Chain-detection for DBSCAN

Janis Held [1], Anna Beer [2], Thomas Seidl [2]

Abstract:

Chains connecting two or more different clusters are a well known problem of the probably most famous density-based clustering algorithm DBSCAN. 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 DBSCAN 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 by DBSCAN. Also, the desired straightness can be set by the user. We tested our efficient algorithm on a dataset containing traffic accidents in Great Britain and were able to detect chains emerging from streets between cities and villages, which led to clusters composed of diverse villages.

Keywords: DBSCAN, clustering, chain-detection, single link effect

1 Introduction

The human eye can easily detect areas of high density within a set of points. Derived from this human intuitive clustering method the basic idea behind density-based clustering is finding clusters by detecting areas of high density. The famous density-based algorithm DBSCAN [Es96] 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. 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.

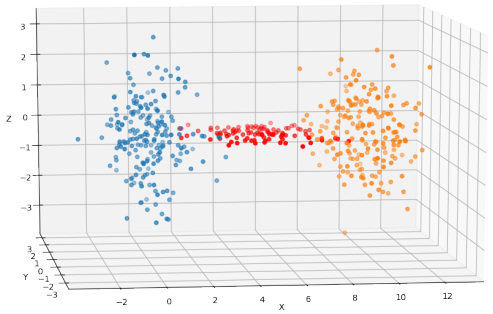


Fig. 1: The red points cause a density-connection between the intentional two clusters and thus form a chain.

[1] Ludwig- Maximilians- Universität München, Institut für Informatik, Oettingenstr. 67, 80538 München, Germany, J.Held@campus.lmu.de
[2] Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstr. 67, 80538 München, Germany, {beer, seidl}@dbs.ifi.lmu.de

doi:10.18420/btw2019-ws-18

While keeping the requirements of DBSCAN, like minimal domain knowledge to determine the input parameters, discovering clusters of arbitrary shape and good efficiency on large databases, we developed an algorithm which detects such chains in clusters found by DBSCAN. For that we use PCA (Principal Component Analysis) assuming that a chain has a lower dimensionality than the clusters it connects. Figure 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 by DBSCAN.

The paper is structured as follows: First, we introduce shortly the related work and basics we use in Section 2. In Sections 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. In Section 6 we conclude and give a brief idea of some future work.

2 Related Work and Basics

There are already many extensions of DBSCAN, e.g. ST-DBSCAN, an extension for clustering spatial-temporal data [BK07], MR-DBSCAN, which is an efficient parallel density-based clustering algorithm using map-reduce [He11], or C-DBSCAN: Density-based clustering with constraints [RSM07]. 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. In this section, we give the basics needed for the following sections, namely some details of DBSCAN and the Principal Component Analysis (PCA).

DBSCAN Density-based spatial clustering of applications with noise [Es96] 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 Figure 2 shows.

PCA (Principal Component Analysis) [JC16] 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 [JC16].

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

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

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 The Approach

Let $DBSCAN_{\epsilon,minPts}(X)$ be the clustering of DBSCAN with parameters ϵ and $minPts$ of some data X and C be a cluster found by DBSCAN 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 utilize the definition of neighborhood from DBSCAN and look for an algorithm that decides for each point if it lies within a subdimensional neighborhood. Additionally the algorithm has to fulfill some restraints: first of all it has to be rotation invariant as the direction of the chains does not matter. Secondly it has to be error resistant, as we want to be able to allow some bending of chains and apply it on a application with noise. The idea is to use the distribution of all points in the ϵ-range 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.

3.1 Chains

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.

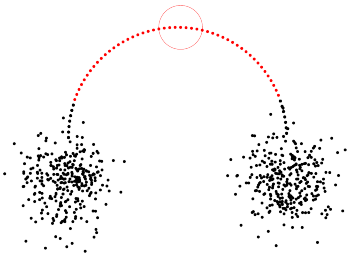


Fig. 4: The red dots may or may not be a chain, depending on the user. The red circle is one of the ϵ ranges.

Assume the user wants to detect one-dimensional chains in a two-dimensional data space and *DBSCAN* would not label the red dots in the following figures as noise, then Figure 2 shows a simple example of a chain. The red dots in Figure 3 are not considered a chain, because they do not form a connection between two clusters. The red dots in Figure 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

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 $ev_j, j \in \{i, ..., d\}$ of 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 two parameters

1. $chainDim \in \{1, ..., d-1\}$, which describes the dimensionality of chains the user wants to detect.
2. $allowedVariation \in [0, 1[$, which allows variation beyond the allowed dimensionality of the chain.

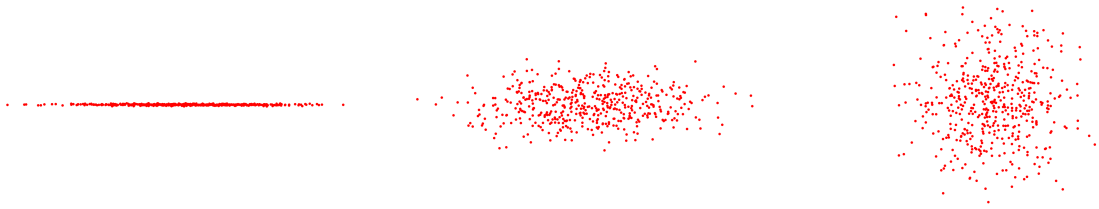
Like in Section 3.2, let $N = \{n_1, ..., n_m\}$ be the ϵ range of some point $p \in C$ and $\lambda_1, ..., \lambda_d$ the descending sorted normalized eigenvalues of the covariance matrix Θ corresponding to N. To calculate how good N lies within a $chainDim$ dimensional subspace, one calculates the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

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

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

3.4 Fuzziness of Chains

In Figure 5 examples for various values of normed errors are given for a two dimensional data space with $chainDim = 1$. $\bar{e}$ 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 Figure 5c the error is close to 1 since the points are almost perfectly distributed in all directions.



(a) $\bar{e} \approx 0.0002$ (b) $\bar{e} \approx 0.1563$ (c) $\bar{e} \approx 0.9997$

Fig. 5: Various degrees of fuzziness dependent on the normed error $\bar{e}$

Let us have a look at some synthetic example data. In Figure 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 Figure 6 results in the shape-based chain-point candidates seen in Figure 7.

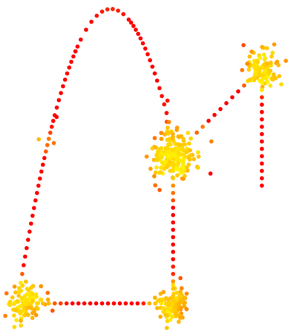


Fig. 6: Example data: Each point is colored by the normed error $\bar{e}$ derived from its ϵ range. Yellow means the error is close to 1 and red means it is close to 0.

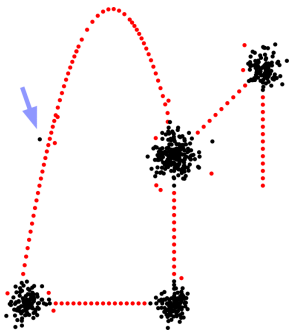


Fig. 7: Example data: With $allowedVariation = 0.2$ the red points are selected as shape-based chain-point candidates. The arrow highlights an outlier.

3.5 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 $DBSCAN_{eps,minPts}$ 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 Figure 7. These points are now added to the set of chain-point candidates.

Clustering the set of chain-point candidates by $DBSCAN_{eps,minPts}$ 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 $DBSCAN_{eps,minPts}(R)$. 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 DBSCAN with metric $dist(\cdot,\cdot)$ and parameters ϵ and $minPts$. $chainDim$ and $allowedVariation$ are the parameters of chain detection. $RangeQuery(C, dist, p, \epsilon)$ returns the set $\{q \in C | dist(p,q) \leq \epsilon\}$. For the sake of simplicity assume the result of DBSCAN contains the property "Noise", which is the set of points marked as noise and the property "Clusters", which is the set of clusters. Algorithm 1 recapitulates our complete approach. For a full implementation with example code 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 $O(n(n + n * d^2 + d^3))$. The DBSCANs on a subset of the cluster each have the worst case run time complexity of $O(n^2)$. 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 $d << n$ one can simplify the runtime complexity to $O(n^2)$.

Algorithm 1 Chain-detection

procedure VALIDATECHAINCANDIDATE($Chain, R, DB_R, dist, \epsilon$)
  $clusterFound \leftarrow$ null
  for $c \in Chain$ **do**
    for $p \in RangeQuery(R, dist, c, \epsilon)$ **do**
      if $clusterFound ==$ null **then**
        $clusterFound \leftarrow DB_R$.labelFor($p$)
      else
        if $clusterFound \neq DB_R$.labelFor(p) **then**
          return True
  return False
procedure CHAIN-DETECTION($C, dist, \epsilon, minPts, chainDim, allowedVariation$)
  $d \leftarrow dim(C)$ ▷ The dimensionality of the data
  $C_c \leftarrow \{\}$ ▷ The set of chain-points
  for $p \in C$ **do** ▷ Find all chain-point candidates
    $N \leftarrow RangeQuery(C, dist, p, \epsilon)$
    $EV \leftarrow EigenValues(CovarianceMatrix(N))$
    $EV \leftarrow EV/EV$.sum() ▷ Norm eigenvalues
    $EV \leftarrow EV$.sorted(descending=TRUE) ▷ Sort eigenvalues descending
    $e \leftarrow EV$.sum(start=$d - chaindim + 1$) ▷ Calculate error
    $e \leftarrow e * (d/(d - chainDim))$ ▷ Norm error
    if $e \leq allowedVariation$ **then** ▷ Compare error with parameter
      $C_c \leftarrow C_c \cup \{p\}$ ▷ Add p to the set of chain-points
  if $|C_c| == 0$ **then return** $\{\}$
  $R \leftarrow C \setminus C_c$ ▷ The set of remaining points
  $DB_R \leftarrow DBSCAN(R, dist, \epsilon, minPts)$ ▷ Cluster the remaining points
  $C_c \leftarrow C_c \cup DB_R$.Noise ▷ Add noise to the set of chain-points
  $DB_{C_c} \leftarrow DBSCAN(C_c, dist, \epsilon, minPts)$ ▷ Cluster chain-points
  if $|DB_{C_c}$.clusters$| == 0$ **then return** $\{\}$ ▷ No chain-candidate found
  $R \leftarrow C \setminus \cup DB_{C_c}$.Clusters ▷ Update the set of remaining points
  $DB_R \leftarrow DBSCAN(R, dist, \epsilon, minPts)$ ▷ Cluster the remaining points
  if $|DB_R$.clusters$| \leq 1$ **then return** $\{\}$ ▷ No chain-candidate can be validated
  $Chains \leftarrow []$ ▷ The list of validated chains
  for $V \in DB_{C_c}$.Clusters **do** ▷ Validate each chain-candidate
    if ValidateChaincandidate($V, R, DB_R, dist, \epsilon$) **then**
      $Chains$.append(V)
  return $Chains$

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. It was downloaded on February the 27th 2018 from `https://www.kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales/data` and clustered by DBSCAN with parameters $\epsilon := 0.01$ and $minPts := 15$. These parameters were obtained by trial and error while clustering the area of roughly 100km in each direction around London's center with the goal to obtain a cluster which contains chains of traffic accidents.

Traffic accidents in London The chain-detection will be demonstrated on the cluster found at London city, see Figure 8. The results obtained by DBSCAN 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. Figure 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.

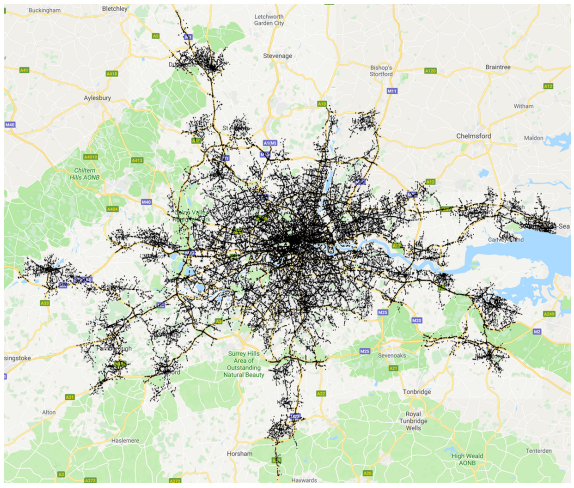


Fig. 8: The cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. The dots are stretched to fit the underlying map.

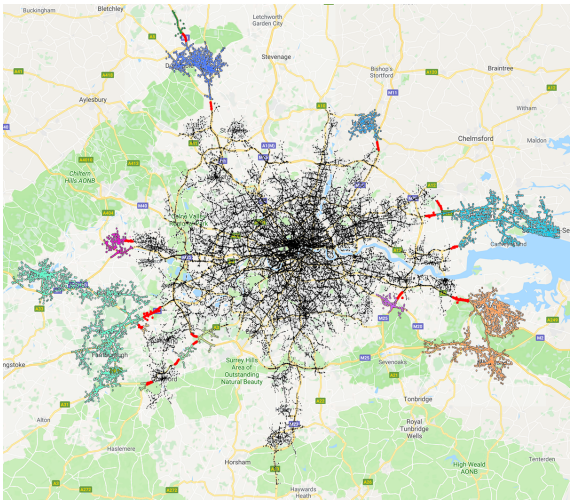


Fig. 9: Chain-detection applied on the cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. Chains are marked red.

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 Figure 10. Let us apply the chain-detection algorithm with parameters $chainDim := 1$ and $allowedVariation := 0.2$, for the same reasons as in the previous example. In Figure 11 we can see how the traffic accident clusters are now well divided, one cluster in Liverpool and one in Manchester.

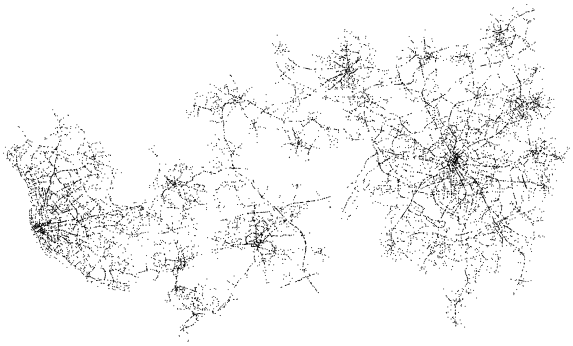


Fig. 10: The cluster of traffic accidents at Liverpool and Manchester.

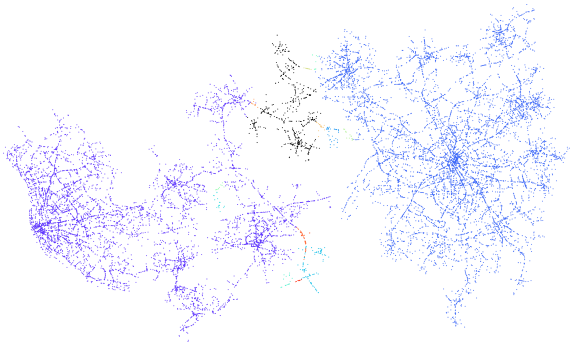


Fig. 11: Result of the chain-detection algorithm applied on the traffic accidents in Liverpool and Manchester.

6 Conclusion

In conclusion we developed the first algorithm which solves the problem that DBSCAN unintentionally detects 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 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. Our approach is not limited to DBSCAN, but could also be of use after executing other 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. We plan to examine further areas of application and experiments in future work.

References

[BK07] Birant, D.; Kut, A.: ST-DBSCAN: An algorithm for clustering spatial-temporal data. Data & Knowledge Engineering 60/1, pp. 208–221, 2007.

[Es96] Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, 1996, URL: https://ocs.aaai.org/Papers/KDD/1996/KDD96-037.pdf, visited on: 01/11/2019.