Data Analysis (IDA)*, pages 70–80, 2007.
13. A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proc. ACM Conf. Knowledge Discovery and Data Mining (KDD)*, pages 58–65, 1998.
14. L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.
15. A. K. Jain and M. H. C. Law. Data clustering: A user's dilemma. In *Proc. Pattern Recognition and Machine Intelligence*, pages 1–10, 2005.
16. W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz maps into a Hilbert space. In *Contemporary Mathematics 26*, pages 189–206, 1984.
17. M. Koyutürk, A. Grama, and N. Ramakrishnan. Compression, clustering, and pattern discovery in very high-dimensional discrete-attribute data sets. *IEEE Trans. Knowl. Data Eng. (TKDE)*, 17(4):447–461, 2005.
18. C.-J. Lin. LibSVM datasets. `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`, 2011.
19. W.-K. Loh and H. Yu. Fast density-based clustering through dataset partition using graphics processing units. *Information Sciences*, 308(0):94 – 112, 2015.
20. J. Schneider, J. Bogojeska, and M. Vlachos. Solving Linear SVMs with Multiple 1D Projections. In *Proc. Intl. Conf. Information and Knowledge Management (CIKM)*, pages 221–230, 2014.
21. J. Schneider and M. Vlachos. Fast parameterless density-based clustering via random projections. In *Proc. Intl. Conf. Information and Knowledge Management (CIKM)*, pages 861–866, 2013.
22. J. Schneider and M. Vlachos. On randomly projected hierarchical clustering with guarantees. In *Proc. SIAM Int. Conf. Data Mining (SDM)*, pages 407–415, 2014.
23. J. Schneider and R. Wattenhofer. Distributed Coloring Depending on the Chromatic Number or the Neighborhood Growth. In *Intl. Colloquium Structural Information and Communication Complexity (SIROCCO)*, pages 246–257, 2011.
24. E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid, and A. Zimek. A framework for clustering uncertain data. *PVLDB*, 8(12):1976–1987, 2015.
25. T. Urruty, C. Djeraba, and D. A. Simovici. Clustering by random projections. In *Industrial Conference on Data Mining*, pages 107–119, 2007.
26. C. J. Veenman, M. J. T. Reinders, and E. Backer. A maximum variance cluster algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(9):1273–1280, 2002.
27. J. J. Whang, X. Sui, and I. S. Dhillon. Scalable and memory-efficient clustering of large-scale social networks. In *Proc. IEEE Conf. Data Mining (ICDM)*, pages 705–714, 2012.
28. Y. Yu, J. Zhao, X. Wang, Q. Wang, and Y. Zhang. Cludoop: An efficient distributed density-based clustering for big data using hadoop. *International Journal of Distributed Sensor Networks*, pages 2–2, Jan. 2015.

10 Appendix

Proof of Theorem 7: By definition and using linearity of expectation, the expectation of X_A^{short} is $\mathbb{E}[X_A^{short}] := \sum_{C \in \mathcal{S}_A \setminus \mathcal{N}(A, c_d r)} p(E_A^{short}(C))$. Using Theorem 6 to bound $p(E_A^{short}(C))$,

$$\mathbb{E}[X_A^{short}] \leq 3 \log(c_d)/c_d |\mathcal{S}_A \setminus \mathcal{N}(A, c_d r)|$$

The probability that the random variable X_A^{short} exceeds the expectation $\mathbb{E}[X_A^{short}]$ by a factor $c_d/\log(c_d)^2$ or more is at most $\log(c_d)^2/c_d$ using Markov's inequality. Thus, for the probability of event E_0 as defined below we have

$$p(E_0) := p(X_A^{short} < \mathbb{E}[X_A^{short}] \cdot c_d \leq 3 \log(c_d) |\mathcal{S}_A \setminus \mathcal{N}(A, c_d r)|) \geq 1 - \log(c_d)^2/c_d.$$

Analogously, let us bound the probability of event $E_A^{long}(C)$ that a projection of two

points C, A results in a distance $L \cdot (C - A)$ much beyond the expectation. Next, we use Theorem 6 to bound $p(E_A^{long}(C))$. By definition the expectation of X_A^{long} is $\mathbb{E}[X_A^{long}] := \sum_{C \in \mathcal{S}_A \cap \mathcal{N}(A,r)} p(E_A^{long}(C))$. Consider the upper bound of $\mathbb{E}[X_A^{long}]$ being $\mathbb{E}[X_A^{long}] \cdot c_d$, i.e., $c_d/(c_d)^{\log(c_d)/2}|\mathcal{S}_A \cap \mathcal{N}(A,r)| \geq 1/(c_d)^{\log(c_d)/3}|\mathcal{S}_A \cap \mathcal{N}(A,r)|$ (for c_d sufficiently large). Thus, define the probability of event E_1 and bound as before using Markov's inequality as follows:

$$p(E_1) := p(X_A^{long} \leq |\mathcal{S}_A \cap \mathcal{N}(A,r)|/(c_d)^{\log(c_d)/3}) \geq 1 - 1/c_d.$$

Assume E_0 occurs. This excludes at most a fraction $\log(c_d)^2/c_d$ of all possible projections for event E_1, leaving

$$(1 - 1/c_d - \log(c_d)^2/c_d) > 1 - 2\log(c_d)^2/c_d.$$

Thus, the probability of E_1 given E_0 becomes $p(E_1|E_0) = 1 - 2\log(c_d)^2/c_d$. The probability of event $E_3 := E_0 \cap E_1$ is

$$\begin{aligned} p(E_1|E_0) \cdot p(E_0) \geq \quad & (1 - 2\log(c_d)^2/c_d) \cdot (1 - \log(c_d)^2/c_d) \\ \geq \quad & (1 - 2\log(c_d)^2/c_d)^2. \end{aligned}$$

□

Proof of Theorem 8: The idea of the proof is to look at a point A and remove "very" far away points until there are only relatively few of them left. Then, we consider somewhat closer points (but still quite far away) and remove them until we are left with only some very close points and some potentially further away points. Consider a partitioning of set $\mathcal{S}_A$ into two sets $\mathcal{S}_0$ and $\mathcal{S}_{1,A}$, i.e., $A \in \mathcal{S}_{1,A}$ using algorithm Partition and random projection line L. Assume that the following condition holds for set $\mathcal{S}_A$: There are many more points "very far away" from A than not so distant points using some factor $f_d \geq c_d$:

$$c_r|\mathcal{S}_A \cap \mathcal{N}(A, f_d \cdot r)| \leq |\mathcal{S}_A \setminus \mathcal{N}(A, f_d \cdot r)| \tag{2}$$

The value c_r is defined later; we require $c_r \geq f_d \geq 1$. We prove that even in this case after a sequence of splittings of the point set only few very far away points end up in set $\mathcal{S}_{1,A}$. (If there are fewer faraway points than somewhat close points, the probability that many of them end up in the same set is even smaller.) Define event E_1 as follows: A splitting point is picked such that for the subset $\mathcal{S}_{1,A}$ most very close points from $\mathcal{N}(A,r) \cap \mathcal{S}_A$ remain, i.e.,

$$|\mathcal{S}_{1,A} \cap \mathcal{N}(A,r)| \geq |\mathcal{S}_A \cap \mathcal{N}(A,r)| \cdot (1 - 1/c_r).$$

The probability of event E_1 can be bounded as follows. Assume that E' as defined in Theorem 7 occurs (using $f_d > c_d$ instead of c_d), i.e., most distances are scaled roughly by the same factor from a point A to other points. To minimize the probability of $p(E_1|E')$ we assume that all projected distances from faraway points to A are minimized and those of close points are maximized. This means that at most a fraction $1/\log f_d$ of all very far away points $\mathcal{S}_A \setminus \mathcal{N}(A, f_d \cdot r)$ are below a factor $3\log(f_d)/f_d$ of their expected length and that the distances to all other points in $\mathcal{S}_A \setminus \mathcal{N}(A, f_d \cdot r)$ are shortened exactly by that factor. We assume the worst possible scenario, i.e., those far away points are split such that they end up in the same set as A, i.e., they become part of $S_{1,A}$. At most a fraction $1/(f_d)^{\log(f_d)/3}$ of all very close points $\mathcal{S}_A \cap \mathcal{N}(A,r)$ are above a factor $\log(f_d)$ of the expectation. We assume that those points behave in the worst possible manner, i.e., the close points exceeding the expectation are split such that they end up in a different set than A, i.e., S_0 not $S_{1,A}$. Next, we bound the probability that no other points from $\mathcal{S}_A \cap \mathcal{N}(A,r)$ are split. If we pick a splitting point among the fraction of $1 - 1/\log f_d$ points from $\mathcal{S}_A \setminus \mathcal{N}(A, f_d r)$ that are not shortened by more than a factor $3\log(f_d)/f_d$, then $p(E_1|E')$ occurs. By initial assumption we have $(1 - 1/f_d^{\log(f_d)/3})|\mathcal{S}_A \cap \mathcal{N}(A, f_d \cdot r)| \leq (1 - 1/\log f_d) \cdot c_r|\mathcal{S}_A \setminus \mathcal{N}(A, f_d r)|$ and thus, $|\mathcal{S}_A \setminus \mathcal{N}(A, f_d r)|/|\mathcal{S}_A| \leq 2/c_r$ for $1 - 1/\log f_d > 1/2$, i.e., f_d sufficiently large, and because $|\mathcal{S}_A| \geq |\mathcal{S}_A \setminus \mathcal{N}(A, f_d r)|$. Put differently, the probability to pick a bad splitting point is at most $2/c_r$. The occurrence of event E' reduces the probability of E_1 at most by $1 - p(E')$, i.e., $p(E_1|E') \geq p(E_1) - (1 - p(E'))$.

Therefore,

$$\begin{aligned} p(E_1) = & \quad p(E')p(E_1|E') \\ = & \quad p(E') \cdot (1 - |\mathcal{S}_A \setminus \mathcal{N}(A, f_d \cdot r)|/|\mathcal{S}_A| - (1 - p(E'))) \\ \geq & \quad p(E') \cdot (1 - 2/c_r - 2\log(f_d)^2/f_d)) \\ \geq & \quad (1 - 2\log(f_d)^2/f_d) \cdot (1 - 4\log(f_d)^2/\min(f_d, c_r))^3 \\ = & \quad (1 - 4\log(f_d)^2/f_d)^3 \text{ since by definition } c_r \geq f_d \end{aligned}$$

Define event E_2 as follows: At least $1/3 - 1/c_r$ of all far away points $|\mathcal{S}_A \setminus \mathcal{N}(A, f_d r)|$ are not contained in $\mathcal{S}_{1,A}$, i.e.,

$$|\mathcal{S}_A \setminus \mathcal{N}(A, f_d r)| \geq 2/3|\mathcal{S}_{1,A} \setminus \mathcal{N}(A, f_d r)|.$$

The probability that the size of the set resulting from the split $\mathcal{S}_{1,A}$ is at most 2/3 of the original set $\mathcal{S}_A$ is 1/3, because a splitting point is chosen uniformly at random. When restricting our choice to far away points $\mathcal{S}_A \setminus \mathcal{N}(A, f_d r)$, we can use that owing to Condition (2) at most a fraction $1/c_r$ of all points are not far away. The probability of E_2 given E_1 can be bounded by assuming that all events, i.e., choices of random lines and splitting points, that are excluded owing to the occurrence of E_1 actually would have caused E_2. More precisely, we can subtract the probability of the complementary event of E_1, i.e., $p(E_2|E_1) = 2/3 - 1/c_r - (1 - p(E_1)) \geq 2/3 - 1/c_r - (1 - 2\log(c_d)^2/f_d)^3 \geq 1/4$ for a sufficiently large constant f_d. The initial set $\mathcal{S} := \mathcal{P}$ has to be split at most $c_L \log N$ times until the final set $\mathcal{S}_A$ containing A (which is not split any further) is computed (see proof of Theorem 3). We denote a trial T as up to $\log f_d$ splits of a set $\mathcal{S}$ into two sets. A trial T is successful if after at most $\log f_d$ splits of a set $\mathcal{S}_A$ the final set $\mathcal{S}'_A \subset \mathcal{S}_A$ is of size at most $|\mathcal{S}_A|/2$ and E_1 occurred for every split. The probability for a successful trial $p(T)$ is equal to the probability that E_1 always occurs and E_2 at least once. This gives:

$$\begin{aligned} p(T) & = p(E_1)^{\log f_d} \cdot (1 - p(E_2|E_1)^{\log f_d}) \\ & \geq (1 - 2\log(f_d)^2/f_d)^{3\log f_d} \cdot (1 - 1/4^{\log f_d}) \\ & \geq (1 - 2\log(f_d)^2/f_d)^{4\log f_d} \end{aligned} \tag{3}$$

Starting from the entire point set we need $\log(N/minSize) + 1$ (consecutive) successful trials until a point A is in a set of size less than $minSize$ and the splitting stops. Next we prove that the probability to have that many successful trials is constant given that the required upper bound on the neighborhood holds, i.e., (1). Assume there are n_i points within distance $[i^{3/2+c_s} \cdot c_d \cdot r, (i+1)^{3/2+c_s} \cdot c_d \cdot r]$ for a positive integer i. In particular, note that the statement holds for arbitrarily positioned points. We do not even require them to be fixed across several trials.

The upper bound on the neighborhood growth (1) yields that $n_i \leq 2^{i^{1/2}} \cdot |\mathcal{N}(A, c_d r)|$. Furthermore, we have that $\sum_{i=1}^{\infty} n_i \leq N$. Next, we analyze how many trials we need to remove points n_i until only the close points $\mathcal{N}(A, c_d r)$ remain. We are going from large i to small i, i.e., remove distant points first. For each n_i we need at most $\log n_i - \log |\mathcal{N}(A, c_d r)| \leq i^{1/2}$ successes. Let E_{n_i} be the event that this happens, i.e., that we have that many consecutive successes.

$$
\begin{aligned}
p(E_{n_i}) &:= \prod_{j=1}^{\log n_i - \log|\mathcal{N}(A,c_d r)|} p(T) \\
&= \prod_{j=1}^{\log n_i - \log|\mathcal{N}(A,c_d r)|} (1 - 4\log(x)^2/(x))^{2\log x} \qquad \text{(Defining } x := i^{3/2+c_s} \cdot c_d\text{)} \\
&= \prod_{j=1}^{\sqrt{i}} (1 - 1/2^{\log(x) - 4\log\log x})^{4\log(x)} \\
&= 2^{4\sqrt{i}\log(x)\log(1-1/2^{\log(x)-4\log\log(x)})} \\
&\geq 2^{-4\sqrt{i}\cdot\log(x)\cdot\log(x)^4/2^{\log(x}} \\
&= 2^{-\sqrt{i}\log(i^{3/2+c_s}\cdot c_d)^5/(i^{3/2+c_s}\cdot c_d)} \\
&\geq 2^{-1/(i\cdot c_d)} \text{ (for } c_s \text{ and } c_d \text{ sufficiently large)} \qquad (4)
\end{aligned}
$$

As the number of points N is finite, the number of $n_i > 0$ is also finite. Let m_A be the largest value such that $n_{m_A} > 0$. Let $p_A := p(\wedge_{i\in[1,m_A]} E_{n_i})$ be the probability that all trials for all n_i in $i \in [1, m_A]$ and $n_i > 0$ are successful. Note that the events E_{n_i} are not independent for a fixed point set $\mathcal{P}$. However, the bound (4) on $p(E_{n_i})$ holds as long as condition 2 is fulfilled, i.e., for an arbitrary point set. Put differently, the bound (4) holds even for the "worst" distribution of points. Therefore, we have that $p_A := p(\wedge_{i\in[1,m_A]} E_{n_i}) \geq \prod_{i\in[1,m_A]} 2^{-1/(i\cdot c_d)}$ using stochastic domination. Note that our choice of maximizing n_i, i.e., the number of required successful trials for E_{n_i} minimizes the probability of a $p(E_{n_i})$. This is quite intuitive, since it says that we should maximize the number of points closest to A that should not be placed in the same set as A (i.e., they are just a bit too far to yield the claimed approximation guarantee). We also need to be aware of the fact that the distribution for the n_i under the constraint that $\sum_{i=1}^{m_A} n_i \leq N$ should minimize the bound for p_A. It is also apparent from the derivation of (4) that this happens when we maximize n_i; the probability for p_A decreases more if we maximize small i. Essentially, this follows from line 2 in (4) because the number of trials n_T is less than $\sqrt{i}$ and each trial is successful with probability of $(1 - 1/i^{3/2})$ (focusing on dominating terms), yielding an overall success probability of $(1 - 1/i^{3/2})^{n_T}$ for a trial. Thus, $(1 - 1/i^{3/2})^{\sqrt{i}} > (1 - 1/l^{3/2})^{\sqrt{l}}$ for $1 < i < l$. Put differently, choosing n_i large for a large i is not a problem for our algorithm, because it is unlikely that these points will be projected in between the nearest points to A.

Therefore, when maximizing the number of points close to A, we have that $m_A = (\log N)^2$, i.e., all n_i for $i > (\log N)^2$ are 0 because $2^{\sqrt{(\log N)^2}} = n_{(\log N)^2} = N$. Additionally, note that we need at most $c_8 \log N$ trials in total. As each trial slices the number of points by 1/2, we only need to take into the account the subset $X \in [1, m_A]$ for which the number of points doubles, i.e., $n_j = 2 \cdot n_i$, for $n_i = 2^{i^{1/2}}$. This happens whenever $i^{1/2}$ is an integer, i.e., for $i = 1, 4, 9, 16, ...$, we get $n_i = 1, 2, 3, 4, ...$ Thus, we only need to look at $i^2 \in [1, m_A]$

$$
\begin{aligned}
p_A &\geq \prod_{i^2\in[1,m_A]} 2^{-1/(c_d\cdot i)} \\
&\geq \prod_{i^2\in[1,\log^2 N]} 2^{-1/(c_d\cdot i)} \\
&\geq 2^{-1/c_d \sum_{i^2\in[1,\log^2 N]} 1/i} \\
&= 2^{-1/c_d \sum_{i\in[1,\log N]} 1/i^2} \\
&\geq 2^{-2/c_d} \\
&\geq 1/2^{2/c_d}
\end{aligned}
$$

Thus, when doing $c_p(\log N)$ partitionings, we have at least $c_p/16 \log N$ successes for point A whp using Theorem 1 and $c_d \geq 1$. This also holds for all points whp using Theorem 2.

Finally, let us bound the number of nearby points that remain. We need at most $c_L \log N$ (see Theorem 3) projections until a point set will not be split further. Each projection reduces the points $|\mathcal{N}(A,r)|$ at most by factor $1-1/c_r$. We give a bound in two steps, i.e., for $c_r \geq \log^3 N$ and $c_r \in [c_d, \log^3 N]$.

$$\prod_{i=1}^{c_L \log N} (1-1/c_r) \geq (1-1/\log^3 N)^{c_L \log N} \text{ (Assuming } c_r \geq \log^3 N) \\ \geq 1-1/\log N$$

To reduce the number of points by a factor of $\log^3 N$ requires $3 \cdot \log\log N$ trials, each reducing the set by a factor $1/2$. Thus, trial i is conducted using a factor $c_r = \log^3 N/2^i$ of the original points or, equivalently, trial $3 \cdot \log\log N - i$ is conducted with $c_r = 2^i$. Thus, in total the fraction of remaining points in $\mathcal{N}(A,r)$ is

$$(1-1/\log N) \prod_{i=1}^{3\log\log N} (1-1/2^i)^{\log c_d} = (1-1/\log N) \cdot (\prod_{i=1}^{3\log\log N} (1-1/2^i))^{\log c_d} \\ = (1-1/\log N) \cdot (2^{\sum_{i=\log c_d}^{3\log\log N} \log(1-1/2^i)})^{\log c_d} \\ \geq (2^{1-2/c_d})^{\log c_d} \geq 1/(2c_d)$$

□

Proof of Theorem 9: First we bound the number of neighbors. Using Theorem 8 we obtain $c_p/16(\log N)$ sets $\mathcal{S}_A$ containing A. Define $\mathfrak{S}_A$ to be the union of all sets $\mathcal{S}_A \in \mathfrak{S}$ containing A. Before the last split of a set $\mathcal{S}_A$ resulting in the sets $\mathcal{S}_{1,A}$ and $\mathcal{S}_2$, the set $\mathcal{S}_A$ must be of size at least $c_m \cdot minPts$; the probability that splitting it at a random point results in a set $\mathcal{S}_{1,A}$ with $|\mathcal{S}_A| < c_m/2 \cdot minPts$ is at most $1/2$. Thus, using a Chernoff bound (Theorem 1), at least $c_p/128 \log N$ sets $\mathcal{S}_A \in \mathfrak{S}_A$ are of size at least $c_m/2 \cdot minPts$ whp.

Let $\mathcal{S}_A$ be a set $\mathcal{S}_A$ with size at least $c_m/2 \cdot minPts$. Consider the process when the neighborhood $\mathcal{N}(A)$ is built by inspecting one set $\mathcal{S}_A$ after the other. Assume that the number of neighbors $|\mathcal{N}(A)| < c_m/2minPts/(2c_d)$. Thus, the probability of event $p(\text{Choose new close neighbor } B) = p(B \notin \mathcal{N}(A) \wedge B \in \mathcal{N}(A,r))$ that a point $B \in \mathcal{S}_A$ but not already in $\mathcal{N}(A)$ is chosen from $\mathcal{N}(A,r) \cap \mathcal{S}_A$ is at least $c_m/(4c_d)$.

$$p(\text{Choose new close neighbor } B | |\mathcal{N}(A)| < c_m/2 \cdot minPts/(2c_d)) := \\ p(B \notin \mathcal{N}(A) \wedge B \in \mathcal{N}(A,r))) = c_m/(4c_d)$$

As by assumption $minPts < c_m \log N$ and there are at least $c_p/128 \log N$ sets $\mathcal{S}_A$ with $|\mathcal{S}_A| \geq c_m/2 \cdot minPts$ and $c_p \geq c_m \cdot 128$, using the Chernoff bound in Theorem 1 we get that there are at least $c_m/(4c_d) minPts$ points within distance $D_{c_m minPts}(A)$ in $\mathcal{N}(A)$ whp for every point A. Setting $c_m \geq 8c_d$ completes the proof. □