LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR DATENBANKSYSTEME
UND DATA MINING

Bachelor Thesis
in Computer Science

Chain-detection for DBSCAN

Janis Held

Supervisor:	Prof. Dr. Thomas Seidl
Advisor:	Anna Beer
Submission Date:	14.03.2018

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.
This paper was not previously presented to another examination board and has not been published.

Munich, 14.03.2018

.....
Janis Held

Abstract

Density based clustering methods are able to identify areas with dense clusters of any shape and size. DBSCAN is one of the basic methods in this group, but clustering with DBSCAN has a major drawback if the clusters in question are connected by small chains. Imagine two separated clouds of points connected by a small chain of points. Intuitively these two clouds are different clusters, but DBSCAN may detect these as a single huge one, because the algorithm expands the cluster along these chains. This is sometimes called the single link effect. In this thesis an algorithm is proposed to detect these chains in multidimensional data spaces.

Contents

Chapter 1

Introduction

Partitioning methods (K-means, PAM clustering) and hierarchical clustering work well only for compact, convex and well separated clusters. Moreover, they are severely affected by noise and outliers in the data. Whereas density-based clustering is a methodology for finding clusters of arbitrary shape. 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. DBSCAN is looking for points with minimal density and starts expanding a cluster around these points, as described in chapter 2.0.3. As long as the clusters are well separated this works very well, but if there is a relatively small chain, compared to the clusters, between those intuitive clusters, DBSCAN will continue expanding the cluster along these chains resulting in a single huge cluster instead of the intuitive ones.

With respect to the three requirements of DBSCAN [3] namely

1. Minimal requirements of domain knowledge to determine the input parameters

2. Discovery of clusters with arbitrary shape

3. Good efficiency on large databases

the objective is to find an algorithm, which detects chains in clusters found by DBSCAN.

Chapter 2

Related Work

There are already many extensions of DBSCAN, e.g. ST-DBSCAN: An extension for clustering spatial-temporal data [2], MR-DBSCAN an efficient parallel density-based clustering algorithm using map-reduce [4] or C-DBSCAN: Density-based clustering with constraints [6].

2.0.1 Intrinsic dimensionality

The chain-detection algorithm utilizes the intrinsic dimensionality of some data points to determine if they could form a chain. There are several estimators for intrinsic dimensionality [1], but in this thesis the idea behind Principal Component Analysis is used to detect low intrinsic dimensionality.

2.0.2 Principal Component Analysis

The Principal Component Analysis (PCA) transforms the data 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 calculates these components and the variance explained by them, more at chapter 3.2.1. One can utilize this to explain how good some data lies within a lower dimensional subspace.

2.0.3 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a density based clustering algorithm that clusters (groups) points that are closely packed together and marks outliers that lie in low-density regions.

2.0.3.1 Parameters

DBSCAN relies on two input parameters ϵ and $minPts$. The parameter ϵ is required to define the range of neighbourhood in question and $minPts$ is the required number of neighbours inside that ϵ neighbourhood to form a cluster.

2.0.3.2 Definitions

Consider a set S of points in some metric space. Let d denote the metric.

Definition 1. The ϵ range or ϵ neighbourhood of a point $s \in S$ is defined as the set $\{p \in S | d(p, s) \leq \epsilon\}$. All points in the ϵ range of p are called ϵ-neighbours of p.

Definition 2. A point $p \in S$ is considered a **core point** if at least $minPts$ points are within the ϵ range of p.

Definition 3. Let $p \in S$ be a core-point. All points within the ϵ range of p are considered **directly reachable** from p.

Definition 4. A point $p \in S$ is considered **reachable** from a point $q \in S$ if there exists a path $p_1, ..., p_n$ with $p_1 = p$ and $p_n = q$, where p_i is directly reachable from p_{i+1} for all $i = 1, ..., n - 1$.

Definition 5. A subset $C \subseteq S$ is considered a **cluster** if there is at least one core-point $p \in C$ and $C = \{q \in S \mid q \text{ reachable from } p\}$.

Definition 6. A point $p \in S$ is considered **noise** if there is no cluster C with $p \in C$.

2.0.3.3 Algorithm

Let DB be the set of points to cluster and $dist(\cdot, \cdot)$ be a metric on DB. ϵ and $minPts$ are the parameters for DBSCAN. Let $RangeQuery(DB, dist, p, \epsilon)$ be the ϵ range of p. $p.label$ is initially undefined for all points in DB.

Algorithm 1 DBSCAN

```
procedure DBSCAN(DB, dist, ε, minPts)
    clusterID ← 0                                   ▷ The current cluster ID
    for p in DB do
        if p.label ≠ undefined then
            continue        ▷ This point was previously processed in inner loop
        N ← RangeQuery(DB, dist, p, ε)
        if |N| < minPts then
            p.label ← Noise
            continue
        C ← C + 1                                   ▷ Increase cluster label
        p.label ← C          ▷ Set the label of p to the current cluster label
        S ← N \ {p}            ▷ Create the set S with all neighbours of p
        while |S| > 0 do                                    ▷ Expand cluster
            q ← S.first()                        ▷ Take the first element of S
            if label(q) = Noise then
                label(q) ← C
            if label(q) ≠ undefined then
                continue        ▷ q was either noise or previously processed
            label(q) ← C
            N ← RangeQuery(DB, dist, q, ε)
            if |N| ≥ minPts then
                S ← S ∪ N                       ▷ Continue expanding around q
            S ← S \ {q}                                   ▷ Remove q from S
    return DB
```

After DBSCAN terminates, for all p in DB $p.label$ is either Noise or a number, which identifies its cluster.

Chapter 3

Chain-detection

After DBSCAN found a cluster the chain-detection algorithm will determine if this cluster contains a chain and which points belong to a chain. Throughout this chapter let C be the cluster found by DBSCAN with fixed ϵ and $minPts$ parameters in some data space Ω.

3.1 The Approach

Definition 7. Let $DBSCAN_{\epsilon,minPts}(X)$ be the clustering of DBSCAN with parameters ϵ and $minPts$ of some data X.

Intuitively one could think that a point within a chain has significantly less ϵ-neighbours than the average point in C. In figure 3.1 one can clearly see the chain marked red, because each point within the chain has significant less ϵ-neighbours than the average cluster-point.

Figure 3.1: Example of a chain between two clusters. Each point is coloured by the relative cardinality of its ϵ range, where red means a low cardinality.

But this approach fails if the chain is too dense. Furthermore, the borders of clusters tend to be detected as chains. In figure 3.2 the chain can not be

detected this way, because each point in the chain has a much higher number of ϵ-neighbours than the average cluster-point.

Figure 3.2: Example of a very dense chain between two clusters. Each point is coloured by the relative cardinality of its ϵ range, where yellow means a high cardinality.

Because of the problem shown above, defining a chain-point by looking at the shape of its ϵ range and not its cardinality is a much better approach.

Definition 8. A point $p \in C$ is considered a **shape-based chain-point candidate** if there exists a subspace S with a lower dimensionality than Ω, such that all points of the ϵ range of p lie close to S.

Note that "lower dimensionality" and the word "close" will become parameters for the chain-detection algorithm.

Definition 9. Let S be the set of all shape-based chain-point candidates in C. A point $p \in C$ is considered a **chain-point candidate** if $p \in S$ or p is marked as noise in $DBSCAN_{\epsilon,minPts}(C \setminus S)$.

Definition 10. Let C_c be the set of all chain-point candidates. Let $C_i, i \in \{1, ..., n\}$ be the clusters found by $DBSCAN_{\epsilon,minPts}(C_c)$. Each cluster C_i is considered a **chain-candidate**.

Note that all chain-point candidates which were marked as noise are not part of a chain-candidate, because a chain has to be at least big and dense enough to form a cluster itself.

Definition 11. Let $C_i, i \in \{1, ..., n\}$ be the chain-candidates. A chain-candidate C_i is considered a **chain** if there are two points in C_i, where the union of their ϵ range contains at least two points from different clusters of $DBSCAN_{\epsilon,minPts}(C \setminus \cup_{i \in I} C_i)$.

In other words a chain-candidate is considered a chain, if it connects two clusters of the non-chain points $C \setminus \cup_{i \in I} C_i$.

3.1.1 Examples

Assume the user wants to detect one-dimensional chains in a two-dimensional data space and $DBSCAN_{\epsilon,minPts}$ would not label the red dots in the following figures as noise, then figure 3.3 shows a simple example of a chain. The red dots in figure 3.4 are not considered a chain, because they do not form a connection between two clusters. The red dots in figure 3.5 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 depend on the user if he wants to detect those as a chain.

Figure 3.3: Example of a chain. The red dots form a chain.

Figure 3.4: Example of what is not considered a chain.

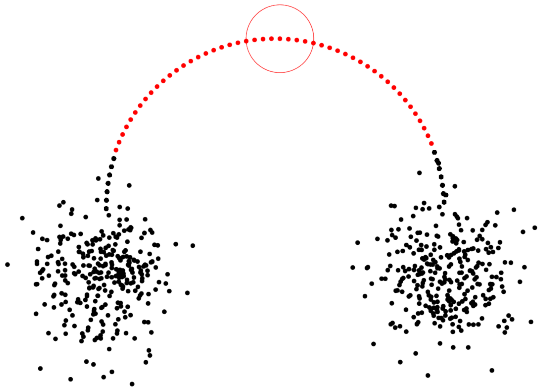

Figure 3.5: The red dots may or may not be a chain, depending on the user. The red circle is one of the ϵ ranges.

3.2 Detecting chain-point candidates

For each point in C the objective is to determine if this point may belong to a chain. 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 Ω. To be more precise, PCA is utilized to calculate the explained variation of those ϵ-neighbours of p which do not fit inside this subspace.

3.2.1 Theory

PCA can be done by eigenvalue decomposition of the data covariance matrix, usually after mean centring the data matrix for each dimension. Then the eigenvectors of the covariance matrix form a orthogonal basis and **each eigenvalue describes how much variance is explained by its corresponding eigenvector** [5]. In figure 3.6 the eigenvectors of the covariance matrix, scaled by the square root of the corresponding eigenvalue, are shown.

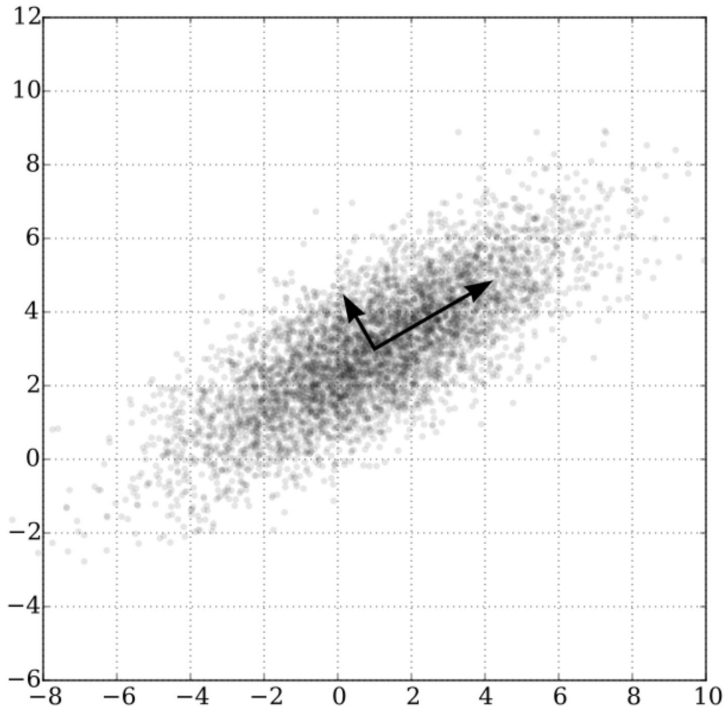

Figure 3.6: A multivariate Gaussian distribution centred at (1,3) with a standard deviation of 3 in roughly the (0.866, 0.5) direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean. Downloaded from https://commons.wikimedia.org/wiki/File:GaussianScatterPCA.svg.

Let d be the dimensionality of Ω and $N = \{n_1, ..., n_m\}$ the ϵ range of some point $p \in C$. The data matrix is defined as

$$X := \begin{bmatrix} n_{11} & n_{12} & ... & n_{1d} \\ n_{21} & n_{22} & ... & n_{2d} \\ ... & & & \\ n_{m1} & n_{m2} & ... & n_{md} \end{bmatrix} \tag{3.1}$$

The mean of each column of the data matrix is defined as

$$m_i := \frac{\sum_{j=1}^{m} n_{ij}}{m}, \quad i = 1...d. \tag{3.2}$$

One can now calculate the covariance matrix Θ with

$$\Theta_{ij} = \frac{\sum_{k=1}^{m}(n_{ki} - m_i)(n_{kj} - m_j)}{m}, \quad i = 1, .., d, \quad j = 1, .., d. \tag{3.3}$$

Note that the covariance matrix is symmetric and positive semi-definite, thus its eigenvalues are non negative. Finally the eigenvalues are normalized

$$\bar{\lambda_i} := \lambda_i / \sum_{j=1}^{d} \lambda_j, \tag{3.4}$$

such that the sum of all normalized eigenvalues equals to 1.

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 entirely 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 entirely 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

$$\lambda_j = 0, \forall j \in \{i, ..., d\}. \tag{3.5}$$

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 $ev_j, j \in \{1, ..., i-1\}$.

□

3.2.2 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 chapter 3.2.1, 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

$$e := \sum_{i=chainDim+1}^{d} \lambda_i. \tag{3.6}$$

The sum starts with $chainDim+1$, because only the explained variation of the $d - chainDim$ least significant principal components are summed up. 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

$$e \in [0, (d - chainDim)/d] \tag{3.7}$$

To make the user-input independent of the dimensionality of Ω and $chainDim$, one normalizes the error by

$$\bar{e} := e * \frac{d}{d - chainDim} \in [0, 1]. \tag{3.8}$$

Now, p is a chain-point candidate if

$$\bar{e} \leq allowedVariation. \tag{3.9}$$

3.2.3 Examples

In figure 3.7, 3.8, 3.9, 3.10, 3.11 and 3.12 some normed error examples are given for a two dimensional data space with $chainDim = 1$, so $\bar{e}$ 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 3.12 the error is close to 1 because the points are almost perfectly distributed in all directions.

Figure 3.7: Normed error example with $\bar{e} \approx 0.0002$

Figure 3.8: Normed error example with $\bar{e} \approx 0.0054$

Figure 3.9: Normed error example with $\bar{e} \approx 0.023$

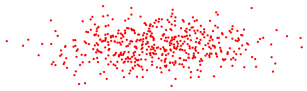

Figure 3.10: Normed error example with $\bar{e} \approx 0.1563$

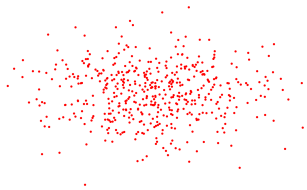

Figure 3.11: Normed error example with $\bar{e} \approx 0.3922$

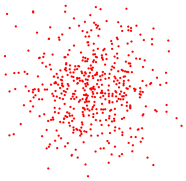

Figure 3.12: Normed error example with $\bar{e} \approx 0.9997$

Note that the normed error is rotation invariant, so rotating figure 3.7 results in the same normed error, as seen in figure 3.13.

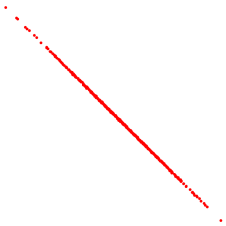

Figure 3.13: Rotated error example with $\bar{e} \approx 0.0002$

In a 3 dimensional data space setting $chainDim := 2$ will detect two dimensional planes as chains, see figure 3.14. With the $allowedVariation$ parameter

those planes can even be detected if they are a bit bent or lie within a noisy environment. Setting $chainDim := 1$ will only detect 1 dimensional threads as chains, see figure 3.15. The parameter $allowedVariation$ will allow the algorithm to detect those threads as chains even though they are not perfectly linear.

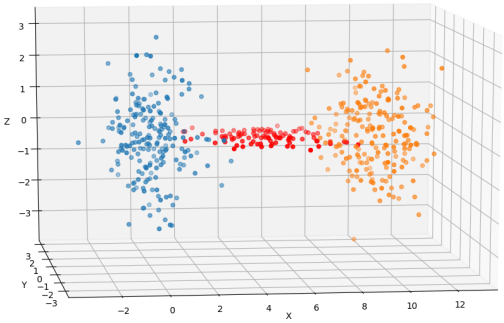

Figure 3.14: Example of a two dimensional chain (marked red) parallel to the XY plane between three-dimensional clusters.

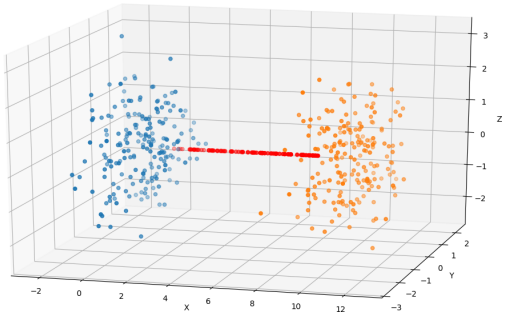

Figure 3.15: Example of a one dimensional chain (marked red) parallel to the X axis between three-dimensional clusters.

Let us have a look at some example data. In figure 3.16 the points are coloured 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 3.16 results in the chain-point candidates seen in figure 3.17.

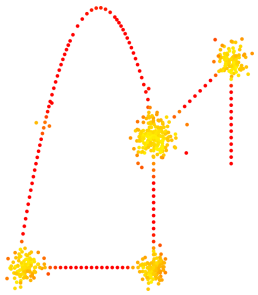

Figure 3.16: Example data: Each point is coloured by the normed error $\bar{e}$ derived from its ϵ range. Yellow means the error is close to 1 and red means it is close to 0.

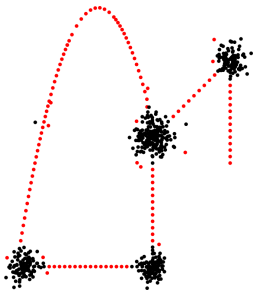

Figure 3.17: Example data: With $allowedVariation = 0.2$ the red points are selected as chain-point candidates.

3.3 Refining the set of chain-point candidates

The set of chain-point candidates has to be refined, because it may contain noise when clustered by $DBSCAN_{eps,minPts}$ and chains have to be big and dense enough to form a cluster itself. On the other hand, the set of non-candidates also may have noise when clustered by $DBSCAN_{eps,minPts}$. These points did not get selected as chain-point candidates, but are also not part of a cluster of non-candidates, indicating that they also might be part of a chain.

3.3.1 Adding candidates by clustering remaining points

Let $C_{\bar{e}}$ be the set of chain-point candidates found with the normed error $\bar{e}$ estimation of chapter 3.2. After clustering the remaining points $C \setminus C_{\bar{e}}$ by $DBSCAN_{eps,minPts}$ all points marked as noise are now added to the set of chain-point candidates.

Let us stick to the example data from figure 3.17. Clustering all points except the chain-point candidates results in the clusters seen in figure 3.18. All outliers (marked black) are now added to the set of chain-point candidates.

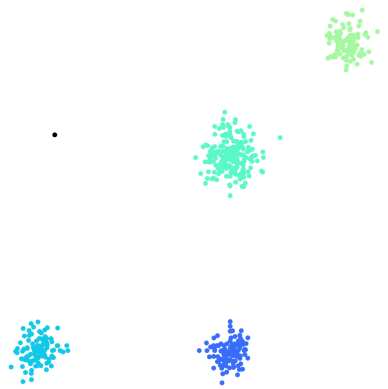

Figure 3.18: Example data: After clustering all points except the chain-point candidates all outliers are also considered as chain-point candidates, see the black dot in the upper left.

3.3.2 Removing candidates by clustering candidates

Let $C_c \subset C$ be the set of all chain-point candidates after the refinement step of chapter 3.3.1. Clustering C_c by $DBSCAN_{eps,minPts}$ results in clusters of chain-point candidates, which are validated later, and noise. All points classified as noise can not form a chain, because they do not lie within a dense enough area

of chain-point candidates to form a chain. All clusters found are considered **chain-candidates**.

Back to the example data from figure 3.17. Clustering the set C_c can be seen in figure 3.19. These clusters are now considered chain-candidates. Combining the chain-candidates with the remaining points can be seen in figure 3.20.

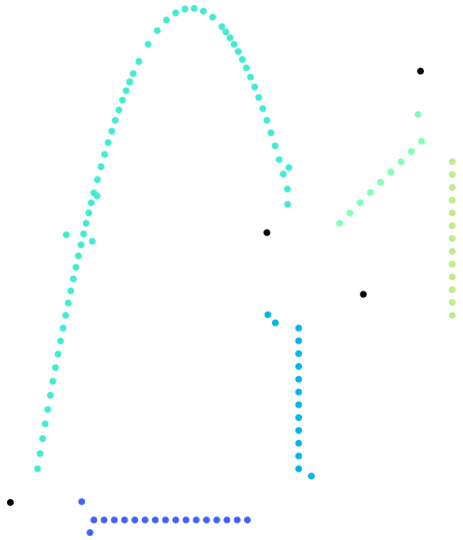

Figure 3.19: Example data: Chain-point candidates clustered.

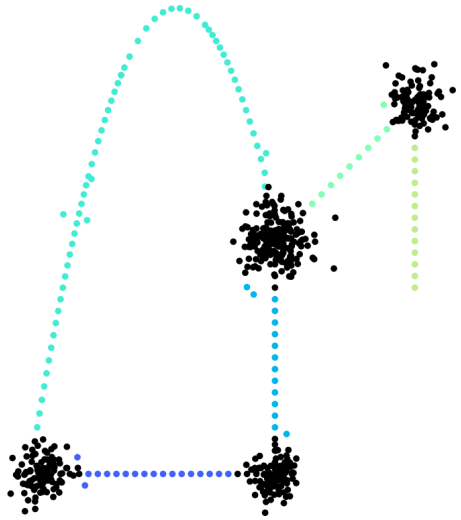

Figure 3.20: Example data: The coloured dots (not the black ones) are the clusters of the final set of chain-point candidates. Those clusters are called chain-candidates.

3.4 Validating chain-candidates

The objective is to verify each chain-candidate by checking if it indeed connects two clusters of the remaining points. Let $C_{ci}, i \in I$ be the chain-candidates. Let $R := C \setminus \cup_{i \in I} C_{ci}$ be the set of the remaining points. Note that R contains those chain-point candidates, which were marked as noise by clustering all chain-point candidates. Let DB_R be $DBSCAN_{eps,minPts}(R)$. One can now validate a chain-candidate C_{ci} by testing if it is between two clusters of R. For each point $p \in C_{ci}$, let $range_\epsilon(p)$ be the ϵ range of p. For each $r \in R \cap range_\epsilon(p)$, 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.

Looking at the example data from figure 3.20, one can now validate each chain-candidate. The first step is to calculate $DBSCAN_{eps,minPts}(R)$. In fig-

ures 3.21, 3.22, 3.23 and 3.24 the red dots do form a chain, because the red dots at the beginning and the end of the chain each have a different cluster in their ϵ range. In figure 3.25 however, the red dots are not considered a chain, because there is only one cluster in the ϵ range of all red dots. Indeed, looking at figure 3.20 one can see that the chain candidate on the right does not form a connection between two black clusters.

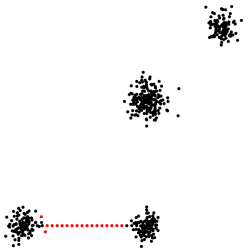

Figure 3.21: Example of a chain (red dots).

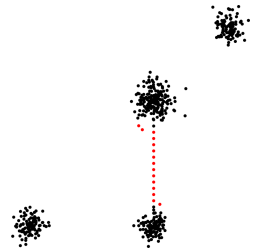

Figure 3.22: Example of a chain (red dots).

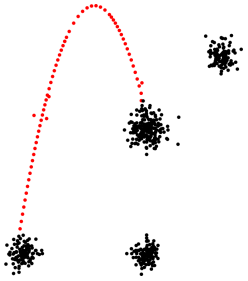

Figure 3.23: Example of a chain (red dots).

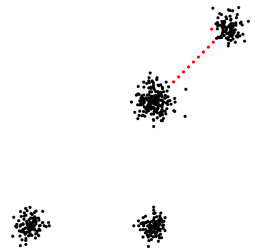

Figure 3.24: Example of a chain (red dots).

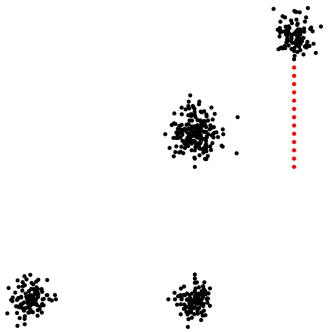

Figure 3.25: Example of what is not considered a chain (red dots).

Let $C_{ci}, i \in V \subseteq I$ be the set of all chains. To get the final clustering without chains, simply cluster the non-chain points $C \setminus \cup_{i \in V} C_{ci}$.

Looking at the example data, the final clustering of the non-chain points can be seen in figure 3.26. Adding the validated chains and marking them red results in figure 3.27.

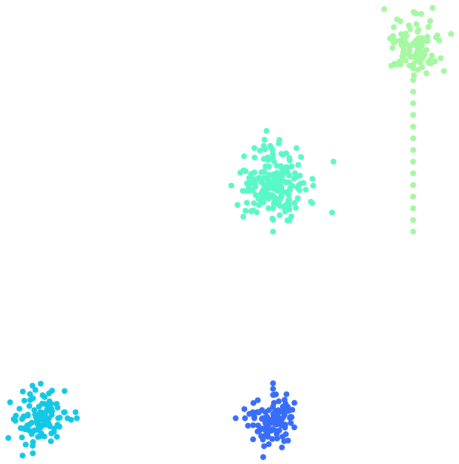

Figure 3.26: The final clustering of C without chains.

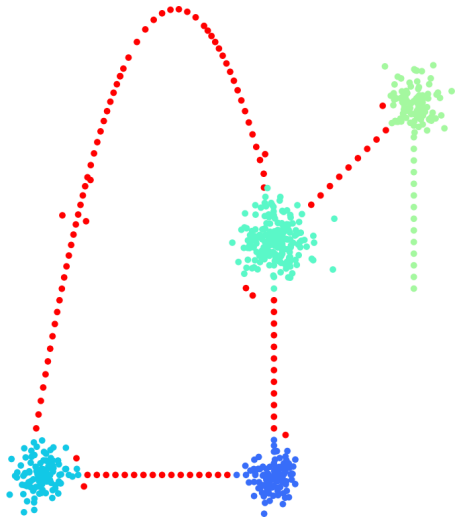

Figure 3.27: The output of the algorithm with chains marked in red.

3.5 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 2 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** $\{\}$
  $R \leftarrow C \setminus C_c$ ▷ Refine the set of chain-point candidates
  $DB_R \leftarrow DBSCAN(R, dist, \epsilon, minPts)$ ▷ 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
  $C_c \leftarrow C_c \setminus DB_{C_c}$.Noise ▷ Remove noise from the set of chain-points
  $R \leftarrow C \setminus C_c$ ▷ 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-c. can be validated
  for $V \in D$.Clusters **do** ▷ Validate each chain-candidate
    if NOT ValidateChaincandidate($V, R, DB_R, dist, \epsilon$) **then**
      $C_c \leftarrow C_c \setminus V$
  return C_c

3.6 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-neighbourhood, 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 $O(n^2)$.

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.

3.7 Limitation

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 figure 3.28. To counter this limitation, one could introduce a third parameter for the chain-detection to increase the range while calculating the error, see chapter 4.3.

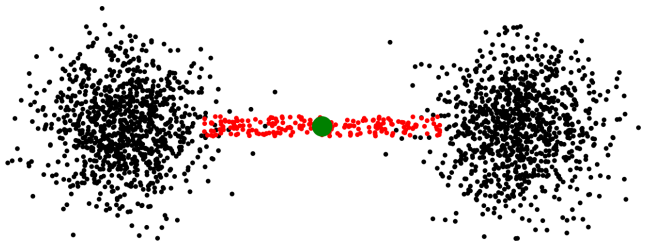

Figure 3.28: The red dots will not be detected as a chain, because the ϵ range (marked as a green circle) is too small to detect the chain.

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

Chapter 4

Experiments

The dataset on which the experiments were done consists of all traffic accident locations in Great Britain from the years 2014 - 2016. Each accident has a longitude and latitude value, describing its location. Note that all the contained accident data comes from police reports, so this data does not include minor or not reported incidents. The dataset was downloaded on February the 27th 2018 from https://www.kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales/data. The license for this dataset is the Open Government Licence used by all data on data.gov.uk (http://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/). The raw datasets are available from the UK Department of Transport website https://www.dft.gov.uk/traffic-counts/download.php.

The dataset was 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. The total result of the clustering can be see in figure 4.1.

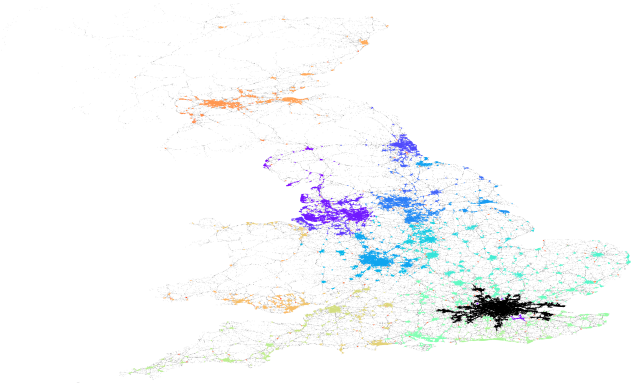

Figure 4.1: DBSCAN clustering of traffic accidents in Great Britian

4.1 Traffic accidents in London

In this chapter the chain-detection will be demonstrated on the cluster found at London city, see figure 4.2.

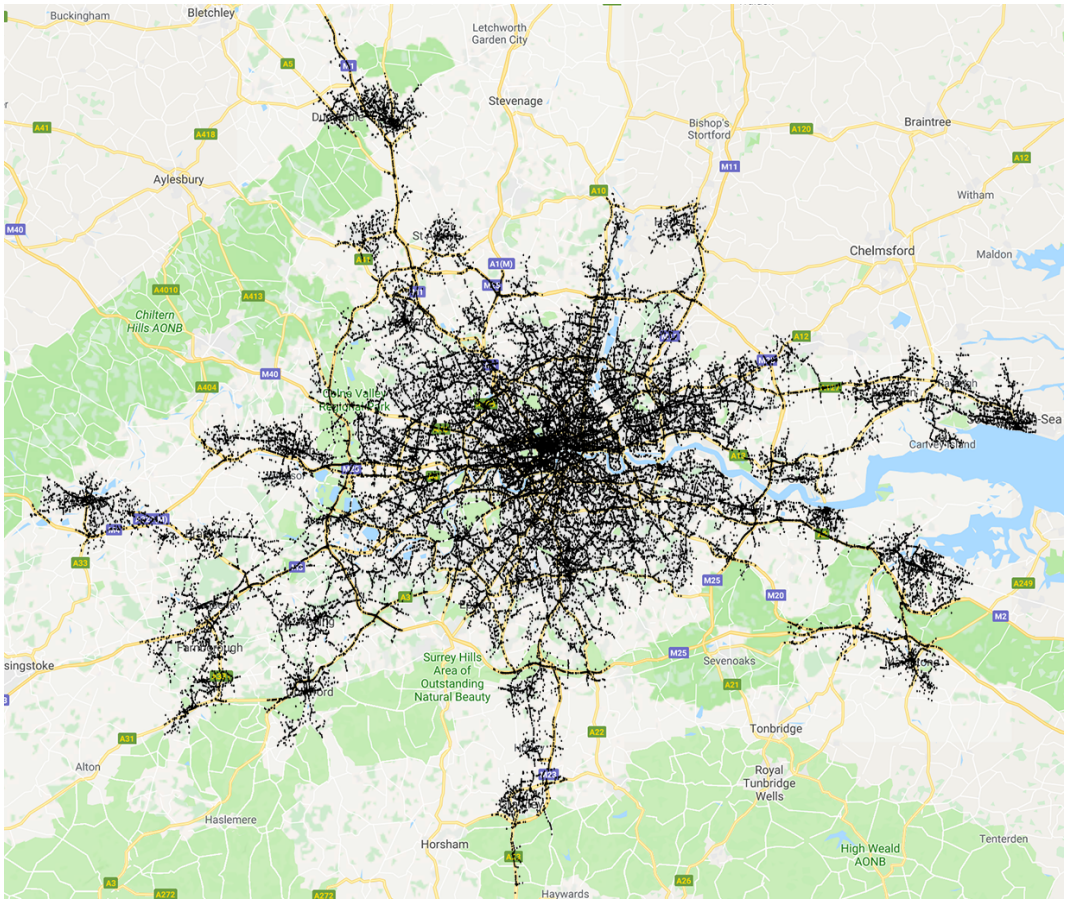

Figure 4.2: The cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. The dots are stretched to fit the underlying map.

This may not be 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. Because the highways are not perfectly linear and surrounded by noise, one wants to allow some error and set the $allowedVariation$ parameter to 0.2.

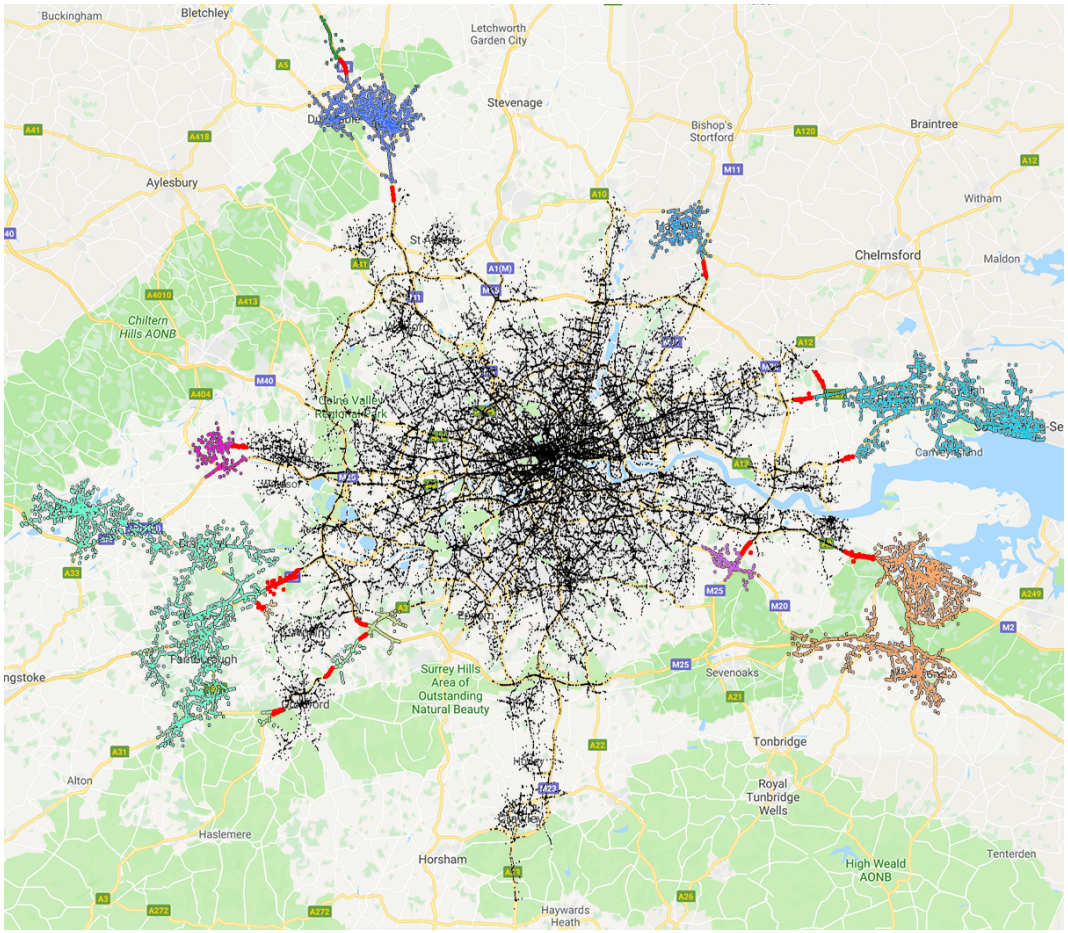

Figure 4.3: Chain-detection applied on the cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. Chains are marked red. There is a small black border added to non-black cluster points to make them more visible. The dots are stretched to fit the underlying map.

In figure 4.3 most of the suburban areas are now separated from the main cluster of London city. Almost all chains are found on highways, as seen in figure 4.4, where only the chains are drawn on top of London's map.

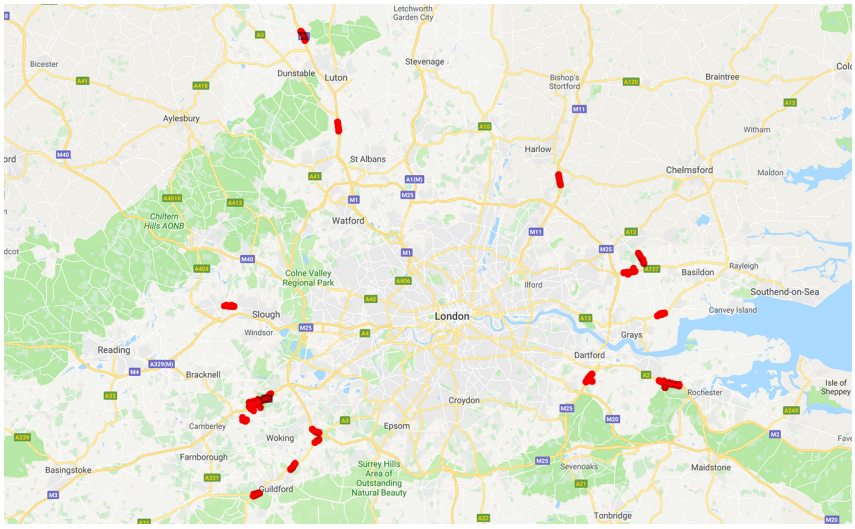

Figure 4.4: Detected chains are highlighted red on top of London's map.

4.2 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 figure 4.5.

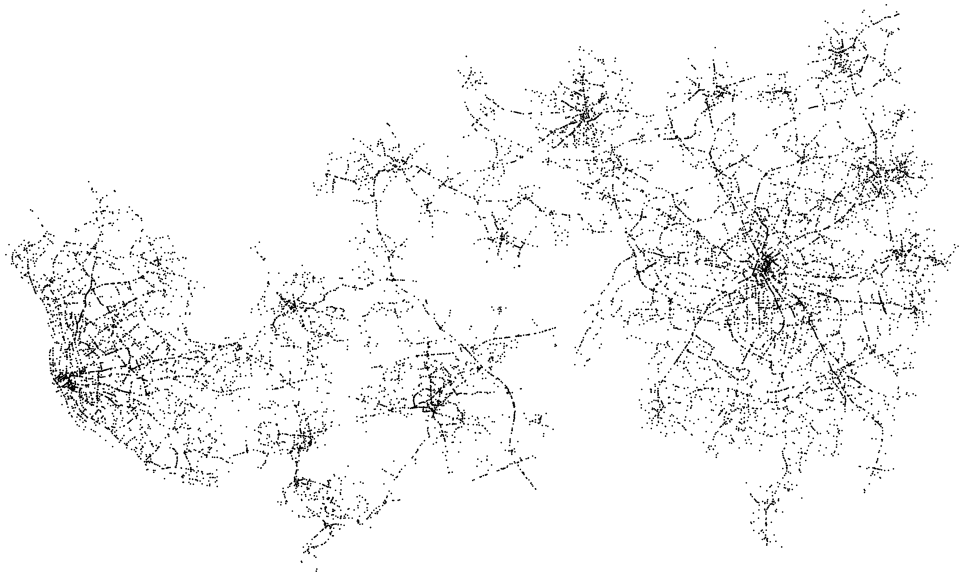

Figure 4.5: The cluster of traffic accidents at Liverpool and Manchester.

Let us apply the chain-detection algorithm with parameters $chainDim := 1$ and $allowedVariation := 0.2$, because highways between those cities are not perfectly linear. In figure 4.6 the traffic accident clusters are now well divided, one cluster in Liverpool and one in Manchester.

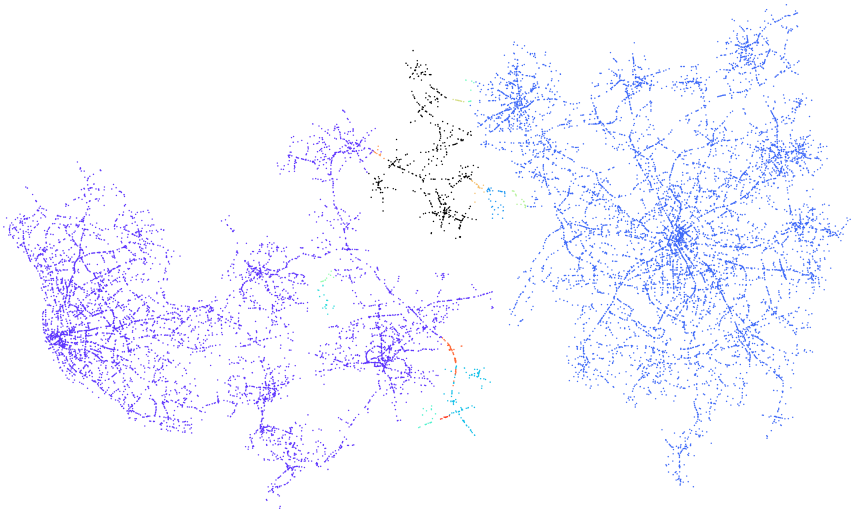

Figure 4.6: Result of the chain-detection algorithm applied on the traffic accidents in Liverpool and Manchester.

4.3 Parameters

There are many papers about the parameter estimation of ϵ and $minPts$ for the DBSCAN algorithm, e.g. the adaptive methods for determining DBSCAN parameters [7]. The $chainDim$ parameter is easily set if the user knows the dimensionality of the chains he wants to detect. If the data space consists of n dimensions, then the most common settings for $chainDim$ are probably $n-1$ to detect hyperplanes or 1 for one-dimensional chains.

4.3.1 Allowed Variation

The $allowedVariation$ parameter decides for each point if it is a chain-point candidate by comparing it to the $normedError$ value of that point. This parameter is useful if the user knows there is some noise next to the chains or if he wants to detect bent chains. Some examples for the $normedError$ were already given in chapter 3.2.3, but let us have a look how $allowedVariation$ influences the output of the chain detection algorithm applied on the traffic accident cluster around London. In figure 4.7, in which only a few chains are detected, the output with a low value of $allowedVariation$ is shown.

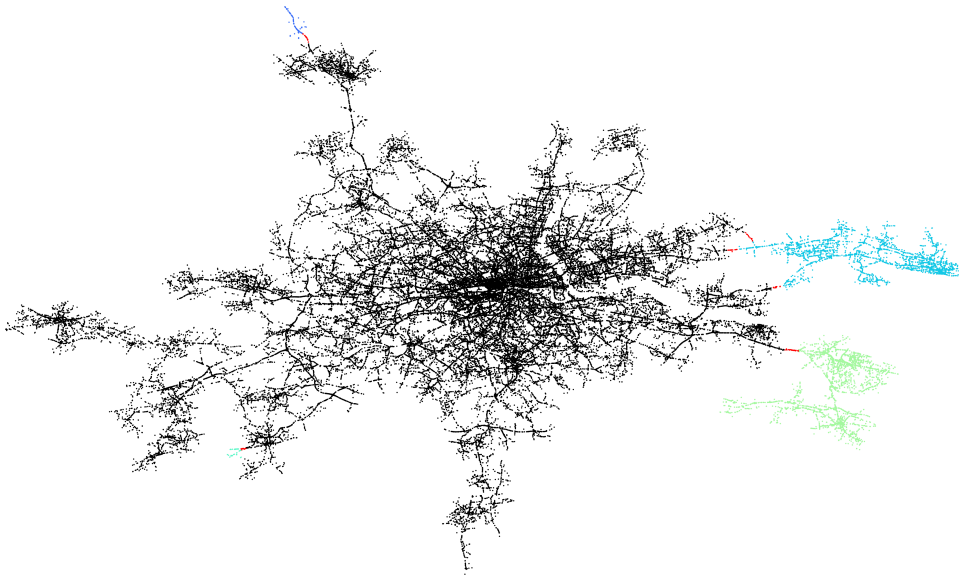

Figure 4.7: Chain-detection with $allowedVariation := 0.1$ and $chainDim := 1$ applied on the cluster of traffic accidents around London. Chains are marked red.

The following two figures 4.8 and 4.9 show that increasing the $allowedVariation$ parameter tends to increase the number of detected chains.

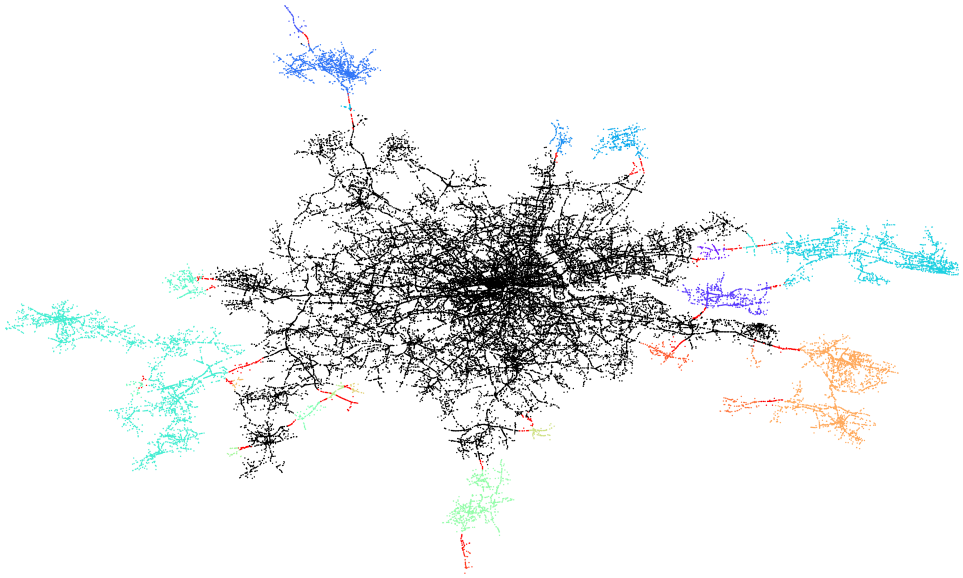

Figure 4.8: Chain-detection with $allowedVariation := 0.3$ and $chainDim := 1$ applied on the cluster of traffic accidents around London. Chains are marked red.

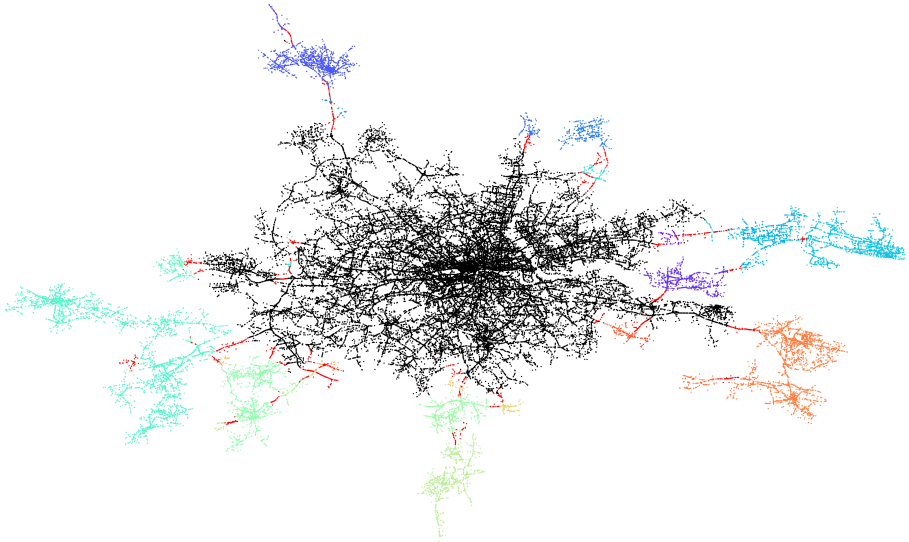

Figure 4.9: Chain-detection with $allowedVariation := 0.4$ and $chainDim := 1$ applied on the cluster of traffic accidents around London. Chains are marked red.

Setting the $allowedVariation$ parameter too high results in too many points detected as chain-point candidates and this can result in chains being detected within a cluster, see figure 4.10.

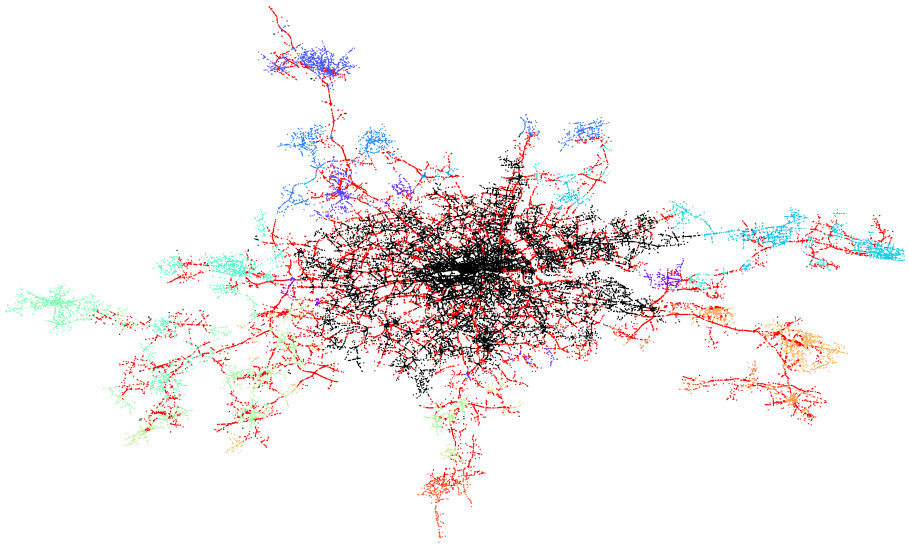

Figure 4.10: Chain-detection with $allowedVariation := 0.6$ and $chainDim := 1$ applied on the cluster of traffic accidents around London. Chains are marked red.

4.3.2 Countering the Limitation: A third parameter

Regarding the limitation of chapter 3.7 one could introduce a third parameter ϵ_2 to set the range of the neighbourhood when checking each point if it is a chain-point candidate. In the previous chapters it was simply assumed that the neighbourhood in question was the same ϵ range as in the DBSCAN algorithm.

The ϵ_2 parameter is a double-edged sword, because although increasing ϵ_2 reduces the error effect of noise within the smaller ϵ 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 range may be considered, thus increasing the error. This effect can be seen by looking at the chains shown in figure 4.11 and compare those to figure 4.12. In figure 4.11 the bottom chain is shorter, because ϵ_2 is smaller and the close range error gets too high. Increasing ϵ_2 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. The upper chain is not detected any more because there is too much noise added within the ϵ_2 range.

If not for specific reasons its probably best to avoid the ϵ_2 parameter, because of its double-edged attribute making it hard to predict its influence on the outcome and a meaningful sense of neighbourhood is already given by the ϵ parameter of the overlying DBSCAN algorithm.

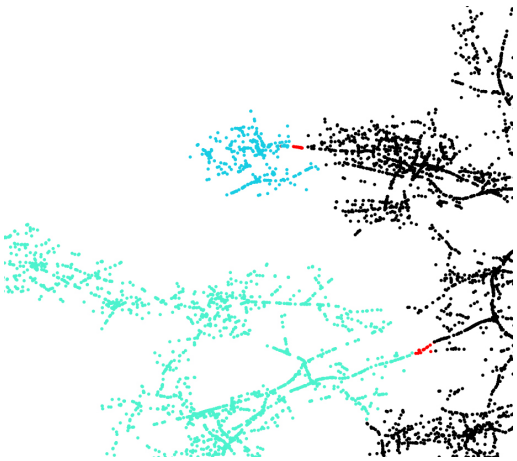

Figure 4.11: Zoom in of figure 4.15 with $\epsilon_2 := 0.15$. Chains are marked red.

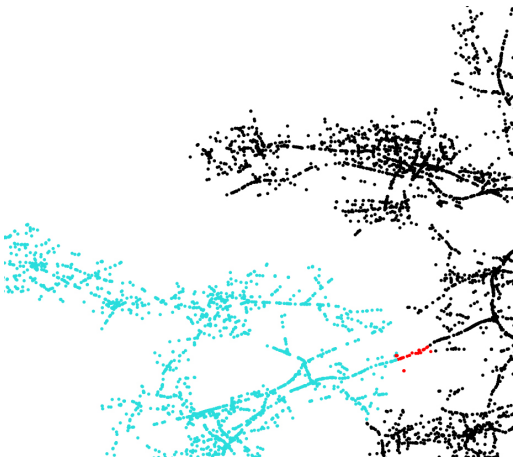

Figure 4.12: Zoom in of figure 4.16 with $\epsilon_2 := 0.2$. Chains are marked red.

In figure 4.13 it is shown that setting ϵ_2 too small results in too many chain-points detected and clustering those with $DBSCAN_{\epsilon,minPts}$ can lead to

chains within clusters, which is probably not desirable. In figure 4.14, 4.15 and 4.16 the results of increased ϵ_2 values are shown.

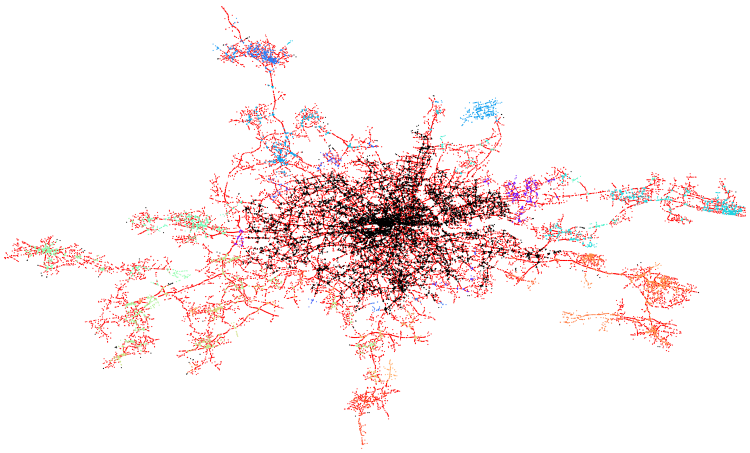

Figure 4.13: Chain-detection with $allowedVariation := 0.2$, $chainDim := 1$ and $\epsilon_2 := 0.025$ applied on the cluster of traffic accidents around London. Chains are marked red.

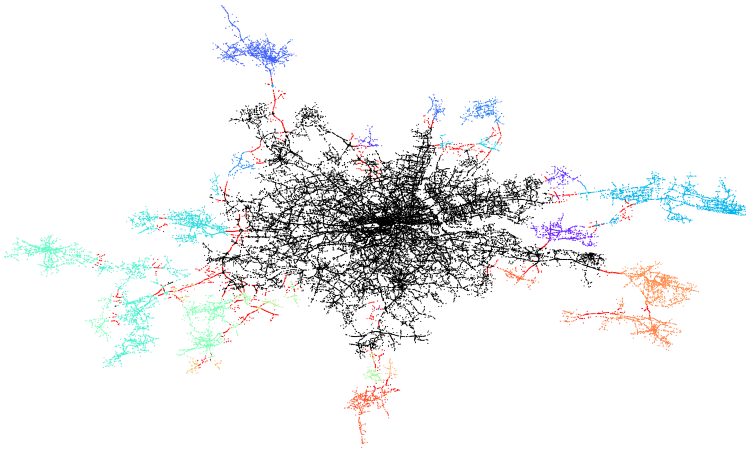

Figure 4.14: Chain-detection with $allowedVariation := 0.2$, $chainDim := 1$ and $\epsilon_2 := 0.05$ applied on the cluster of traffic accidents around London. Chains are marked red.

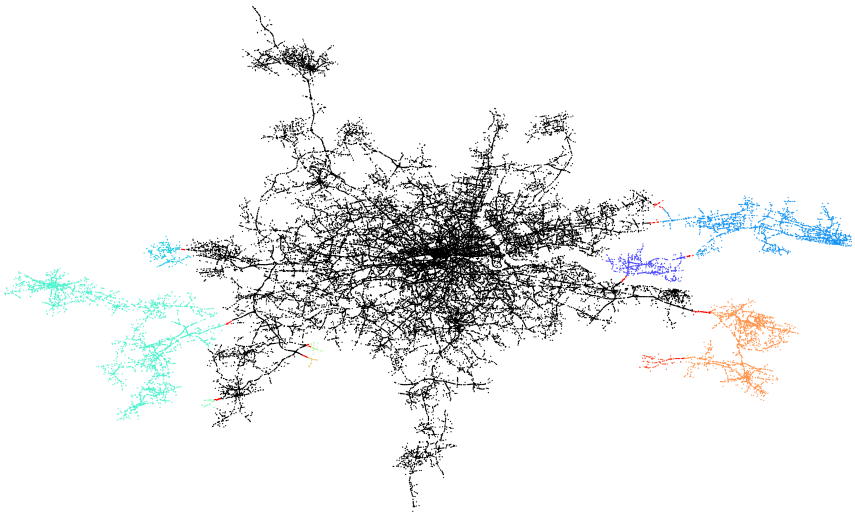

Figure 4.15: Chain-detection with $allowedVariation := 0.2$, $chainDim := 1$ and $\epsilon_2 := 0.15$ applied on the cluster of traffic accidents around London. Chains are marked red.

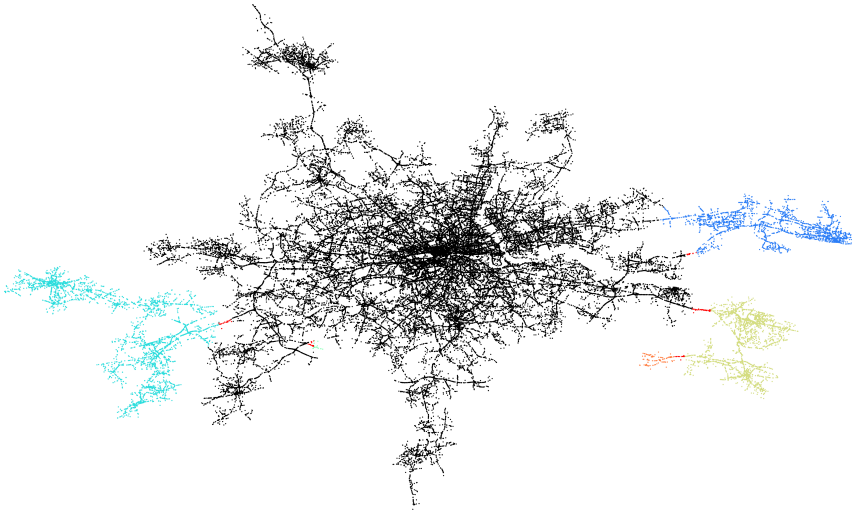

Figure 4.16: Chain-detection with $allowedVariation := 0.2$, $chainDim := 1$ and $\epsilon_2 := 0.2$ applied on the cluster of traffic accidents around London. Chains are marked red.

Chapter 5

Conclusion

In this work, a shape based chain-detection algorithm with intuitive parameters has been presented. Therefore the shape of the neighbourhood of each point is used to indicate whether this point is part of a chain, because after all a chain is defined by its shape. After some refinement and clustering of the chain-point candidates, the chain-candidates are validated if they form a connection between two clusters. In the end the user can either simply mark those validated chains or assign each to a new cluster.

The chain-detection algorithm enhances DBSCAN by eliminating its main weakness, namely that DBSCAN detects clusters which are connected by a thin line as one big cluster. The algorithm also refines the result of DBSCAN, not only because those connected clusters are split up, but also because each chains itself is a cluster, resulting in a much more valuable output. This can improve many applications relying on DBSCAN.

List of Figures

Bibliography

[1] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Ken-ichi Kawarabayashi, and Michael Nett. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 29–38, New York, NY, USA, 2015. ACM.

[2] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial-temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[3] Martin Ester, Hans-Peter Kriegel, Jiirg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise, 1996.

[4] Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao, Di Ma, Shengzhong Feng, and Jianping Fan. Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 473–480. IEEE, 2011.

[5] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments, 2016.

[6] Carlos Ruiz, Myra Spiliopoulou, and Ernestina Menasalvas. C-dbscan: Density-based clustering with constraints. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pages 216–223. Springer, 2007.

[7] Kedar Sawant. Adaptive methods for determining dbscan parameters. *International Journal of Innovative Science, Engineering & Technology*, 1(4), 2014.