Fast Parameterless Density-Based Clustering via Random Projections

Johannes Schneider
IBM Research - Zurich

Michail Vlachos
IBM Research - Zurich

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. We present two fast density-based clustering algorithms based on random projections. Both algorithms demonstrate one to two orders of magnitude speedup compared to equivalent state-of-art density based techniques, even for modest-size datasets. We give a comprehensive analysis of both our algorithms and show runtime of $O(dN \log^2 N)$, for a d-dimensional dataset. Our first algorithm can be viewed as a fast variant of the OPTICS density-based algorithm, but using a softer definition of density combined with sampling. The second algorithm is parameter-less, and identifies areas separating clusters.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; I.5.3 [**Clustering**]: Algorithms

Keywords

Clustering, Data Mining, Random Projections

1. INTRODUCTION

Data doubles about every two years. This makes the analysis of Big Data a necessity and it also drives the design of more efficient algorithms for data analytics. Clustering is an important operation for knowledge extraction. Its objective is to assign objects into groups such that objects within a group are more similar than objects across different groups. Subsequent inspection of the realized groups can provide important insights, with applications to pattern discovery [13], data summarization/compression [10] as well as data classification [4]. Density-based clustering algorithms have emerged both as high-quality and efficient clustering techniques with solid theoretical foundations on density estimation [8]. They can discover clusters with irregular shapes and require easy to set parameters (e.g., minimum number of points per cluster). They also can help 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-art in density-based clustering techniques by presenting algorithms that significantly improve runtime, while providing analytical guarantees on the preservation of cluster quality. We achieve this 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 first algorithm, requires only the setting of a single parameter, the minimum number of points of a cluster, which is customarily required as input in density-based techniques. Our second algorithm lifts this requirement and presents a *parameterless* density-based clustering algorithm by creating a sequence of clusters for varying density parameters. In general, we make the following **contributions**:

- We depict how to use random projections to improve the performance of existing density-based algorithms, such as OPTICS and its performance-optimized version DeLi-Clu. We introduce a new density estimate based on computing average distances. We also provide guarantees on the preservation of cluster quality and runtime.
- We provide an algorithm that eliminates the need for setting any density related parameter. It allows to identify clusters in datasets of strongly varying densities that could not be identified easily with state-of-the-art density-based algorithms. For d-dimensional data, it runs in $O(dN \log^2 N)$ time.
- Both algorithms are evaluated extensively and yield performance gains of up to two orders of magnitude with a *provable* degree of distortion on the clustering result.

2. PRELIMINARIES

For points and lines we use capital letters, e.g. P, T, Q, L. For a set of points or a sequence of lines we use calligraphic font, e.g. $\mathcal{P}, \mathcal{S}, \mathcal{L}$. For a set of sets or sequences we use Fraktur letters, e.g. $\mathfrak{S}, \mathfrak{L}, \mathfrak{W}$. We are given a set of N points $\mathcal{P}$ in the d-dimensional Euclidean space, i.e., for a point $P \in \mathcal{P}$ holds $P \in \mathbb{R}^d$. We use the term *whp*, i.e., with

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2505515.2505590.

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. Define the distance $D(A, B) := ||B - A||_2$ for two points $A, B \in \mathcal{P}$ to be the $L2$-norm. Assume or norm the shortest distance $\min_{A,B \in \mathcal{P}, A \neq B} D(A, B)$ to be 1. We often use the following Chernoff bound:

THEOREM 2.1. *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 $c_1 \in]0, 1[$ can be bounded by $p(X \leq (1-c_0)\mathbb{E}[X] \vee X \geq (1+c_1)\mathbb{E}[X]) < 2e^{-\mathbb{E}[X] \cdot \min(c_0, c_1)^2/3}$*

If an event occurs whp for a point (or edge) it occurs for all whp. The proof uses a standard union bound.

THEOREM 2.2. *For n^{c_0} (dependent) events E_i with $i \in [0, n^{c_0} - 1]$ and constant c_0 s.t. each event E_i occurs with probability $p(E_i) \geq 1 - 1/n^{c_1}$ for $c_1 > c_0 + 2$, the probability that all events occur is at least $1 - 1/n^{c-c_0-2}$.*

3. PRE-PROCESS: DATA PARTITIONING

Our density based clustering algorithms consist of two phases: the first partitions the data so that nearby points are placed in the same partition. The second phase uses these partitions to compute distances or densities only within pairs of the same set. This allows for much faster execution.

The partitioning phase splits the dataset into smaller sets (Partition algorithm). We perform multiple of these partitions by using different random projections (MultiPartition algorithm). Intuitively, if the projection $P \cdot L$ and $Q \cdot L$ of two points P, Q onto line L is of similar value then the points should be close. Thus, they are likely kept together whenever the point set is divided.

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 on the line is chosen uniformly at random. All points with a projected value smaller than that of the chosen point constitute one part and the remainder the other part.

More formally, MultiPartition algorithm chooses the set $\mathfrak{L} := \{\mathcal{L}^0, \mathcal{L}^1, ...\}$ of $c_0 \log N$ sequences (for a constant c_0), where $\mathcal{L}^i := (L_0, L_1, ...)$ is a sequence of $c_1 \log N$ random lines for a constant c_1 with $L_j \in \mathbb{R}^d$. The Partition algorithm is called for a sequence $\mathcal{L}^i$. The points $\mathcal{S}$ are projected onto each line $L_j \in \mathcal{L}^i$. First, after the projection onto L_0, the points $\mathcal{S}$ are split into two disjoint sets $\mathcal{S}_0^1 \subseteq P$ and $\mathcal{S}_1^1$ using the value $r_s := L \cdot A$ of a randomly chosen point $A \in \mathcal{S}$. The set $\mathcal{S}_0^1$ contains all points $P \in \mathcal{P}$ with smaller projected value than the chosen number r_s, i.e., $Q \cdot L_0 \leq r_s$, and the other points $\mathcal{P} \setminus \mathcal{S}_0^1$ end up in $\mathcal{S}_1^1$. Afterwards, recurse on sets $\mathcal{S}_0^1$ and $\mathcal{S}_1^1$, that is to say, for line L_1 we first consider set $\mathcal{S}_0^1$ and split it into sets $\mathcal{S}_0^2$ and $\mathcal{S}_1^2$. Then, the process is repeated for $\mathcal{S}_1^1$ to obtain sets $\mathcal{S}_2^2$ and $\mathcal{S}_3^2$. For line L_2, we consider all four sets $\mathcal{S}_0^2$, $\mathcal{S}_1^2$, $\mathcal{S}_2^2$ and $\mathcal{S}_3^2$. The recursion ends once a set $\mathcal{S}$ contains fewer than $minSize+1$ points. We compute the union of all sets having at most $minSize$ points of all projection sequences $\mathcal{L}^i \in \mathfrak{L}$. An instance of the above process is illustrated in Figure 1. Similar techniques to algorithm Partition have been used in the RP-tree [5]. However, here we use a much simpler splitting rule.

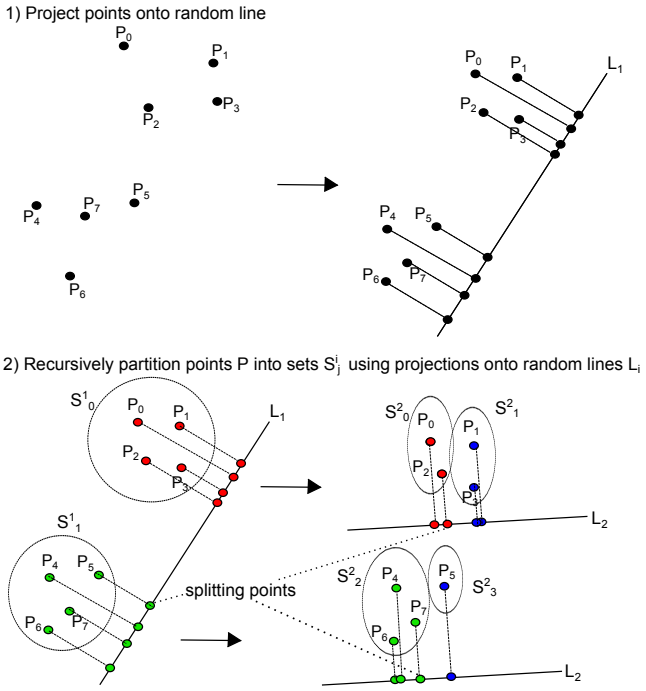

Figure 1: A single partitioning of points using random projections. The splitting points are chosen uniformly at random among all projected points.

Algorithm 1 MultiPartition(points $\mathcal{P}$, minimum set size $minSize$) return set of point sets $\mathfrak{S}$

```
Choose sequences of random lines L^i := (L_0, L_1, ..., L_{c_1 log N})
  for i ∈ [0, c_0 · log N − 1] for constants c_0, c_1 with L_j ∈ R^d being
  a random vector of unit length
for i = 1..c_0 · log N do
    W := result of Partition(P, 0, i, minSize)
    S := S ∪ W
end for

Algorithm Partition(points S, line j, sequence i, minSize)
  return set of sets S
if |S| > minSize then
    r_s := value chosen uniformly at random from {Q · L_j | Q ∈
    S, L_j ∈ L^i}
    S_0 := {Q ∈ S | Q · L_j ≤ r_s, L_j ∈ L^i}
    S_1 := S \ S_0
    Partition(S_0, j + 1, i, minSize)
    Partition(S_1, j + 1, i, minSize)
else
    S := S ∪ {S}
end if
```

The following hold, but we omit the proofs for brevity.

THEOREM 3.1. *For a d-dimensional dataset, Algorithm Partition runs in $O(dN \log N)$ time whp.*

Algorithm MultiPartition calls Algorithm Partition $c_0 \log N$ times, thus using Theorem 2.2:

COROLLARY 3.2. *Algorithm MultiPartition runs in $O(dN \log^2 N)$ time whp.*

4. DENSITY MEASURE AND NEIGHBORHOOD SAMPLE

Using the previous data partitioning we compute for each point a probabilistic neighborhood and an estimate of density.

Sampled Neighbors: For each point A we compute a sample of close neighbors using a set of sequences of points $\mathfrak{S}$ for a parameter $dPts$. A sequence is an ordering of points projected onto a random line (see Figure 1). For each sequence $\mathcal{S} \in \mathfrak{S}$ and every point $A \in \mathcal{S}$ we choose randomly a point B being at most $dPts$ points after point A in the sequence $\mathcal{S}$. All the selected points around A form the sampled neighbors $\mathcal{N}(A)$. See Algorithm 2 and for an example consider Figure 2.

Algorithm 2 SampledNeighbors(set of sequences of points $\mathfrak{S}$, distance in points $dPts$, 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 with |S| ≥ 2 · dPts do
3:    Sort S according to values of projected points
4:    for i = 1 to |S| − dPts do
5:       j := Random integer in [1, dPts]
6:       𝒩(S(i)) := 𝒩(S(i)) ∪ S(i + j)
7:    end for
8: end for
```

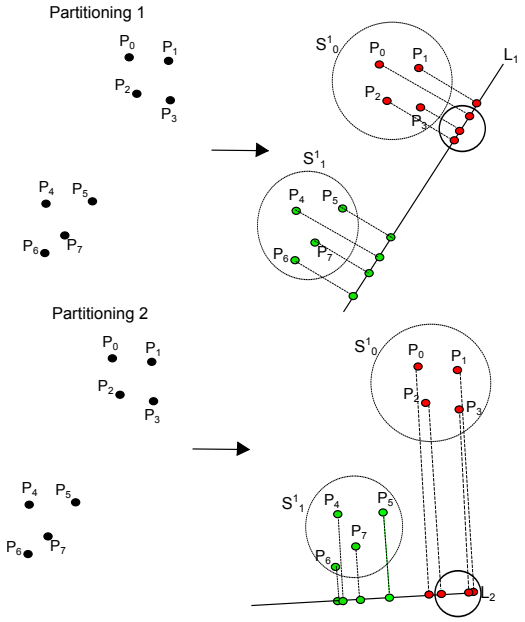

Figure 2: Two partitionings using random projections. For $dPts = 1$ a density estimate for point P_3 is obtained as follows: For the first partitioning either P_2 or P_0 is added to the sampled neighbors $\mathcal{N}(P_3)$. For the second, P_1 or P_2. Say P_0 and P_2 are chosen, i.e. $\mathcal{N}(P_3) = \{P_0, P_2\}$. Then, for $f = 1$ the average distance $Davg(P_3)$ becomes $(D(P_3, P_0) + D(P_3, P_2))/2$. The candidate(s) for P_3 can only be P_2, since $D(P_3, P_0) > m(P_3, P_0) \geq Davg(P_3)$.

Density Estimate: The density of a point A is the average distance $D_{avg}(A)$ of a subset of all sampled neighbors $\mathcal{N}(A)$ for a parameter $dPts$. More precisely, we only consider the points from $N_f(A) \subset \mathcal{N}(A)$ that are among the fraction of f closest points in $\mathcal{N}(A)$ for some constant f. Mathematically speaking, let C be the $|\mathcal{N}(A)| \cdot f$-th closest point in $\mathcal{N}(A)$ to A then $N_f(A) := \{B \in \mathcal{N}(A) | D(A, B) \leq D(A, C)\}$ and $D_{avg}(A) := \sum_{B \in N_f(A)} D(A, B)/|N_f(A)|$. Thus, for $f = 1$, $D_{avg}(A)$ is just the average of all sampled neighbors. Note, the smaller the average distance, the larger the density.

Assume that there are just $dPts+1$ points. Then $D_{avg}(A)$ is an approximation of the average distance to the $f \cdot dPts$-th (closest) neighbors. If we add a point to the dataset that is closer than the $f \cdot dPts$-th nearest neighbor N' then the average distance will decrease (in expectation). If we add a point that is farther away than the $f \cdot dPts$-th nearest neighbor B then the average distance increases. So, if we add many points that are somewhat further away than B then the added points may cause the average distance to increase significantly beyond the average distance of the $f \cdot dPts$-th closest neighbors B. However, as we shall see due to our partition process (Algorithm MultiPartition) points distant from A only appear in a set $\mathcal{S} \in \mathfrak{S}$ with low probability. However, to ensure that the average is not significantly distorted with high probability, we do not compute only the average of all sampled neighbors but restrict ourselves to a subset dependent on the parameter f.

Candidate Mergers: Two points A, B are candidates to be merged if their distance is within merging distance. The merging distance $m(A, B)$ is just the minimum of the average distances of the two points, i.e. $m(A, B) := \min(D_{avg}(A), D_{avg}(B))$. A point A may merge with any sampled neighbor $B \in \mathcal{N}(A)$, if and only if their distance is less than the merging distance $m(A, B) < D(A, B)$. Thus, we restrict the pairs of points that can be merged by any clustering algorithm according to some criterion to the pair of points A, B with $m(A, B) < D(A, B)$ with $B \in \mathcal{N}(A)$ (or $A \in \mathcal{N}(A)$), i.e. we define the candidates for point A as $\mathcal{N}_C(A) := \{B \in \mathcal{N}(A) | m(A, B) < D(A, B)\}$. This process is captured in Algorithm 3. For an example, the reader is directed to Figure 2.

Algorithm 3 CandidateMergers(points $\mathcal{P}$, distance in points $dPts$, return for each point A candidates $\mathcal{N}_C(A)$ for potential mergers)

```
1: 𝔖 := MultiPartition(𝒫, 2 · dPts)
2: 𝔖' := {S | S ∈ 𝔖, |S| < 6 · dPts}
3: Compute SampledNeighbors(𝔖', dPts)
4: for all P ∈ S ∈ 𝔖 do 𝒩_C(P) := {} end
5: for all A ∈ 𝒫 do
6:    for all B ∈ 𝒩(A) do
7:       m(A, B) := min(D_avg(A), D_avg(B))
8:       if m(A, B) < D(A, B) then 𝒩_C(A) := 𝒩_C(A) ∪ B
         end if
9:    end for
10: end for
```

Assume that to estimate the density of a point A we measure a volume $V(A)$ containing $dPts$ points. If the volumes $V(A)$ and $V(B)$ of two points intersect significantly then the density at a point contained in the intersection is likely to be of similar density of either A or B (or both). However, in case the two volumes do not intersect this does not hold. For density-based clustering we want to form clusters of points of similar density - at least all nearby points must have similar density. Therefore, it suffices to consider nearby points, i.e. points with intersecting volumes. Note, that for merging

candidates A and B the volumes $V(A)$ and $V(B)$ used for the computation of the density intersects by definition since $m(A,B) := \min(D_{avg}(A), D_{avg}(B)) < D(A,B)$.

Algorithm Complexity: Now we state our main theorems regarding the complexity of the presented techniques.

THEOREM 4.1. *Algorithm 3 runs in $O(dN \log^2 N)$ whp.*

Next, we state a bound on $D_{avg}(A)$ for a point in $\mathbb{R}^d$, i.e. we relate $D_{avg}(A)$ and the average of the distance to the $dPts$-nearest neighbors of a point A.

Theorem 4.2 states that for the smallest set $\mathcal{S}_A$ returned by the Partition algorithm containing A, a sufficient fraction of all points are not substantially far from A than $D_{avg}(A)$. Therefore, if we sample points from $\mathcal{S}_A$ at least some of these points are close to A (Theorem 4.3).

For a point A, define $D(A, dPts)$ to be the distance to the $dPts$-th nearest point from A. Define $\mathcal{N}(A,r)$ to be all points $C \in S$ within radius $r := D(A, dPts)$ from A, i.e. $D(A,C) < r$. Denote the average distance of the $dPts$ nearest points to a point A as $\overline{D}(A, dPts) := \sum_{B \in \mathcal{N}(A, D(A,dPts))} D(A,B)/|\mathcal{N}(A, D(A,dPts))|$.

THEOREM 4.2. *For a point A, the probability that $|\mathcal{S}_A \setminus \mathcal{N}(A, c_7 r)|/|\mathcal{S}_A| > 1/{c_7}^{1/4}$ is at least $1/2^{-48 \log\log c_7 / c_7}$ for $dPts > c_9$ for some constants c_7, c_9.*

THEOREM 4.3. *For every point $A \in \mathbb{R}^d$ holds $\overline{D}(A, dPts \cdot c_{10}) < D_{avg}(A) < 2\sqrt{c_1} \cdot \overline{D}(A, dPts)$ for a constant c_{10} whp.*

As an example, assume that $\overline{D}(A, dPts \cdot c_{10}) \approx D_{avg}(A) \cdot c_{10}$, which holds in general, if the number of points from A do not increase much faster with distance than to the $dPts \cdot c_{10}$ nearest point. In this case, we compute an $O(1)$-approximation of $\overline{D}(A, dPts)$. Assuming that $\overline{D}(A, dPts)$ is an equivalently valid density measure as the distance to the $minPts$-th neighbor used by OPTICS we compute a $O(1)$-approximation of the density, i.e. core-distance, used by OPTICS.

5. DENSITY BASED CLUSTERING USING REACHABILITY

We apply our ideas to speed up the computation of the popular **OPTICS** algorithm (Ordering points to identify the clustering structure) [2]. OPTICS defines a sequence of all points and a distance for each point. This allows for an easy visualization to identify clusters. Similarity between two points A, B is measured by computing a reachability distance. Figure 3 provides an illustration. This distance is the maximum of the Euclidean distance between A and B and the core-distance (or density of a point), i.e. the distance of A to the $minPts$-th points, where $minPts$ corresponds to the minimum size of a cluster. 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 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 an arbitrary point) and updates the list by computing the reachability distance from A to each neighbor. The algorithm investigates each point's neighbors only once. For performance reasons OPTICS requires a parameter ϵ that denotes an upper bound on the distance between any two points that are considered to compute the reachability distance. The parameter ϵ should be at least the distance to the $minPts$-nearest point of any point.

5.1 Fast OPTICS *FOPTICS*

We compute a similar ordering as for OPTICS but use a different definition of reachability. As core-distance of a point A we use the average distance $D_{avg}(A)$ as proposed in Section 4. Whereas 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 matter, too. Two points are reachable if they are within merging distance. In other words, we dynamically adjust the parameter ϵ for each point A.

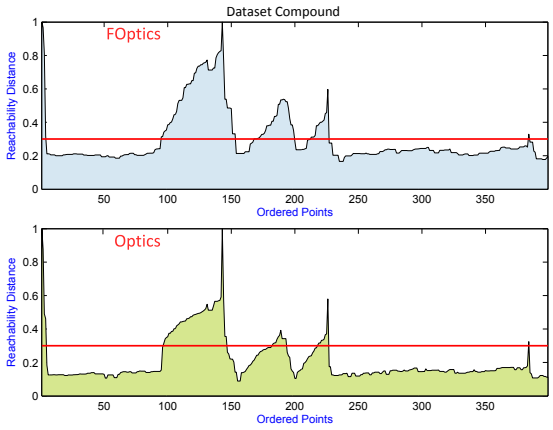

Figure 3: Reachability plots of FOPTICS and OPTICS for the Compound dataset. Note that both exhibit the same hills and valleys and hence discover the same clusters.

More precisely, $\epsilon(A)$ states the maximal distance of a point B that might be merged with A, i.e. $\epsilon(A) := D_{avg}(A)$. Point A can reach point B, if B is a candidate to be merged, i.e. $B \in \mathcal{N}_C(A)$, see Section 4. The definition of the reachability distance for a point A and a reachable point B from A is the same as for OPTICS, it is the maximum of the core-distance of A and the distance of A and B. However, for a point A we only compute the reachability distance to all sampled neighbors $B \in \mathcal{N}(A)$.

In Figure 3 we provide a visual illustration for the reachability plot computed on one dataset for OPTICS and FOPTICS. It is apparent that both techniques can reveal the same cluster structure.

THEOREM 5.1. *FOPTICS runs in $O(dN \log^2 N)$ whp.*

PROOF. Computing all candidate neighbors takes time $O(dN \log^2 N)$ whp according to Theorem 4.1. For each point we consider all candidate mergers at most once. This takes time $O(N \log N)$. □

6. PARAMETER-FREE DENSITY BASED CLUSTERING

Density-based clustering algorithms require as input a parameter that captures some notion of density, e.g. the minimum number of points which constitute a cluster or the volume used to estimate density. Different parameters may

lead to different clustering outcomes. Figure 4 shows such an example, consisting of two elliptical clusters connected by a thin line of points.

The thin line is denser when the density is estimated using small volumes (smaller ϵ values). For larger volumes the left and right elliptical areas become denser. The outcome of OPTICS for different parameters is also shown in Fig. 4. Notice that the cluster structure may change, leading to clusters merging or separating. We propose an efficient algorithm without data-dependent parameters to capture the variability of cluster structure.

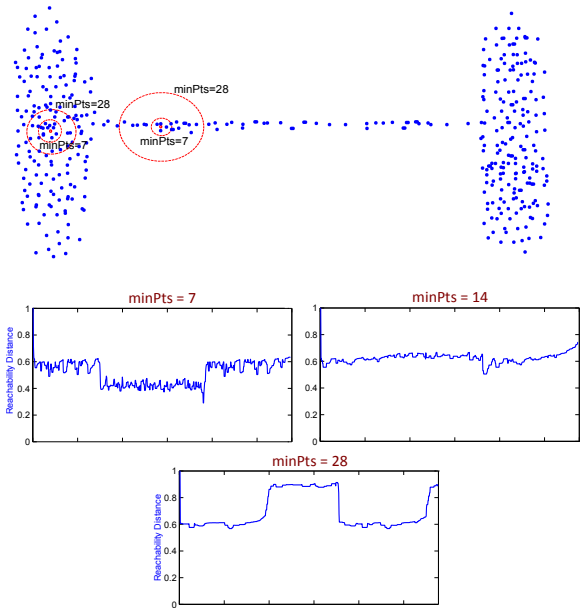

Figure 4: The choice of parameters, i.e. data density, greatly affects the resulting cluster structure. The outcome of OPTICS for different parameters $minPts = 7, 14$ and 28, suggesting three, one and three clusters, respectively.

6.1 Algorithm *DeBaRa*

Algorithm *DeBaRa* (*Density Based Clustering via Random Projections*) gradually increases the average distance of a point, i.e. parameter $dPts$ for computing the sampled neighbors (Algorithm 2). For a fixed value of $dPts$ it identifies points that potentially split clusters, i.e. points that belong to volumes of lower density than their surrounding. Any point that is not classified as low density point merges with all nearby points, whereas low density points greedily merge with the point of maximum density within some distance.

Two points A and B are adjacent (or nearby) if their Euclidean distance $D(A,B)$ is less than their merging distance: $D(A,B) \leq m(A,B)$. Each point A identifies the point $B \in \mathcal{N}_C(A)$ among the merging candidates of maximum density (that might also be itself). Any such identified point B is called *non-separating*. Any non-separating point A merges with all clusters that have a point B within half the merging distance and any non-separating point C within the merging distance.

The assumption behind this rule is that two clusters are separated by points of low density. Points of low density have large average distance D_{avg} and it is more likely that they merge with a distant point A. Such a point A has larger density, i.e. smaller D_{avg}. Furthermore, point A is likely to be somewhat distant from the cluster border. Therefore, it can merge with all the points within (half the) merging distance without merging with another cluster separated by points of low density.

Algorithm 4 DeBaRa(points $\mathcal{P}$) return $\mathfrak{C}^i$

```
c_0 := 1.1 {Granularity of parameter increase}
𝔖 := result of MultiPartition(P, N/c_0^{kc_1})
for i = 2 to log N / log c_0 do
    𝔖_i := {S | S ∈ 𝔖, s.t. |S| ∈ [|P|/c_0^i, |P|/c_0^{i+1}]} {All sets within size [|P|/c_0^i, |P|/c_0^{i+1}]}
    D_avg(A) := AverageDistance(𝔖_i, c_0^i/2) {Approximate average distance of A}
    m(A,B) := min(D_avg(A), D_avg(B)) {Merging distance of A and B}
    N(A) := for each set S with A ∈ S choose a point B ∈ S randomly with D(A,B) < m(A,B) {Neighbors of A}
    Pmax(A) := arbitrary B ∈ N(A), s.t. D_avg(B) = min_{C∈N(A)} D_avg(B) {Neighbor of maximum density of A}
    S^{no.sep} := {P ∈ P | P = ∃A ∈ P with Pmax(A) = P} {Non-separating points}
    𝔠_i := {Cl(P) | Cl(P) = {P}, P ∈ P} {Initial clusters}
    Merge clusters Cl(A) with Cl(Pmax(A)), ∀A ∈ P
    for all P ∈ S^{no.sep} do
        Merge all clusters Cl(P) with Cl(A) for A ∈ N(P) and D(A,B) < m(A,B)/2
    end for
    for all P, Q ∈ S^{no.sep} do
        Merge all clusters Cl(P) with Cl(Q) for Q ∈ N(P)
    end for
end for
```

THEOREM 6.1. *Algorithm DeBaRa runs in $O(dN \log^2 N)$ time whp.*

PROOF. Computing all candidate neighbors requires time $O(dN \log^2 N)$ whp based on Theorem 4.1. For a fixed $dPts$ for each point we consider all candidate mergers at most once. This takes time $O(N \log N)$. A merger of clusters $Cl(P)$ and $Cl(Q)$ takes time proportional to the size of the smaller cluster. There are at most $N-1$ mergers. The running time is maximal if both merged clusters are of the same size. Therefore, we have $N/2$ mergers of clusters of size 1, $N/4$ of clusters of size 2 and so on. Thus all merger operations together require at most $\sum_{i\in[0,\log N-1]} 2^i \cdot N/2^{i+1} = \sum_{i\in[0,\log N-1]} N/2 = N/2 \log N - 1$ time. We consider $\log N$ distinct values for $dPts$ yielding time $O(N \log^2 N)$. □

7. EMPIRICAL EVALUATION

We evaluate the runtime and clustering quality of the proposed random projection based techniques. The FOPTICS and DeBaRa algorithms have been implemented in Java. We compare their performance with OPTICS and DeLi-Clu, from the Elki Java Framework [1]. DeLi-Clu represents an improvement of OPTICS leveraging indexing structures, such as R*-trees, to improve performance.

Parameter setting: OPTICS requires parameters ϵ, and $minPts$. ϵ is set to infinity, which provides the most accurate results. $minPts$ depends on the dataset. DeLi-Clu requires

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

only the *minPts* parameter. FOPTICS uses the same parameter value for *dPts* as *minPts* for OPTICS (and DeLi-Clu): $dPts = minPts$. DeBaRa uses no data-dependent parameters. We performed 100 partitionings, that is, calls to algorithm Partition from MultiPartition, of the entire dataset for FOptics and DeBaRa and used $f_c = 1$ for the computation of D_{avg}.

Cluster Quality: The original motivation of our work was to provide faster versions of existing density-based techniques while not compromising accuracy. To compare the clustering we use the Rand index [11], which returns a value between 0 and 1, where 1 indicates identical cluster results. The results for various datasets are summarized in Table 1. FOPTICS provides almost perfect cluster preservation with OPTICS. As shown in previous examples the reachability plots for FOPTICS and OPTICS are very similar. For the chosen parameter $dPts = minPts$ the reachability plot of FOPTICS is typically *smoother*, following from the definition of the average distance D_{avg} that computes a smoother estimate of the density than OPTICS.

Name	Type	Objects	Dim	Clusters	Rand Index FOPTICS - OPTICS
Aggrega.[7]	shape	788	2	7	0.99
Compou.[14]	shape	399	2	6	0.98
Spiral[3]	shape	312	2	3	0.99
R15[12]	shape	600	2	15	0.98
Jain[9]	shape	373	2	2	0.97
Flame[6]	shape	240	2	2	0.94
Glass[1]	material	214	10	7	0.99
Iris	plants	150	4	3	1

Table 1: Cluster Preservation (1 is best)

Runtime: Our performance comparison in Figure 5 suggests a drastic improvement of FOPTICS and DeBaRa compared to both OPTICS and DeLi-Clu. FOPTICS is more than 500 times faster than OPTICS and more than 20 times faster than DeLi-Clu. Note, that DeLi-Clu is using an R*-tree structure to speedup various operations. Our approach bases its runtime improvements on random projections, thus is much simpler to implement and maintain.

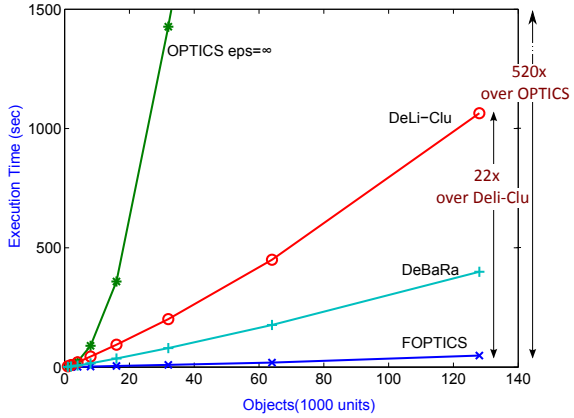

Figure 5: FOPTICS runs faster than OPTICS or DeLi-Clu.

Figure 6 highlights the runtimes for increasing data dimensionalities for a synthetic dataset of Gaussian clusters. 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

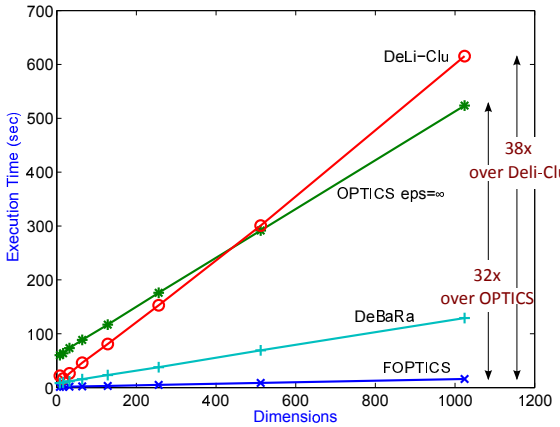

Figure 6: Performance of DeLi-Clu diminishes for higher dimensions, due to its use of indexing techniques.

a by-product of the use of indexing techniques by DeLi-Clu. It is well understood that the performance of space-partitioning indexing structures like R-trees, diminishes for increasing dimensionalities. The performance improvements of FOPTICS compared to OPTICS range from 47x (at low dimensions) to 32x (for high dimensions). A different trend is suggested in the runtime improvement against DeLi-Clu ranging from 17x (at low dimensions) to 38x (at high dimensions). Therefore, when dealing with high-dimensional datasets it is best to resort to techniques based on random projections.

In general, DeBaRa is slower than FOPTICS but does not require to set any parameters, making it an excellent candidate for exploratory data analysis.

8. REFERENCES

[1] UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml/datasets.html`. Accessed: 10/11/2012.

[2] M. Ankerst, M. M. Breunig, H. peter Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *SIGMOD*, 1999.

[3] H. Chang and D.-Y. Yeung. Robust path-based spectral clustering with application to image segmentation. In *Pattern Recognition*, 2008.

[4] R. Chitta and M. N. Murty. Two-level k-means clustering algorithm for k-tau relationship establishment and linear-time classification. *Pattern Recognition*, 2010.

[5] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *STOC*, 2008.

[6] L. Fu and E. Medico. Flame, a novel fuzzy clustering method for the analysis of dna microarray data. In *BMC bioinformatics*, 2007.

[7] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *TKDD*, 2007.

[8] A. Hinneburg and H.-H. Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. In *IDA*, pages 70–80, 2007.

[9] A. K. Jain. Data clustering: User's dilemma. In *MLDM*, 2007.

[10] M. Koyut, A. Grama, and N. Ramakrishnan. Compression, clustering, and pattern discovery in very high-dimensional discrete-attribute data sets. *IEEE TKDE*, 2005.

[11] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, (336), 1971.

[12] C. J. Veenman, M. J. T. Reinders, and E. Backer. A maximum variance cluster algorithm. *Pattern Anal. Mach. Intell.*, 2002.

[13] J. J. Whang, X. Sui, and I. S. Dhillon. Scalable and memory-efficient clustering of large-scale social networks. In *ICDM*, 2012.

[14] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. Comput.*, 1971.