Datenbank Spektrum (2019) 19:219–230
https://doi.org/10.1007/s13222-019-00324-9

SCHWERPUNKTBEITRAG

Chain-detection Between Clusters

Janis Held[1] · **Anna Beer**[1] · **Thomas Seidl**[1]

Received: 10 June 2019 / Accepted: 31 August 2019 / Published online: 13 September 2019

Abstract
Chains connecting two or more different clusters are a well known problem of clustering algorithms like DBSCAN or Single Linkage Clustering. Since already a small number of points resulting from, e. g., noise can form such a chain and build a bridge between different clusters, it can happen that the results of the clustering algorithm are distorted: several disparate clusters get merged into one. This single-link effect is rather known but to the best of our knowledge there are no satisfying solutions which extract those chains, yet. We present a new algorithm detecting not only straight chains between clusters, but also bent and noisy ones. Users are able to choose between eliminating one dimensional and higher dimensional chains connecting clusters to receive the underlying cluster structure. Also, the desired straightness can be set by the user. As this paper is an extension of [8], we apply our technique not only in combination with DBSCAN but also with single link hierarchical clustering. On a real world dataset containing traffic accidents in Great Britain we were able to detect chains emerging from streets between cities and villages, which led to clusters composed of diverse villages. Additionally, we analyzed the robustness regarding the variance of chains in synthetic experiments.

Keywords DBSCAN · Agglomerative single link clustering · Clustering · Chain-detection · Single link effect

1 Introduction

In contrast to most centroid-based cluster methods, density-based algorithms are able to find non-convex clusters.

The famous density-based algorithm DBSCAN [4] builds clusters around points with high density, so-called seed points, and expands them taking all density-connected points into account as described in Section 2.1. As long as the clusters are clearly separated, this procedure works very well, but if there are, e. g., some density-connected noise points creating a chain between clusters, DBSCAN expands the cluster along these chains resulting in a single huge cluster instead of the intuitive ones. A similar effect occurs for hierarchical clustering with the single link distance, which is why we extend the algorithm introduced in [6] in this paper. We developed an algorithm which detects such chains between clusters. For that we use PCA (Principal Component Analysis), assuming that a chain has a lower dimensionality than the clusters it connects. Fig. 1 shows an example where two 3D clusters are connected by a red chain with only little expansion in two of the three dimensions. Our algorithm is adaptable, users can choose which type of chains they want to connect: straight chains or bent ones, noisy or thin ones. Through recognizing those chains and eliminating them from the clustering the underlying individual clusters can be revealed.

The paper is structured as follows: First, we introduce shortly the related work and basics we use in Section 2. In Section 3 we explain our novel method to find chains in detail, giving an overview over the whole algorithm in Section 3.6. We analyze the complexity in Section 4 and prove its effectiveness in Section 5 with some experiments on real world as well as on synthetic data. In Section 8 we conclude and give a brief idea of some future work.

2 Related Work and Basics

There are already many extensions of DBSCAN like, e. g., ST-DBSCAN, an extension for clustering spatial-temporal data [2], MR-DBSCAN, which is an efficient parallel density-based clustering algorithm using map-re-

Janis Held
j.held@campus.lmu.de

✉ Anna Beer
beer@dbs.ifi.lmu.de

Thomas Seidl
seidl@dbs.ifi.lmu.de

[1] LMU Munich, Munich, Germany

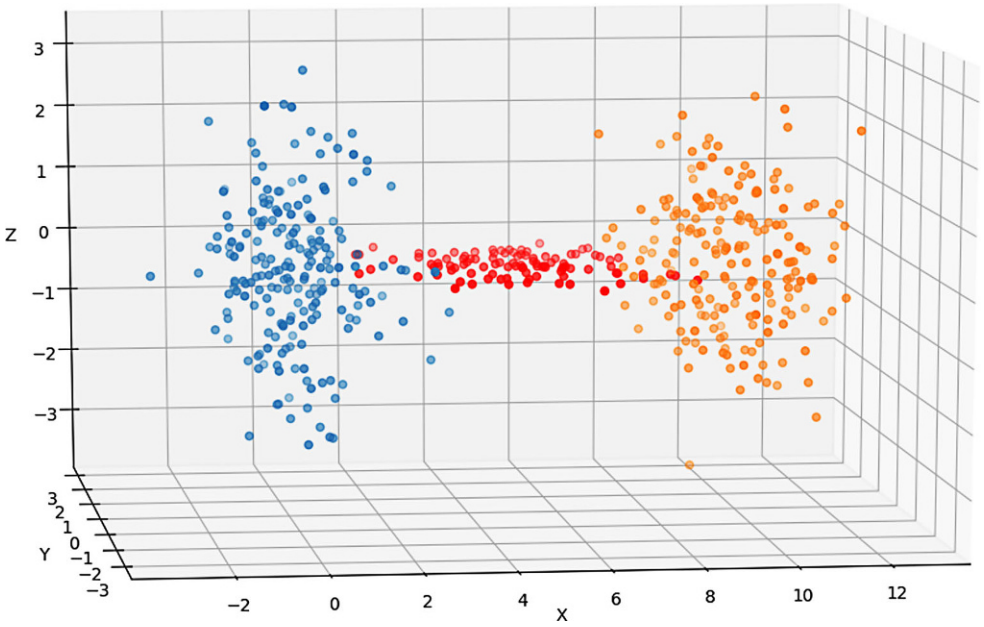

Fig. 1 The *red points* cause a density-connection between the intentional two clusters and thus form a chain

duce [7], or C-DBSCAN: Density-based clustering with constraints [11]. To the best of our knowledge, there is yet no extension of DBSCAN to circumvent the disadvantages of the single-link effect or chains connecting clusters. Also for single-linkage clustering [12] many long-established extensions exist, like, e. g., [5], which uses complete-linkage as stopping criterion for single linkage, or methods focusing on efficiency, as introduced in [3] or [10]. But also here, there are to the best of our knowledge no algorithms yet to prevent chains connecting clusters, even though there are methods like, e. g., [1] focusing on robustness against noise. In this section, we give the basics needed for the following sections, namely some details of DBSCAN, single-linkage clustering, and the Principal Component Analysis (PCA).

2.1 DBSCAN

Density-based spatial clustering of applications with noise [4] is a density based clustering algorithm that clusters points based on their density and marks outliers lying in low-density regions. A point with at least *minPts* points in its ε-range is called a core point. All points in the ε-range of a core point c belong to the same cluster as c and are called density-reachable from c. All reachable points are assigned to the cluster from which they are reachable, while points which are neither reachable nor core points are declared noise. Like that, it is possible that a small chain of density-reachable points connects two clusters as Fig. 2 shows.

2.2 Single Linkage Clustering

[12] is an agglomerative hierarchical method, which starts with every point building its own cluster and combining the two clusters with the lowest single-linkage distance in every step. The single-linkage distance D_{SL} between two clusters X and Y is the smallest distance between two points of those clusters and is defined as follows, where $d(x, y)$ is the distance between two elements x and y:

$$D_{SL}(X, Y) = \min_{x \in X, y \in Y} d(x, y) \tag{1}$$

2.3 PCA

(Principal Component Analysis) [9] transforms given data points to a new coordinate system where the greatest variance by any projection of the data lies along the first coordinate (the first principal component), the second greatest variance along the second coordinate, and so on. PCA is a good indicator of how well some data fits into a lower dimensional subspace.

PCA regards the eigenvalue decomposition of the data covariance matrix, usually after mean centering the data matrix for each dimension. Then the eigenvectors of the covariance matrix form an orthogonal basis and each eigenvalue describes how much variance is explained by its corresponding eigenvector [9].

Let d be the dimensionality of the data space Ω and $N = \{n_1, ..., n_m\}$ the ε range of some point $p \in \Omega$. The data matrix is defined as $(n_1, ...n_m)^T$. Let m_j be the mean

Fig. 2 The *red dots* connect two clusters and thus form a chain

Fig. 3 Since the chain-like looking *red dots* do not connect any clusters, they are not considered a chain

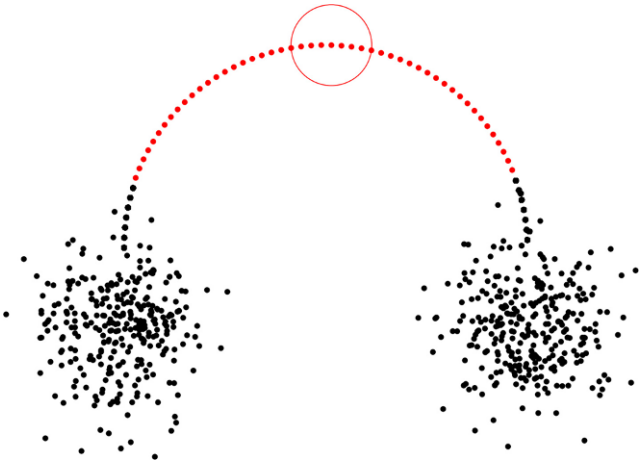

Fig. 4 The *red dots* may or may not be a chain, depending on the user. The *red circle* is one of the ε ranges

of column j. One can now calculate the covariance matrix Θ with

$$\Theta_{ij} = \frac{\sum_{k=1}^{m}(n_{ki} - m_i)(n_{kj} - m_j)}{m},$$
$$i = 1, .., d, \quad j = 1, .., d.$$

Note that the covariance matrix is symmetric and positive semi-definite, thus its eigenvalues are non negative. Finally, the eigenvalues are normalized by dividing them by the sum of all eigenvalues, such that the sum of all normalized eigenvalues equals to 1.

3 Chain-detection

Let $Clustering(X)$ be the DBSCAN or Single-Linkage Clustering of some data X and C be a cluster found in the data space Ω. We want to find a set of candidates that may form chains in C. With the assumption of chains having a subdimensional shape we can use a definition of neighborhood and look for an algorithm that decides for each point if it lies within a subdimensional neighborhood. As both clustering algorithms are metric-based we already have a metric d and with a parameter ε the neighborhood of a point p is defined as $\{x|d(p,x) \leq \varepsilon\}$. For DBSCAN ε should be set to the value of the DBSCAN parameter ε, whereas it is given by the user for Single-Linkage Clustering. The idea is to use the distribution of all points in the neighborhood of each point as an indicator for its likelihood do be part of a chain. Therefore, a point in C is considered a **shape-based chain-point candidate** if there exists a subspace with a lower dimensionality than Ω, such that all points of the ε range of p lie close to it. Note that "lower dimensionality" and the word "close" will become parameters for the chain-detection algorithm. Clustering all remaining points may result in some noise points. We call the union of shape-based chain-point candidates with all those noise points **chain-point candidates**. Now we can cluster the chain-point candidates and each cluster is called a **chain-candidate**. Note that all chain-point candidates which were marked as noise are not part of a chain-candidate, because we want a chain to be at least big and dense enough to form a cluster itself. The last step will be to validate if the chain-candidate indeed connects two clusters of the remaining points and is not some kind of tail.

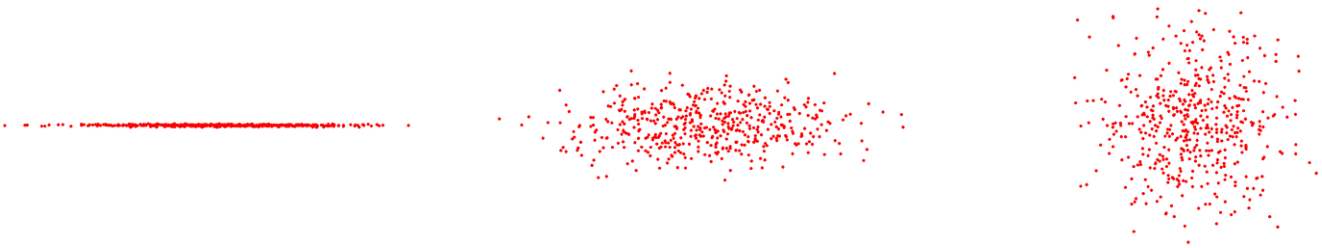

Fig. 5 Various degrees of fuzziness for normalized errors: $\bar{e} \approx 0.0002$, $\bar{e} \approx 0.1563$, and $\bar{e} \approx 0.9997$

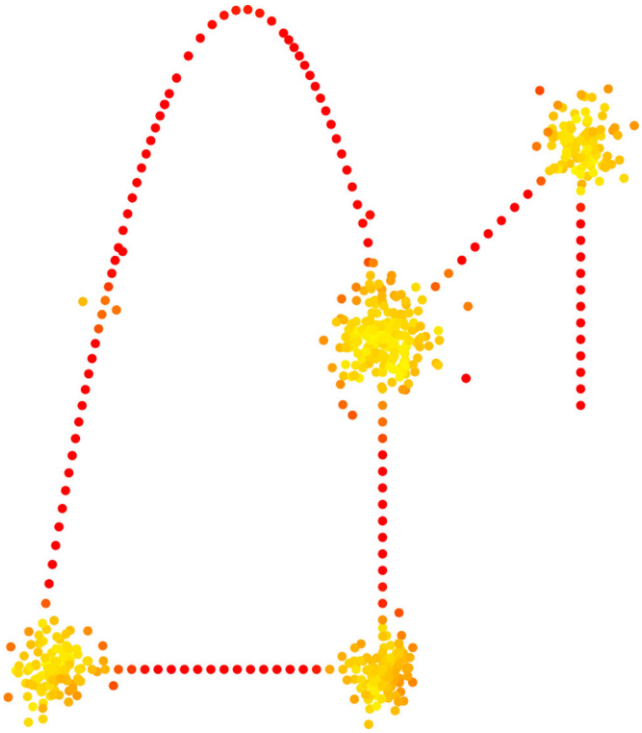

Fig. 6 Example data: Each *point* is colored by the normed error $\bar{e}$ derived from its ε range. *Yellow* means the error is close to 1 and *red* means it is close to 0

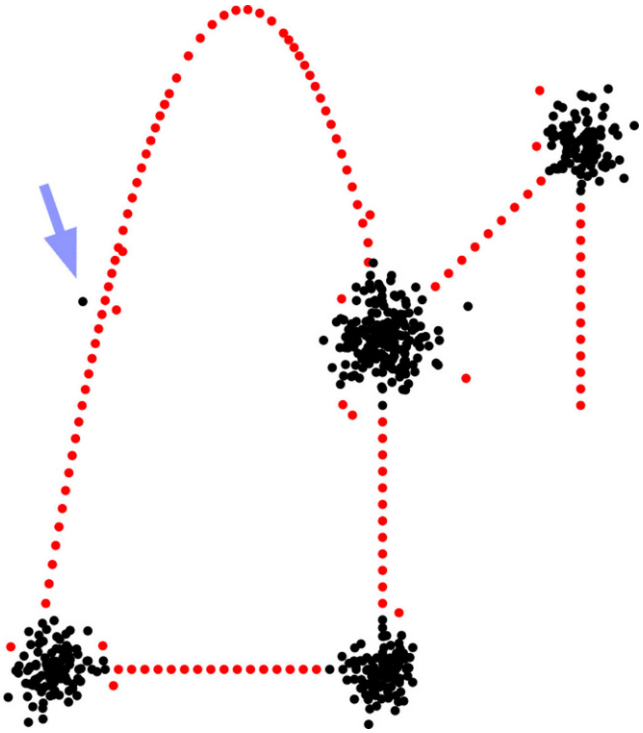

Fig. 7 Example data: With *allowedVariation* = 0.2 the *red points* are selected as shape-based chain-point candidates. The *arrow* highlights an outlier

3.1 Chains

Assume the user wants to detect one-dimensional chains in a two-dimensional data space and *Clustering* would not label the red dots in the following figures as noise, then Fig. 2 shows a simple example of a chain. The red dots in Fig. 3 are not considered a chain, because they do not form a connection between two clusters. The red dots in Fig. 4 are not perfectly linear, because the ε range of each red point (the red circle is one of the ε ranges) does not perfectly fit inside a one dimensional subspace, and thus it depends on the user if he wants to detect those as a chain.

3.2 Chain-Point Candidates

For each point in a cluster C the objective is to determine if this point is a chain-point candidate. To achieve this, for each point $p \in C$ the technique behind principal component analysis (PCA) is utilized to calculate how good the ε range of p fits inside a subspace with a dimensionality lower than the dimensionality of the data space Ω. To be more precise, PCA is utilized to find this subspace and then to calculate the explained variation of those ε-neighbors of p which do not fit inside this subspace.

Theorem 1 Let d be the dimensionality of the data space Ω and $N = \{n_1, ..., n_m\} \subset \Omega$ be the ε range of some point $p \in \Omega$. Furthermore let $\lambda_1 \geq ... \geq \lambda_d$ be the sorted normalized eigenvalues of the covariance matrix Θ derived from N.

1. If $\lambda_d = 0$, then N lies inside a hyperplane.
2. If $\lambda_d = 1/d$, then N is perfectly distributed across all dimensions.
3. if $\lambda_i = 0$ and $1 < i < d$, then N lies inside a subspace with dimension $i - 1$.

Proof

1. If $\lambda_d = 0$, then the corresponding eigenvector ev_d describes 0 variance. Since the eigenvectors form a orthogonal basis N lies entirely in the hyperplane orthogonal to ev_d.
2. Since the sum of all eigenvalues equals to 1 and there are d eigenvalues and all are non negative, each eigenvalues must be equal to $1/d$. That means each eigenvector describes the same variance, thus N is perfectly distributed across all dimensions.
3. Since the eigenvalues are sorted, non negative and $\lambda_i = 0$ it follows that $\lambda_j = 0, \forall j \in \{i, ..., d\}$. That means the corresponding eigenvectors ev_j, with $j \in \{i, ..., d\}$ of

Algorithm 1 Validate Chaincandidates

```
1: procedure ValidateCandidate(Chain, R, DB_R, dist, ε)
2:     clusterFound ← null
3:     for c ∈ Chain do
4:         for p ∈ RangeQuery(R, dist, c, ε) do
5:             if clusterFound == null then
6:                 clusterFound ← DB_R.labelFor(p)
7:             else
8:                 if clusterFound ≠ DB_R.labelFor(p) then
9:                     return True
10:    return False
```

the orthogonal basis describe 0 variance. Thus, N lies entirely in the subspace spanned by $ev_j, j \in \{1, ..., i-1\}$.

3.3 Parameters

With theorem 1 one can now define the three parameters for the chain-detection algorithm

1. $\varepsilon > 0$, which determines the size of the neighborhood for each point.
2. *chainDim* $\in \{1, ..., d-1\}$, which describes the dimensionality of chains the user wants to detect.
3. $allowedVariation \in [0,1[$, which allows variation beyond the allowed dimensionality of the chain.

Like in Section 3.2, let $N = \{n_1, ..., n_m\}$ be the neighborhood of some point $p \in C$ and $\lambda_1, ..., \lambda_d$ the descending sorted normalized eigenvalues of the covariance matrix Θ corresponding to N. To calculate how good N lies within a *chainDim* dimensional subspace, one calculates the accumulated error

$$e := \sum_{i=chainDim+1}^{d} \lambda_i .$$

The sum starts with *chainDim* + 1, because only the $d -$ *chainDim* least significant principal components explain the variation beyond the wanted chain dimensionality. It holds that $\lambda_d \in [0,1/d]$, because the sum of all eigenvalues equals to 1, there are d eigenvalues and λ_d is the smallest one. If $\lambda_d < 1/d$ then $\lambda_1 > 1/d$, otherwise λ_1 would not be the largest normalized eigenvalue. That means the sum of the i smallest normalized eigenvalues is at most i/d, that is if all eigenvalues are $1/d$. Thus, $e \in [0, (d - chainDim)/d]$. To make the user-input independent of the dimensionality of Ω and *chainDim*, one normalizes the error by

$$\bar{e} := e * \frac{d}{d - chainDim} \in [0,1]. \tag{2}$$

p is a chain-point candidate if $\bar{e} \leq allowedVariation$.

3.4 Fuzziness of Chains

In Fig. 5 examples for various values of normed errors are given for a two dimensional data space with *chainDim* = 1.

Algorithm 2 Chain-detection

```
1: procedure Chain-detection(C, dist, ε, minPts, chainDim, allowedVariation)
2:     d ← dim(C)                                   ▷ Dimensionality of the data
3:     C_c ← {}                                     ▷ Set of chain-points
4:     for p ∈ C do                                 ▷ Find all chain-point candidates
5:         N ← RangeQuery(C, dist, p, ε)
6:         EV ← EigenValues(CovarianceMatrix(N))
7:         EV ← EV/EV.sum()                         ▷ Norm eigenvalues
8:         EV ← EV.sorted(descending=TRUE)          ▷ Sort eigenvalues descending
9:         e ← EV.sum(start=d − chaindim + 1)       ▷ Calculate error
10:        e ← e * (d/(d − chainDim))               ▷ Norm error
11:        if e ≤ allowedVariation then             ▷ Compare error with parameter
12:            C_c ← C_c ∪ {p}                      ▷ Add p to the set of chain-points
13:    if |C_c| == 0 then return {}
14:    R ← C \ C_c                                  ▷ Set of remaining points
15:    DB_R ← Clustering(R, dist, ε, minPts)        ▷ Cluster the remaining points
16:    C_c ← C_c ∪ DB_R.Noise                       ▷ Add noise to the set of chain-points
17:    DB_{C_c} ← Clustering(C_c, dist, ε, minPts)  ▷ Cluster chain-points
18:    if |DB_{C_c}.clusters| == 0 then return {}   ▷ No chain-candidate found
19:    R ← C \ ∪DB_{C_c}.Clusters                   ▷ Update the set of remaining points
20:    DB_R ← Clustering(R, dist, ε, minPts)        ▷ Cluster the remaining points
21:    if |DB_R.clusters| ≤ 1 then return {}        ▷ No chain-candidate can be validated
22:    Chains ← []                                  ▷ The list of validated chains
23:    for V ∈ DB_{C_c}.Clusters do                 ▷ Validate each chain-candidate
24:        if ValidateCandidate(V, R, DB_R, dist, ε) then
25:            Chains.append(V)
26:    return Chains
```

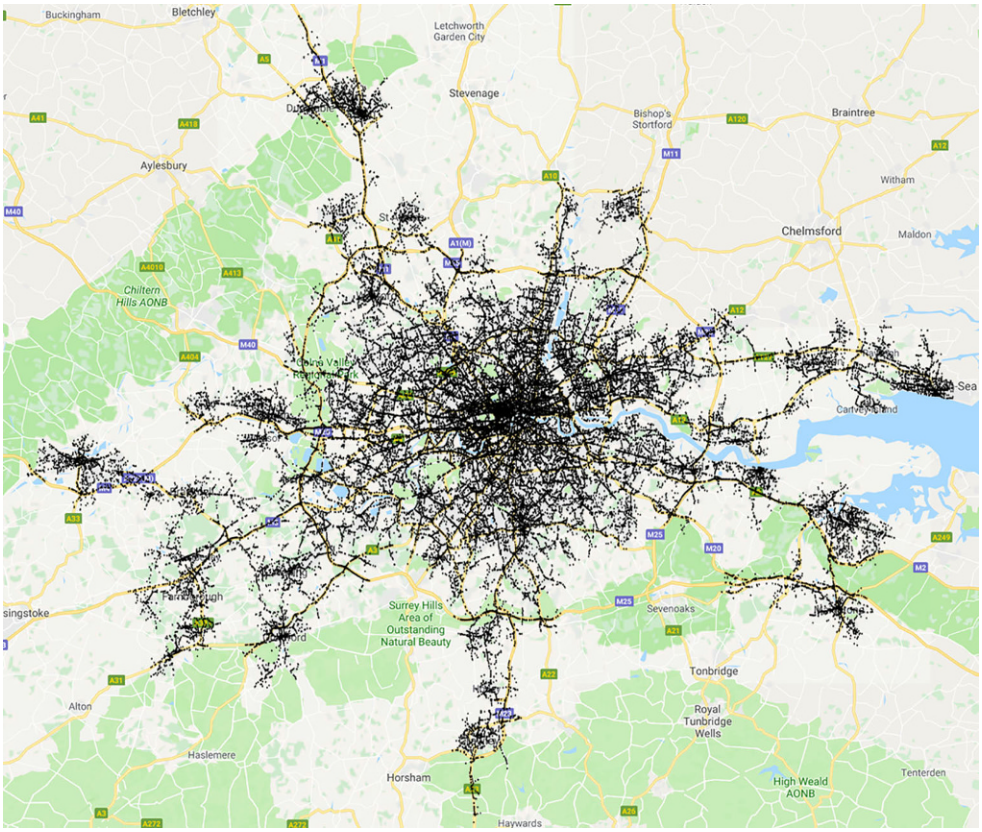

Fig. 8 The cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. The *dots* are stretched to fit the underlying map

$\bar{e}$ describes the variation beyond a linear subspace. The closer the points get to a linear subspace the lower the error gets and vice versa. In the most right example the error is close to 1 since the points are almost perfectly distributed in all directions.

Let us have a look at some synthetic example data. In Fig. 6 the points are colored by its normed error values with *chainDim* set to 1. Some points are clearly marked red, because they have a low normed error, indicating that they might be part of a chain. On the other hand most of the points inside those clouds have a high normed error because their ε range hardly fits into a one-dimensional subspace. Setting *allowedVariation* to some value determines for each point if it is a chain-point candidate. Setting *allowedVariation* to 0.2 on the data of Fig. 6 results in the shape-based chain-point candidates seen in Fig. 7.

3.5 Finding and validating chain candidates

Let $C_{\bar{e}}$ be the set of shape-based chain-point candidates. First of all each shape-based chain-point candidate is added to the set of chain-point candidates. After clustering the remaining points $C \setminus C_{\bar{e}}$ by *Clustering* all points marked as noise are not part of a cluster of non-candidates, indicating that they also might be part of a chain, see the highlighted black dot on the left of Fig. 7. These points are now added to the set of chain-point candidates.

Clustering the set of chain-point candidates results in clusters of chain-point candidates, which are the desired **chain-candidates** and noise.

Let $C_{ci}, i \in I$ be those chain-candidates, $R := C \setminus \cup_{i \in I} C_{ci}$ be the set of the remaining points and DB_R be $Clustering(R)$. Note that R contains those chain-point candidates, which were marked as noise by clustering all chain-point candidates. To validate C_{ci} check for each point $p \in C_{ci}$, if their ε range contains points $r \in R$ and note the cluster of r found in the clustering DB_R. As soon as two clusters are noted the chain is validated and considered a chain. If all points are checked but no two clusters are noted the chain-candidate C_{ci} could not be validated and is not considered a chain.

Finally we receive a set of chains – which can now be considered clusters themselves or simply marked as chains – and a set of remaining points, which remain to be clustered to get the final clustering without chains.

3.6 The complete algorithm

Let C be the cluster found by *Clustering* with parameters ε, *chainDim* and *allowedVariation* for the chain-detection algorithm. $RangeQuery(C, dist, p, \varepsilon)$ returns the set

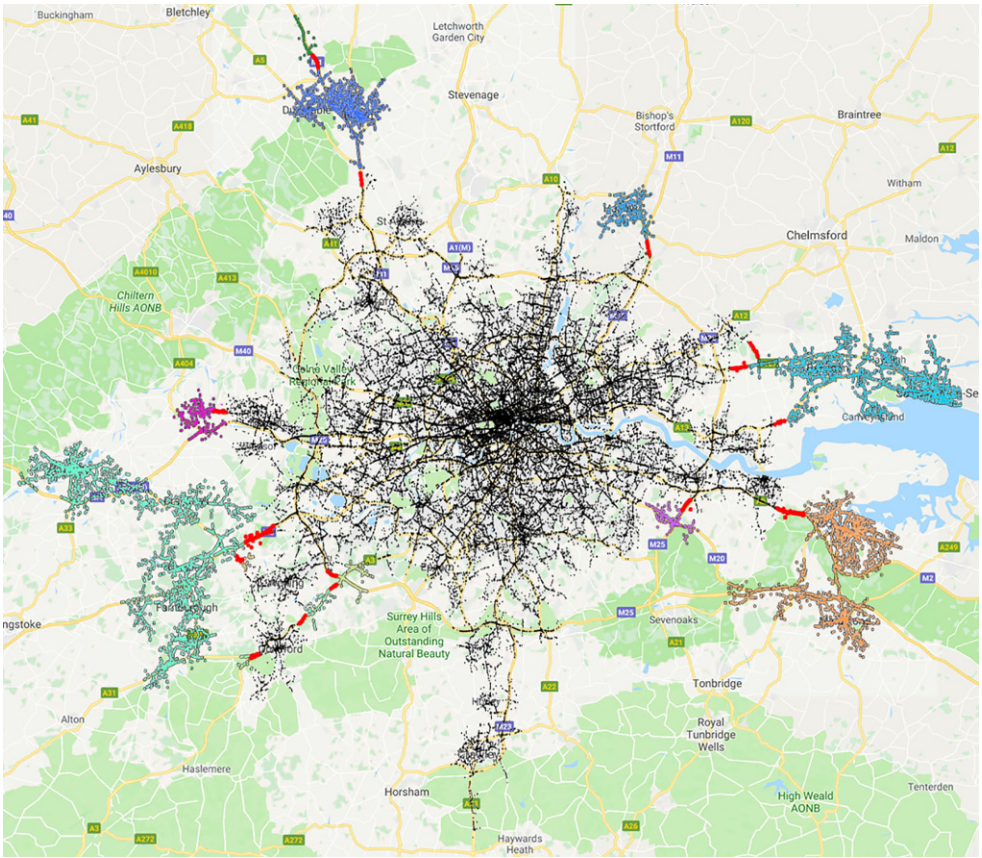

Fig. 9 Chain-detection applied on the cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. Chains are marked *red*

$\{q \in C | dist(p,q) \leq \varepsilon\}$. For the sake of simplicity assume the result of *Clustering* contains the property "Noise", which is the set of points marked as noise and the property "Clusters", which is the set of clusters. For Single-Linkage Clustering for example, a threshold for the cluster size could be set, declaring all points in clusters smaller than that threshold as noise. Algorithms 1 and 2 recapitulate our complete approach.

For a full implementation see https://github.com/Quesstor/DBSCAN-with-density-based-connection-detection.

4 Runtime complexity

Let n be the number of points in the cluster, on which the chain-detection algorithm is applied, in a d dimensional data space. For each point a range query with linear complexity is calculated. Calculating the covariance matrix of the ε-neighborhood, which in the worst case consists of all n points, is $O(n * d^2)$. Then the eigenvalues of the $d \times d$ covariance matrix is calculated, which has runtime complexity of $O(d^3)$. So the total runtime complexity for the *for* loop is $O(n(n + n * d^2 + d^3))$. Assuming *Clustering* on a subset of the cluster each have the worst case run time complexity of $O(n^2)$, as is the case for DBSCAN. The validation step calculates for less than n points a range query resulting in a worst case run time complexity of $O(n^2)$. So the *for* loop is causing the largest performance hit with a runtime complexity of $O(n(n + n * d^2 + d^3))$. Assuming $d \ll n$ one can simplify the runtime complexity to $O(n^2)$.

To improve performance the range queries should be executed on a tree structure and calculating the normed error for each point, which causes the largest performance hit, can easily be parallelized.

5 Experiments

The dataset on which the experiments are performed consists of all reported traffic accident locations in Great Britain from the years 2014–2016 [1].

For DBSCAN, parameters $\varepsilon := 0.01$ and $minPts := 15$, deliver a cluster containing chains of traffic accidents in the area of roughly 100km in each direction around London's center, which allows us to show the properties of our chain-detection algorithm.

[1] https://www.kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales/data

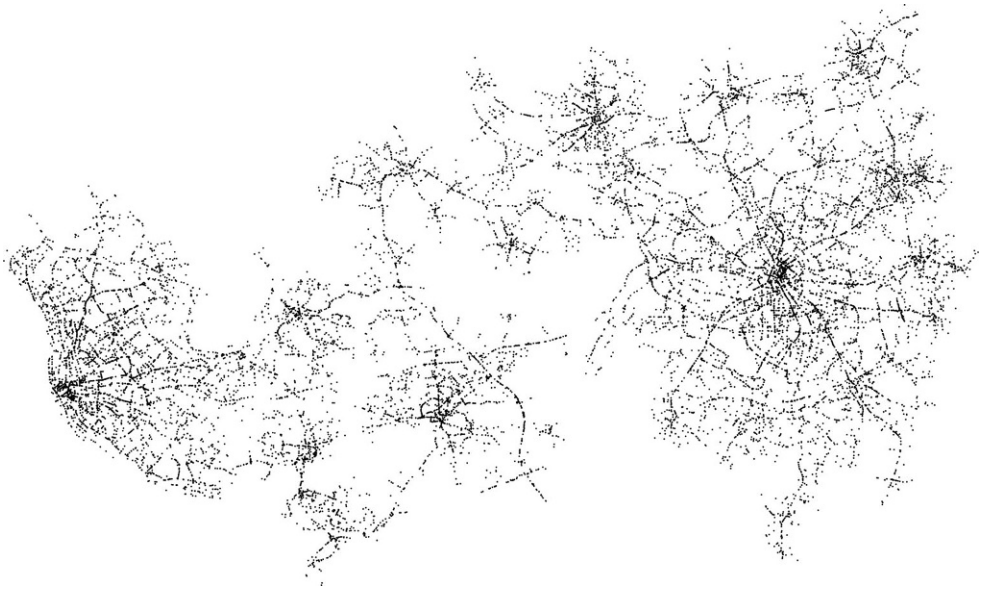

Fig. 10 The cluster of traffic accidents at Liverpool and Manchester

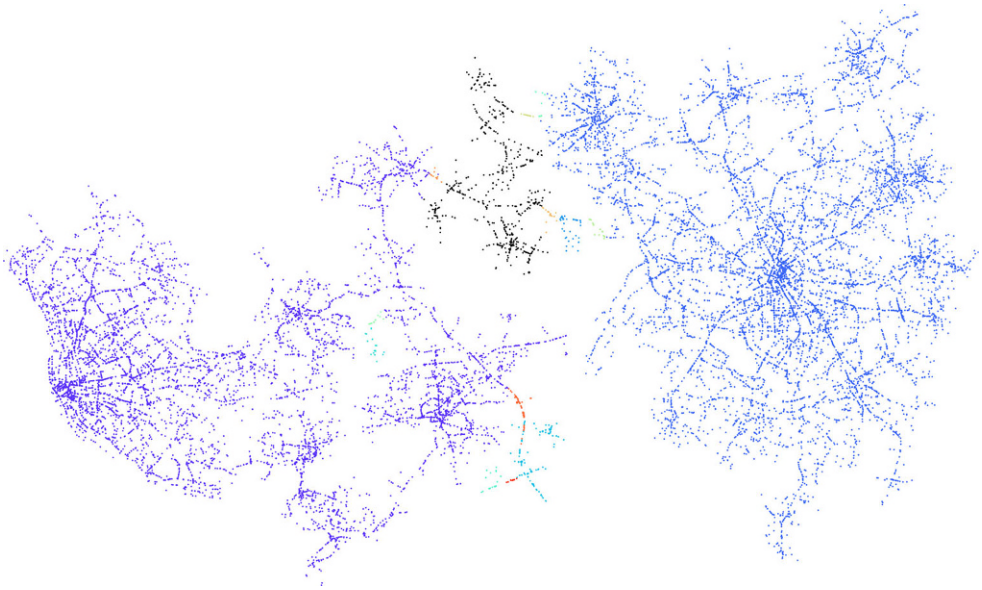

Fig. 11 Result of the chain-detection algorithm applied on the traffic accidents in Liverpool and Manchester

As DBSCAN already has a definition for neighborhood with the parameter ε we will use this value for the ε parameter for the chain-detection algorithm.

5.1 Traffic accidents in London

The chain-detection will be demonstrated on the cluster found at London city, see Fig. 8.

The results obtained by DBSCAN using the above mentioned parameters are not a satisfying clustering, because the highways, on which a lot of accidents happen, connect the suburban areas outside London to a single cluster. So let us apply the chain-detection algorithm. To detect these highways, which are basically one-dimensional chains, one sets the *chainDim* parameter to 1. Since the highways are not perfectly linear and surrounded by noise, one wants to allow some error and set the *allowedVariation* parameter to 0.2.

Fig. 9 shows the resulting clustering after applying the chain-detection algorithm. Most of the suburban areas are now separated from the main cluster of London city and almost all chains are found on highways.

5.2 Traffic accidents in Liverpool and Manchester

Another example is the cluster found at Liverpool and Manchester. As there are a lot of accidents between those cities both end up in the same cluster, see Fig. 10. Let us apply the chain-detection algorithm with parameters *chainDim* := 1 and *allowedVariation* := 0.2, for the same

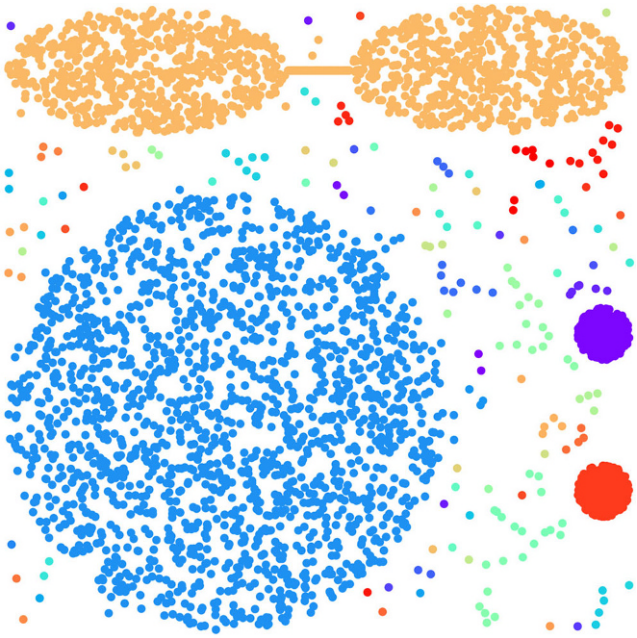

Fig. 12 Optimal single link clustering of cure-t2-4k with NMI score of ~ 0.8196

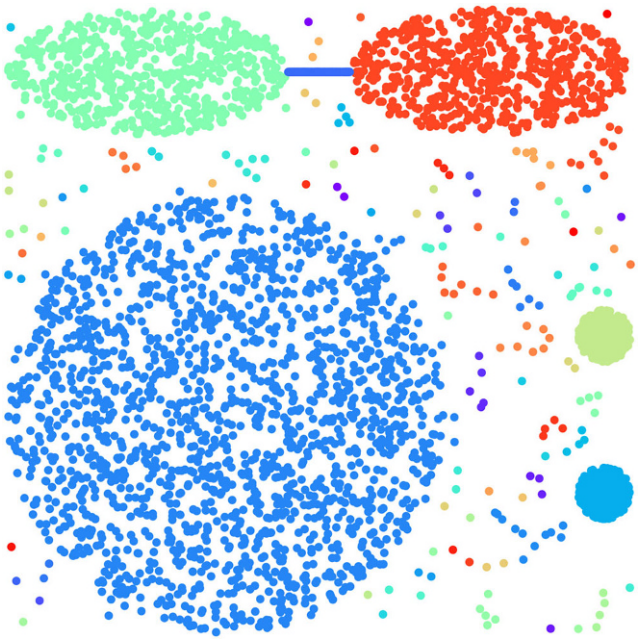

Fig. 13 Optimal single link clustering in combination with chain-detection of cure-t2-4k with NMI score of ~ 0.9382

reasons as in the previous example. In Fig. 11 we can see how the traffic accident clusters are now well divided, one cluster in Liverpool and one in Manchester.

5.3 Synthetic data

The clustering benchmark was performed on the labeled cure-t2-4k dataset[2]. First the parameters for DBSCAN and Singlelink clustering were obtained by optimizing the NMI score over a range of parameters, see below. Then the NMI score was compared to the best NMI score after applying the chain-detection algorithm with the same ε parameter from the DBSCAN or Single-Linkage clustering and $allowedVariation \in \{0.001, 0.003, 0.005, ..., 0.5\}$. As Fig. 12 shows, Single Link merges the two clusters in the upper area even with optimal parameters. Fig. 13 shows the result using the chain-detection, where the clusters are discerned.

For the Single-Linkage clustering, testing $\varepsilon \in \{0.001, 0.0015, 0.002, ..., 0.1\}$ resulted in the best parameter $\varepsilon = 0.0575$ with NMI score of ~ 0.8196. Applying the chain-detection algorithm improved the NMI score by ~ 0.1186 resulting in an NMI score of ~ 0.9382.

For DBSCAN all combinations of $minPts \in \{1, 2, ..., 20\}$ and $\varepsilon \in \{0.001, 0.002, ..., 0.1\}$ resulted in the best parameters $minPts = 6$ and $\varepsilon = 0.054$ with an NMI score of ~ 0.87423. Applying the chain-detection algorithm improved the NMI score by ~ 0.1175, resulting in an almost optimal NMI score of ~ 0.9917.

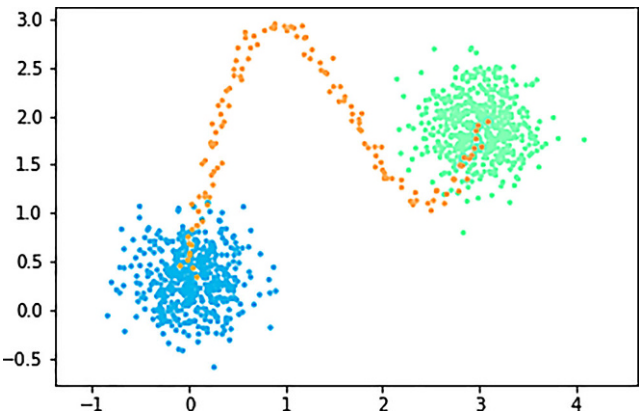

Fig. 14 Some random generated dataset with standard deviation of 0.05 for the chainpoint offset

6 Robustness of *allowedVariation* parameter

To demonstrate the effect of the *allowedVariation* parameter, we randomly generated example data by the following algorithm. First, four points $p_i = (i, y_i)$ are selected, where $y_i \in [0,4], i \in \{0, 1, 2, 3\}$ is a uniformly distributed random variable. Then two clusters are constructed by generating 500 normally distributed points with a standard deviation of 0.3 each around p_0 and p_3. To generate a chain, a univariate spline fit to all p_i is calculated and points are generated

[2] https://github.com/deric/clustering-benchmark/blob/master/src/main/resources/datasets/artificial/cure-t2-4k.arff

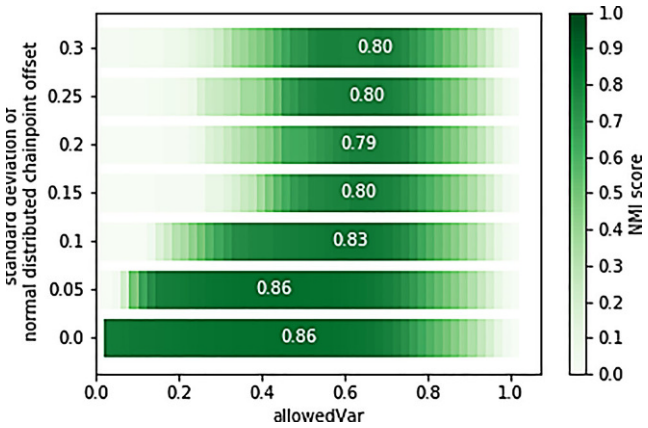

Fig. 15 NMI scores for DBSCAN + chain-detection with different chain densities and *allowedVariation* parameters. The *white number* is the highest NMI score found

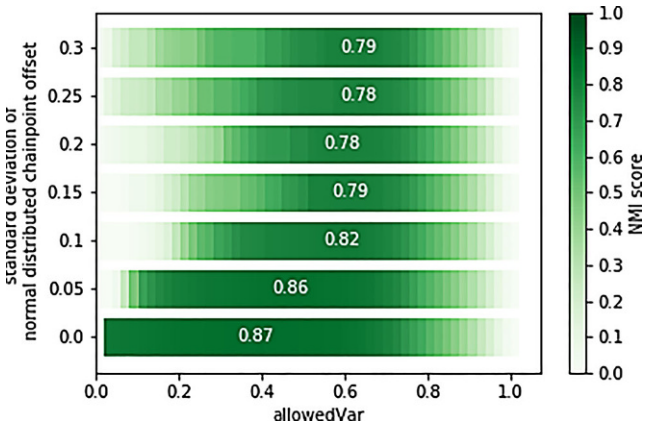

Fig. 16 NMI scores for single-link + chain-detection with different chain densities and *allowedVariation* parameters. The *white number* is the highest NMI score found

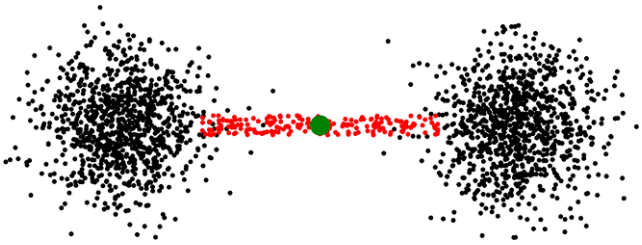

Fig. 17 The *red dots* will not be detected as a chain, because the ε range (marked as a *green circle*) is too small to detect the chain

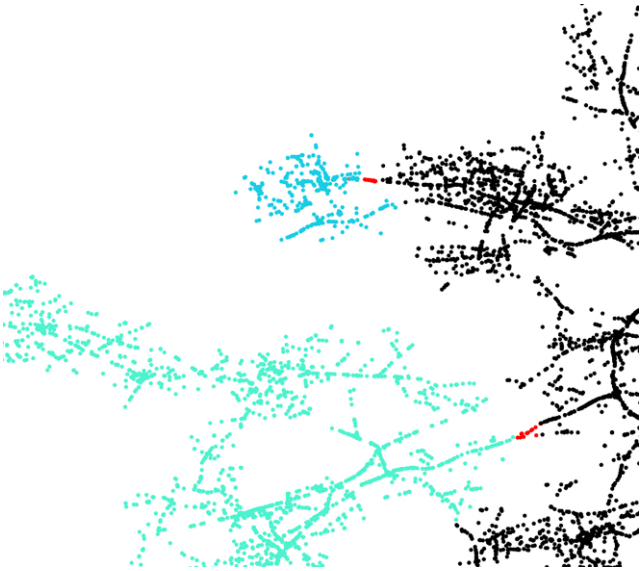

Fig. 18 Zoom in of Fig. 20c with $\varepsilon := 0.15$. Chains are marked *red*

along this spline such that the distance between two points is roughly equal to 0.05.

Finally, these chainpoints are distorted by adding a normally distributed offset with some standard deviation, see Fig. 14.

We tested different standard deviations for the chainpoints offset and for each we generated 50 datasets that clustered by DBSCAN with parameters $\epsilon = 0.3$ and $minPts = 4$ result in a single cluster. That means DBSCAN detected both clusters and the chain as a single cluster. Now we applied the chain-detection algorithm with different *allowedVariation* parameters and noted the average NMI score over all 50 datasets. Note that the NMI score of the DBSCAN clustering is near zero as DBSCAN detects only one cluster. The results in Fig. 15 show, that a large range of values for *allowedVariation* lead to very good NMI scores. For lower densities of the chain, i.e., a higher standard deviation, higher values for *allowedVariation* are better.

We made the same experiments for the single-link clustering, with the only difference that we took only datasets containing a cluster with at least 1000 points found by single-link clustering, because single-link clustering tends to create additional very small clusters.

The results shown in Fig. 16 are similar to those of DBSCAN.

7 Analysis of ε Parameter

Finding the right chain-point candidates, by looking at the shape of the ε range of each point, has a limitation regarding the ε. If the ε is too small, then some chains which may seem as a linear chain (when looking at the whole picture) will not be detected, see Fig. 17. To counter this limitation, one could simply increase the ε parameter.

The points inside that too-big-chain will have errors close to 1 and are therefore not selected as chain-point

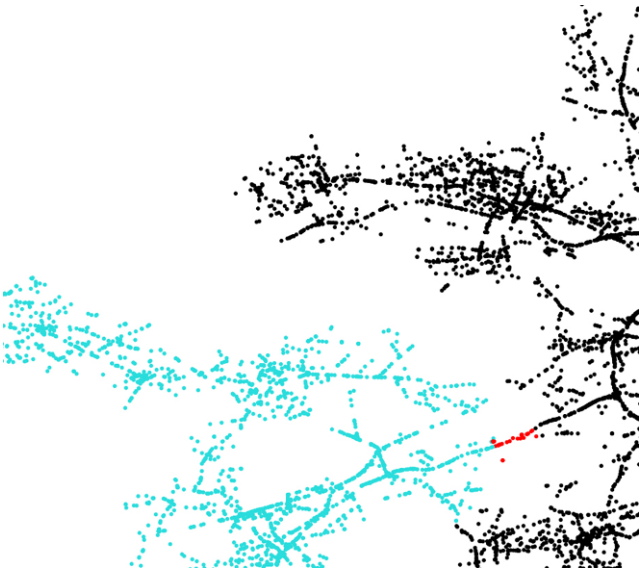

Fig. 19 Zoom in of Fig. 20d with $\varepsilon := 0.2$. Chains are marked *red*

candidates. The border points of the too-big-chain may be selected but may not be verified as chains, because the inner points may keep the clusters connected.

We here regard the possibility to set a third parameter, ε, for the range of the neighborhood when checking each point if it is a chain-point candidate. As getting ε from the user is mandatory for Single-Linkage clustering, we simply identified the ε used for DBSCAN clustering with the range for the neighborhood we regard for the chain-detection. Thus we analyze using a different ε for DBSCAN than for the chain-detection in the following.

Increasing the ε parameter from ε_1 to ε_2 is ambiguous, because although increasing ε reduces the error effect of noise within the smaller ε_1 range, because more points of the chain are considered and thus the variation of the first principal components is increased, new noise within the ε_2 range but not within the ε_1 range may be considered, thus increasing the error. This effect can be seen by looking at the chains shown in Fig. 18 and compare those to Fig. 19. In Fig. 18 the bottom chain is shorter, because ε is smaller and the close range error gets too high. Increasing ε leads to a longer bottom chain, because the close range error effect is reduced and no noise within the increased ε_2 range is added, thus more points are detected as chain-point candidates. But the upper chain is not detected any more because there is too much noise added within the ε_2 range.

Setting ε too small results in too many chain-points detected and clustering those with $DBSCAN_{\varepsilon,minPts}$ can lead to chains within clusters, which is probably not desirable. Fig. 20 shows the results of a chain-detection on the London dataset from above with *allowedVariation* := 0.2, *chainDim* := 1 and different values for $\varepsilon = \{0.025, 0.05, 0.15, 0.2\}$, where chains are marked in red. The higher ε, the less chains are detected. Thus, giv-

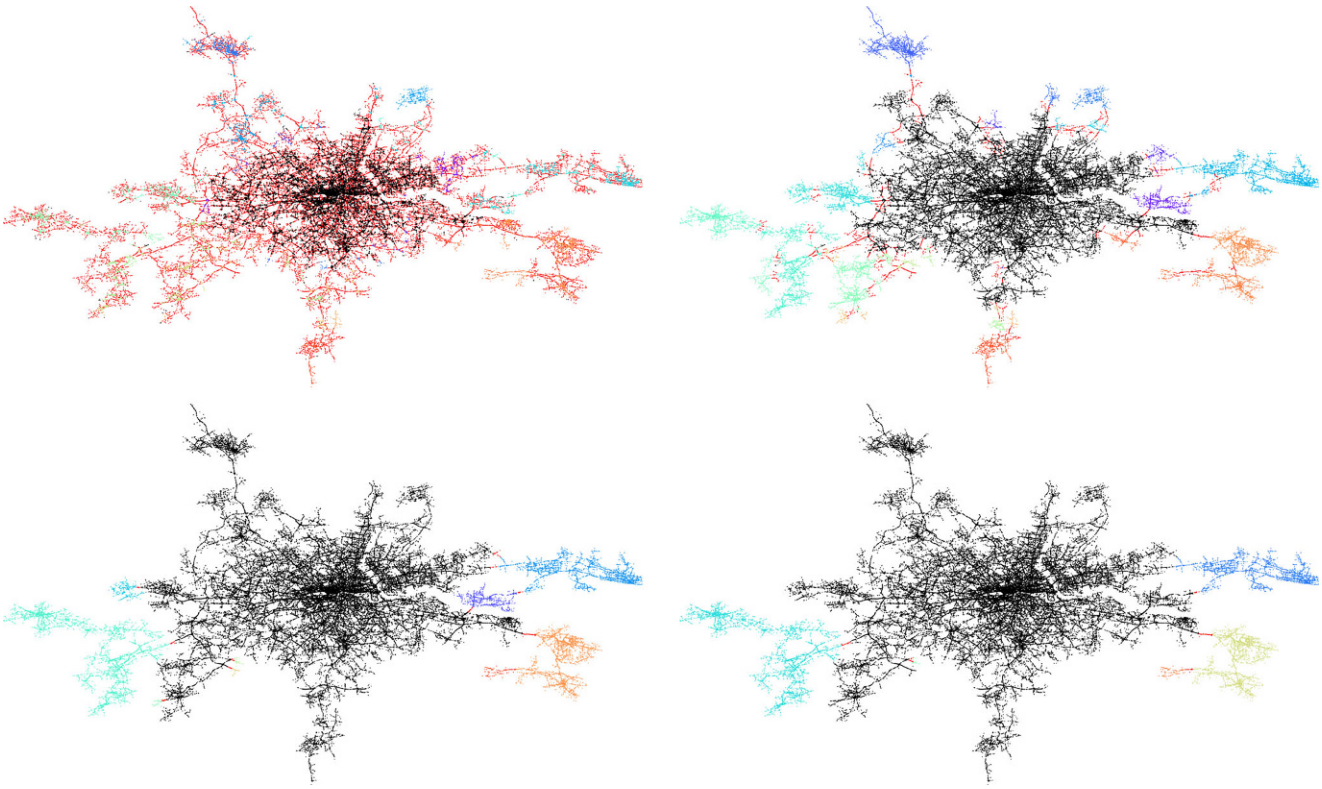

Fig. 20 Chaindetection in the London dataset for different values $\varepsilon = \{0.025, 0.05, 0.15, 0.2\}$

ing the user the possibility to not use the same ε as for DBSCAN can result in better results, but does not have to and is recommended only for users with expert knowledge regarding the data.

8 Conclusion

In conclusion we developed the first algorithm which solves the problem that some clustering algorithms, like, e. g., DBSCAN or Single-Linkage unintentionally detect only one cluster where several are connected by a chain or several noise points. We achieved that by recognizing chain points by analyzing the eigenvalues of the covariance matrix of their neighborhood. In our experiments with DBSCAN we applied the algorithm on a real world dataset containing traffic accidents, where it found the intentional chains and enabled DBSCAN to find the original, smaller clusters in the dataset, instead of aggregated ones. We developed the algorithm introduced in [6] to work also for Single-Linkage clustering, and showed its effectiveness on the benchmarking dataset cure-t2-4k. Our approach is not limited to DBSCAN and Single-Linkage, but could also be of use after executing other metric-based clustering algorithms which tend to aggregate clusters connected by chains. Nevertheless, the ε parameter which determines in which range of each point the distribution of points is regarded, would have to be determined. In future work, ε could be determined automatically, and other clustering algorithms should be investigated concerning the applicability of chain-detection. Also a use of ICA [8] instead of PCA could be interesting.

Acknowledgments This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

References

1. Balcan MF, Liang Y, Gupta P (2014) Robust hierarchical clustering. J Mach Learn Res 15(1):3831–3871
2. Birant D, Kut A (2007) St-dbscan: an algorithm for clustering spatial-temporal data. Data Knowl Eng 60(1):208–221
3. Day WH, Edelsbrunner H (1984) Efficient algorithms for agglomerative hierarchical clustering methods. J Classif 1(1):7–24
4. Ester M, Kriegel HP, Sander J, Xu X et al (1996a) A density-based algorithm for discovering clusters in large spatial databases with noise. KDD 96:226–231
5. Glasbey C (1987) Complete linkage as a multiple stopping rule for single linkage clustering. J Classif 4(1):103–109
6. Held J, Beer A, Seidl T (2019) Chain-detection for dbscan. In: BTW 2019–Workshopband
7. He Y, Tan H, Luo W, Mao H, Ma D, Feng S, Fan J (2011) Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In: 2011 IEEE 17th International Conference on Parallel and Distributed Systems. IEEE, 2011. pp 473–480
8. Hyvärinen A, Karhunen J, Oja E (2004) Independent component analysis vol 46. John Wiley & Sons, Hoboken
9. Jolliffe IT, Cadima J (2016) Principal component analysis: a review and recent developments. Philos Trans Royal Soc A 374(2065):20150202
10. Murtagh F (1983) A survey of recent advances in hierarchical clustering algorithms. Comput J 26(4):354–359
11. Ruiz C, Spiliopoulou M, Menasalvas E (2007) C-dbscan: Density-based clustering with constraints. In: International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing. Springer, Berlin, Heidelberg, 2007. pp 216–223
12. Sibson R (1973) Slink: an optimally efficient algorithm for the single-link cluster method. Comput J 16(1):30–34