On the Edges of Clustering
Creating Synergies with Related Problems

Anna Beer

München 2021

On the Edges of Clustering
Creating Synergies with Related Problems

Anna Beer

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Anna Beer
aus München

München, den 03.09.2021

Erstgutachter: Prof. Dr. Thomas Seidl
Zweitgutachter: Prof. Dr. Ira Assent
Drittgutachter: Prof. Dr. Martin Ester
Tag der mündlichen Prüfung: 18.11.2021

Eidesstattliche Versicherung

Hiermit erkläre ich, Anna Beer, an Eides statt, dass die vorliegende Dissertation von mir selbständig, ohne unerlaubte Hilfe gemäß Promotionsordnung vom 12.07.2011, § 8, Abs. 2 Pkt. 5, angefertigt worden ist.

München, 03.09.2021 Anna Beer

Zusammenfassung

Clustering ist eine der grundlegenden Aufgaben im Data Mining. Das Finden von Gruppen ähnlicher Objekte und gleichzeitige Trennen unähnlicher Objekte liefert wertvolle Ergebnisse in verschiedensten Forschungsbereichen. Da sich sowohl die Daten als auch die Definitionen von Ähnlichkeit je nach Bereich stark voneinander unterscheiden können, eignen sich unterschiedliche Clusteralgorithmen für unterschiedliche Probleme. Aus den diversen Voraussetzungen und Ansprüchen heraus haben sich verschiedene Teilbereiche des Clusterings entwickelt: Im *Subspace Clustering* geht man davon aus, dass nicht alle Attribute für alle Gruppen relevant sind und sucht Gruppen ähnlicher Objekte in Unterräumen des ursprünglichen Datenraums. Falls zudem die Orientierung im Datenraum keine Auswirkung auf die Zusammengehörigkeit von ähnlichen Objekten hat, Datenpunkte also nah beieinander liegen wenn sie auf einen beliebig orientierten Unterraum projiziert werden, befinden wir uns im Bereich des *Correlation Clusterings*. Weitere Forschungsgebiete sind eng mit Clustering verbunden: *Outlier detection* zielt darauf ab, Objekte zu finden, die zu keinem Cluster gehören. Die *Reihenfolge* von Daten wird zwar hauptsächlich im Zusammenhang mit raumfüllenden Kurven oder Indexstrukturen betrachtet, das Sortieren der Daten in eine bestimmte Reihenfolge kann aber auch als wichtiger Vorverarbeitungsschritt für das Clustering dienen. Je nach Anwendungsfall kann auch umgekehrt ein vorheriges Clustern der Daten das Sortieren erleichtern.

In dieser Arbeit betrachten wir Zusammenhänge zwischen *Clustering* und den genannten verwandten Gebieten *Subspace Clustering*, *Correlation Clustering*, *Outlier Detection* und *Reihenfolgen* von Daten. Wir finden Synergien und Übergänge zwischen den Gebieten und Clustering, die uns erlauben, bestehende Probleme zu lösen oder zu vereinfachen.

Subspace Clustering Eine der größten Herausforderungen beim Subspace Clustering besteht darin, gleichzeitig sowohl zu erkennen welche Dimensionen relevant sind, als auch Objekte zu finden, die projiziert auf diese Dimensionen nahe beieinander liegen. Unsere neue Scoring Methode KISS [15] berechnet die Wichtigkeit jeder Dimension für jedes einzelne Objekt. KISS ermöglicht es beispielsweise, die Distanzfunktion eines klassischen Clusteralgorithmus zu gewichten um die Relevanz verschiedener Dimensionen zu berücksichtigen. Gitterbasierte Clustering Ansätze haben viele Vorteile, allerdings ist die mit der Dimensionalität exponentiell ansteigende Anzahl an Gitterzellen problematisch für hochdimensionale Daten. Grace [17] ist ein gitterbasierter Clusteralgorithmus, den wir für hochdimensionale Daten mit volldimensionalen Clustern entwickelt haben, wobei wir

die Anzahl der Gitterzellen in Abhängigkeit von der Anzahl der Punkte begrenzen. Für eine ordentliche Evaluierung von Algorithmen im Zusammenhang mit Subspace Clustering ist die Verwendung synthetischer Daten fast unumgänglich, da bei Realweltdatensätzen nur sehr selten die korrekten oder erwünschten Subspace Cluster im Vorhinein bekannt sind. Deshalb haben wir einen Datengenerator für Datensätze mit Subspace Clustern [20] entwickelt.

Correlation Clustering Wir haben mit LUCK [18] und der Erweiterung LUCKe [23] eine Brücke zwischen Correlation Clustering und Clustering gebaut: Mit Hilfe dieser Methoden können klassische distanzbasierte Clusteralgorithmen eindimensionale bzw. beliebig dimensionale Correlation Cluster finden. Zudem verbessern wir die Ergebnisse von DBSCAN und anderen Clusteralgorithmen [47, 46] mit Hilfe einer der bekanntesten Techniken aus dem Correlation Clustering, der Hauptkomponentenanalyse (PCA). Diese ermöglicht es, Ketten von Objekten, die verschiedene Cluster verbinden, zu erkennen und die Cluster voneinander zu trennen. Zudem haben wir das erste interne Evaluationsmaß für Correlation Clustering [54] entwickelt.

Outlier Detection Die Idee der winkelbasierten Ausreißererkennung kann auf den Bereich des Clusterings übertragen und verfeinert werden, um Randpunkte von Clustern zu erkennen. ABC [21] nutzt diese Randpunkte für ein beschleunigtes Clusterverfahren. Für MORe++ [19] verwenden wir den Clusteralgorithmus k-Means, um schnell und intuitiv interpretierbar Ausreißer in hochdimensionalen Daten zu finden.

Reihenfolge Die Reihenfolge von Daten ist ein relevantes Problem für Data Mining im Allgemeinen und kann ein wichtiger Vorverarbeitungsschritt für Clustering sein [16]. Wir haben den Circle Index [22] entwickelt, ein Qualitätsmaß für die Reihenfolge von Knoten in einem Graphen. Unser Clusteralgorithmus CirClu [22] nutzt aus, dass eine Reihenfolge mit niedrigem Circle Index das Clustern von Knoten erheblich vereinfacht.

Abstract

Clustering is one of the fundamental tasks in data mining. The need to find groups of similar objects while separating dissimilar ones is widespread among diverse research areas. As properties of data from these areas differ heavily, there is no "one fits all" cluster algorithm, and new data gathering methods emerge steadily. For different assumptions on the data, different sub-areas of clustering evolved: *subspace clustering* algorithms find clusters in a subspace of the data space, where we assume some dimensions to be irrelevant. If also the orientation in the data space is supposed to be irrelevant, i.e., points are close when projected onto an arbitrarily oriented subspace, we perform *correlation clustering*. There are also adjacent research areas that are closely linked to clustering: *Outlier detection* aims to find points that are not part of any cluster. And even though *ordering* of data is mainly regarded in the context of space-filling curves or index structures, it can serve as an important preprocessing step for clustering. Also clustering the data first can improve the ordering of the data, depending on the use case.

In this work, we regard coherences between clustering and the aforementioned related tasks *subspace clustering*, *correlation clustering*, *outlier detection*, and *ordering*. We apply knowledge and ideas from clustering to solve related problems and vice versa, creating synergies and finding transitions between the fields.

Subspace Clustering One of the biggest challenges in subspace clustering is to simultaneously find important dimensions and points which are close when projected to these dimensions. Our new score KISS [15] indicates the importance of every dimension for each point individually. KISS enables, e.g., to weight the distance function of a common clustering algorithm to account for the relevance of diverse subspaces. Grid-based approaches can facilitate subspace clustering, but for high dimensionality, an exponential number of grid cells is not manageable. Grace [17] is a grid-based clustering method designed for high-dimensional data containing clusters in the full-dimensional space, where we limit the number of grid cells depending on the number of points. For an extensive evaluation of algorithms in connection with subspace clustering, using synthetic data is almost inevitable. We created a subspace cluster generator [20] as real-world data sets containing subspace clusters which are also labeled as such are rare.

Correlation Clustering Regarding the connection between clustering and correlation clustering, we developed LUCK [18] and its extension LUCKe [23], methods that enable

every distance-based cluster algorithm to find one-dimensional resp. arbitrary dimensional correlation clusters. The other way around, we used one of the most common methods from the field of correlation clustering, namely principal component analysis (PCA), to detect and remove chains of points connecting different clusters to improve clustering results by DBSCAN [46] and other cluster algorithms [47]. Furthermore, we developed the first internal evaluation measure for correlation clusters [54].

Outlier Detection We refined the idea of angle-based outlier detection to identify border points of a cluster, leading to our fast angle-based clustering algorithm ABC [21]. With MORe++ [19] we use k-Means clustering to find outliers in high-dimensional data.

Ordering We found that the ordering of data points is a relevant problem for data mining in general and an important preprocessing step for clustering in [16]. We developed the Circle Index [22], a measure for the quality of node ordering. Reordering nodes such that the Circle Index is minimized simplifies clustering significantly.

Contents

1 Introduction **1**
- 1.1 Clustering . . . 3
- 1.2 Subspace Clustering . . . 9
- 1.3 Correlation Clustering . . . 13
- 1.4 Outlier Detection . . . 16
- 1.5 Ordering . . . 19
- 1.6 Conclusion . . . 20

2 Between Clustering and Subspace Clustering **23**
- 2.1 "Grace - Limiting the Number of Grid Cells for Clustering High-Dimensional Data" . . . 24
- 2.2 "A Generator for Subspace Clusters" . . . 37
- 2.3 "KISS - A fast kNN-based Importance Score for Subspaces" . . . 43

3 Between Clustering and Correlation Clustering **51**
- 3.1 "I fold you so! An internal evaluation measure for arbitrary oriented subspace clustering" . . . 52
- 3.2 "LUCK- Linear Correlation Clustering Using Cluster Algorithms and a KNN Based Distance Function" . . . 61
- 3.3 "LUCKe — Connecting Clustering and Correlation Clustering" . . . 66
- 3.4 "Chain-detection for DBSCAN" . . . 77
- 3.5 "Chain-detection Between Clusters" . . . 89

4 Between Clustering and Outlier Detection **103**
- 4.1 "MORe++: k-Means Based Outlier Removal on High-Dimensional Data" . 104
- 4.2 "Angle-Based Clustering" . . . 120

5 Between Clustering and Ordering **131**
- 5.1 "Orderings of Data - More Than a Tripping Hazard: Visionary" . . . 132
- 5.2 "Graph Ordering and Clustering: A Circular Approach" . . . 137

Bibliography **143**

Chapter 1

Introduction

The amount of data humanity gathers daily is overwhelming, and most of it comes unstructured. Usually, data is collected in order to learn something from it, to improve tasks related to it, or to profit from it in one way or another. The fields gathering data are diverse. If there were fields not gathering or producing any data, they would surely make a shorter list than enumerating all the interesting fields that benefit from data mining. Some of the most frequently named fields that already gathered plenty of interesting and useful data are medicine, finance, social networks, and physics. For over 20 years we know that there is "an urgent need for a new generation of computational theories and tools to assist humans in extracting useful information (knowledge) from the rapidly growing volumes of digital data" [38]. Thus, the field of *knowledge discovery (in databases)* (KDD) emerged. The field is concerned about "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data" [40]. The KDD process encompasses *selection*, *preprocessing*, *transformation*, *data mining* and *interpretation/evaluation* of data [39]. For this thesis, we focus on *data mining*, which comprises tasks like *clustering*, *outlier detection*, *classification*, *regression*, *summarization*, and *dependency modeling* [38]. Some of the tasks are connected to a certain degree. For example, classification is related to clustering as both rely on groups of data points that belong together. Furthermore, outliers can be seen as points that do not belong to any cluster [37]. Some state that "one of the most important of the myriad of data analysis activities is to classify or group data into a set of categories or clusters" [79]. If the categories are known beforehand, we speak of *classification*, otherwise we speak of *clustering*: *clustering* means to unsupervisedly partition the data in groups of similar objects while separating dissimilar ones. Knowing such groups can deliver valuable insights into the data and serve as basis for further analysis.

Several subcategories of clustering as well as other related research fields developed due to diverse new challenges that arose over the years. As storage became cheaper, the volume and dimensionality of gathered data increased, leading to additional, potentially irrelevant information in the data, which can mask the underlying clusters. *Subspace clustering* algorithms consider potentially irrelevant attributes by looking for clusters of points that are similar in only a subspace of the full data space. *Correlation clustering* algorithms focus on correlated or redundant features, leading to *arbitrarily oriented subspace clusters*.

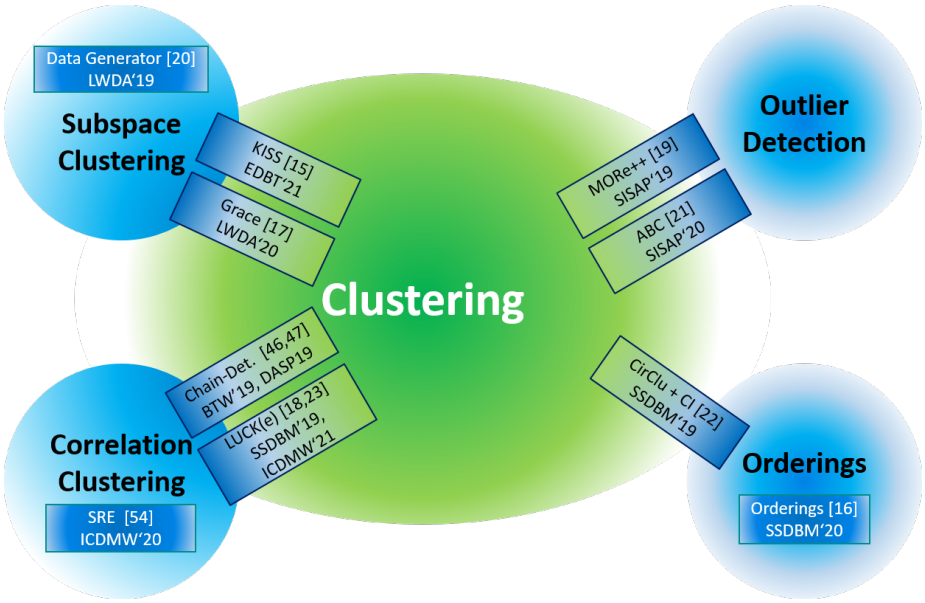


Figure 1.1: Publications included in this thesis building bridges between research areas.

Even though *clustering*, *subspace clustering*, and *correlation clustering* are similar not only regarding their terminology, but also their objective, common approaches differ fundamentally from each other. While specializing on their respective niches, the research fields drifted more and more apart over time. Thus different algorithms with different advantages and disadvantages developed. However, in this thesis, we transfeFr promising ideas from the fields to each other, find synergies between them, exploit their commonalities, and create and improve solutions to open problems.

Subspace clustering and correlation clustering are not the only fields profiting from ideas typically used for clustering and vice versa. Another field is *outlier detection*, which aims to find data points that are anomalous regarding the rest of the data set. As outliers can also be seen as points that do not belong to any cluster, we investigate synergies between outlier detection and clustering.

The last field we regard in connection to clustering is the *ordering* of data. Ordering of the input can not only influence most data mining tasks, but we can also take advantage of a suitable ordering and optimize the ordering to simplify clustering.

The remainder of this thesis is structured as follows. In Chapter 1 we introduce clustering as well as four related research areas, namely subspace clustering, correlation clustering, outlier detection, and ordering. Section 1.1 gives a detailed description as well as fundamental challenges and approaches in connection with clustering, which is the pivotal point of this thesis. Setting our publications into context, we give an introduction to each of the aforementioned related research areas in Sections 1.2 - 1.5 and refer to the respective publications containing all details on the developed methods in Chapters 2 - 5.

Figure 1.1 gives an overview of the publications resp. published methods included in this thesis and at which conferences they were published.

1.1 Clustering

"There are many techniques of cluster analysis and it is difficult to judge their relative merits and demerits, because a cluster is not a well defined concept" [42]. Even though the last sentence dates back to 1967, up to today, the difficulty of judging cluster analysis techniques has not changed. Even though there are thousands of publications related to clustering, there is still no universally agreed upon exact definition of a "good" clustering. For some, it is even debatable if clustering is "Science or Art" [78]. This could be seen as a problem, as researchers in the domain of clustering are trying to solve a problem without a clear definition. But in fact, it is not only a challenge but also a chance to progress and to adapt to the current needs of the world.

In the following, we give an overview of some relevant aspects of clustering from different perspectives and put this thesis into context. First, we regard different notions of a cluster together with the most common algorithm aimed at the detection of clusters of its type. Then we regard the characteristics clustering algorithms can have, like for which data or use case they are suitable. Lastly, we look at challenges that occur independently of the notion of a cluster or characteristics of the clustering algorithm.

Concepts of clustering

As mentioned before, the fundamental idea of clustering is to partition some data into groups of similar objects while separating dissimilar ones. Because of the multitude of use cases for clustering, which can be valuable for basically every research field producing data, there are numerous attempts at defining a "good" clustering. Depending on the use case and data set, different notions of similarity may be applied. That results not only in the application of different similarity measures but also in different notions of a cluster itself. The most common types of clustering are centroid-based, density-based and hierarchical clustering, of which we give an overview in the following.

Centroid-based clustering algorithms represent clusters by their centroids and usually try to minimize the distance between objects and their respective centroid. k-Means [63, 64] is the most famous representative and uses the means of all objects assigned to a cluster as its centroid. In an iterative approach, it assigns each point to its nearest cluster center and then recalculates the centers.

- *Advantages:* Algorithms of this type are typically fast and easy to implement and belong to the probably most frequently used type of clustering algorithms.
- *Disadvantages:* They usually need the number of clusters as user input and find only convexly shaped clusters.
- *Occurrence in this thesis:* We use k-Means++ [8], a successor of k-Means that improves the initial selection of the cluster centers, in our outlier detection method MORe++ [19] in Chapter 4.

Density-based clustering algorithms define a cluster as a continuous dense region, with DBSCAN [36] as the first and most famous representative. The user-defined parameters for the minimal number of points that lie in a range ε around a point define whether a point is a core point. Core points closer to each other than ε are connected to clusters so that clusters are divided by rather sparse regions.

- *Advantages:* Algorithms of this type can find arbitrarily shaped clusters and handle noise. They can work without having information on the exact data points as the mere distances between all data points are sufficient.
- *Disadvantages:* Choosing good parameters is typically not intuitive. For high-dimensional data the notion of density-connection becomes questionable and the *empty space phenomenon* (explained in Section 1.2) exacerbates the selection of meaningful parameters further. Algorithms from this category are prone to the *single-link effect*: an unfortunate "chain" of noisy data points can connect distinct clusters. Many (old) density-based clustering algorithms cannot detect clusters of different densities, however, tackling this challenge can successfully be incorporated into density-based clustering algorithms [58, 35, 72].
- *Occurrence in this thesis:* In Chapter 3 we tackle the single-link effect and use methods from the field of correlation clustering to improve not only DBSCAN but also other clustering algorithms by identifying and removing chains of noise from the clustering process [46, 47].

Hierarchical clustering algorithms provide a hierarchy of clusters based on the assumption that clusters can be part of other clusters. They can operate in a bottom-up agglomerative way or in a top-down divisive way. Agglomerative approaches usually combine the clusters that are most similar regarding single linkage, average linkage, or complete linkage [66].

- *Advantages:* A taxonomy gives detailed information on the data and the number of clusters can be chosen afterwards based on the results.
- *Disadvantages:* A manual analysis of the results is necessary to receive a partitioning of the data, which makes fair comparisons to other methods rather difficult and requires expert knowledge. Single-linkage hierarchical clustering is prone to the single-link effect explained above.
- *Occurrence in this thesis:* We use an adaption of single linkage hierarchical clustering in our clustering algorithm ABC [21] in Chapter 4, as its properties are suitable when applied on a data set containing only bordering points of clusters.

Note that these types of clustering are non-exclusive. For example, OPTICS [7] is density-based as well as hierarchical.

Characteristics of clustering algorithms

Important characteristics of the data and the objective of the clustering lead to further types of clustering algorithms that can be differentiated by answering the following non-exhaustive list of questions:

- **In which form or abstraction level is the data available or best to use?**
  Tabular data consists of a set of objects that have a number of attributes or dimensions. Data can also be available as a graph or be transformed to a graph before clustering, e.g., using a kNN-graph or an ε-graph. A kNN graph represents all points as nodes and connects each point with its k nearest neighbors (kNN), where an ε-graph connects it to all other points within a radius ε. Transforming the data into a graph first abstracts the data by potentially omitting some information like the exact distances between all points. Working on graphs can have diverse advantages that graph-based clustering algorithms like, e.g., spectral clustering [68] exploit.
  In this thesis, we regard mainly *tabular* data. Solely our clustering algorithm CirClu [22] introduced in Section 1.5 and presented in detail in Section 5.2 works on graphs.
- **For tabular data: which *type* do the objects have?**
  Usually, one differentiates between *numerical*, *ordinal*, and *categorical* resp. *nominal*. Not all clustering algorithms can be applied to every data type. E.g., as there is no mean defined for categorical values, some centroid-based clustering algorithms are not applicable.
  In this thesis, we mainly regard numerical data, i.e., with the term *data set* we usually refer to a set X of objects or points p_i in the d-dimensional space $\mathbb{R}^d$.
- **How much *user interaction* is possible or wanted?**
  Cluster algorithms can be fully automatic, but mostly they have user-defined input parameters. There are also approaches performing clustering in a semi-supervised fashion [11], but clustering in the proper sense is an unsupervised task.
  In this thesis, we regard exclusively unsupervised clustering and limit the number of user-defined parameters to a minimum.
- **Should every point be assigned to a cluster unambiguously?**
  Accounting for uncertainty, *fuzzy* or *soft* clustering algorithms assign points to clusters with a certain degree or weight, where *hard* clustering algorithms assign each point to exactly one cluster [80]. Often a clear partitioning of the data is desirable, e.g., for further automatic processing.
  Thus, we regard exclusively hard clustering algorithms in this thesis.
- **Is the data *noisy*?**
  Noise-handling algorithms do not necessarily assign every point to a cluster but allow for noise and outliers. It is also possible to first perform outlier detection or noise removal and subsequently apply the clustering algorithm to improve clustering results.
  All approaches introduced in this thesis that work on tabular data can handle noise.

Having all those aspects in mind, we see that clustering is a very versatile task, which may be the reason for its prevalence in diverse data-driven research areas. But it is also a challenge: the requirements in diverse areas differ heavily and in contrast to, e.g., classification, clustering provides results without any supervision or training. Developing a clustering algorithm that fulfills *all* clustering tasks at once may be a tempting idea. Nevertheless, the continuous progress in research areas relying on clustering algorithms impedes the development of one clustering algorithm that fulfills *all* requirements for all possible clustering tasks. Thus, many approaches focus on solving use case-specific clustering tasks. But even for very specific tasks, some challenges cannot be circumvented.

Challenges

There are challenges that complicate all data mining tasks, like for example incomplete or highly noisy data sets and data sets with errors or duplicates. In the following, we rather focus on challenges specific to clustering. The challenges listed below can occur independently from the exact clustering task and some of them caused the emergence of distinct research areas.

High volume. With the increasing automation of data gathering, the number of objects per data set increases, too. This causes longer running times for basically all clustering algorithms, where algorithms with lower complexity stay applicable in real-world scenarios even for larger data sets. There are several techniques that are frequently used to decrease the complexity or runtime of clustering algorithms. E.g., performing complex operations on only a rather small *sample* of the data set can decrease the runtime. Also, summarizing close objects with a grid instead of working on the original objects can reduce the complexity (as well as the runtime, given a large enough data set):

- *Random sampling:* Performing the clustering algorithm on only a *random sample* of the data set and assigning residual objects afterwards decreases the runtime, but not the complexity. As random sampling is a non-deterministic process, also the results of the clustering are non-deterministic. Unfortunate sampling can lead to bad results especially in noisy data sets or data sets with many outliers.
- *Selective sampling:* Performing the complex parts of a clustering algorithm on only a sample of specifically selected suitable objects from the data set can decrease the runtime significantly without the disadvantages of random sampling. The assignment of the residual out-of-sample objects is usually quite fast. Here, a good selection strategy is decisive for the quality of the overall clustering. In our approach ABC [21] (see Sections 1.4 and 4.2) we select only points at the border of a cluster because these points already define the cluster but constitute only a small fraction of the overall data points.
- *Summarization:* Roughly summarizing close points with a *grid*-based approach has several advantages: Assigning points to grid cells is very fast if a simple grid structure

is used and the subsequent complexity depends on the number of grid cells instead of the number of objects. Thus the runtime decreases if there are fewer grid cells than objects. However, the number of grid cells grows exponentially with respect to the number of dimensions, thus grid-based approaches are often unsuitable for high-dimensional data. With our clustering approach Grace [17] (Section 2.1) we investigate limiting the number of grid cells dependent on the number of points, leading to a hybrid between clustering and subspace clustering as explained in Section 1.2.

High dimensionality. With the decreasing price for main-memory and flash drives as well as various new data gathering methods, e.g., in medicine, chemistry, or physics, the number of attributes in data sets nowadays has multiplied compared to the beginnings of research in the field of clustering. In Sections 1.2 and 1.3 we explain in detail disadvantageous effects of high-dimensional data, often summarized as the *curse of dimensionality*, with respect to the clustering task and research directions that developed to tackle them, i.e., subspace clustering and correlation clustering.

Robustness. "[T]he performance of a method should deteriorate only slightly under small deviations, and it should have a good efficiency (accuracy) of estimation" [33]. Robust algorithms fulfill this by providing similar results for similar inputs, which is desirable for a multitude of reasons. E.g., it allows a constant quality of results and can increase reproducibility. The latter is not only relevant for scientific experiments but also for technical exchange and as a basis for discussions. Robustness in clustering algorithms is multifaceted and can concern diverse aspects. Even though most clustering algorithms regard data *sets*, small perturbations in the ordering of the input data can already influence the final results of algorithms that are not robust. Ordering the input data, a topic we regard in Chapter 5, can stabilize such results. Another type of robustness concerns the input parameters. Often, they are based on "expert knowledge", but two experts may have (slightly) different opinions on the exact "best" parameters. Nevertheless, resulting clusterings should not deviate significantly from each other nor be contradictory. Real-world data is usually noisy, i.e., it contains background noise or jitter. While background noise consists of additional points that do not belong to the original distribution and are often rather uniformly distributed across the data space, jitter denotes small deviations or perturbations in the original distribution. Both, noise as well as jitter, should not compromise the final clustering significantly. Even though noisy data sets are frequently regarded in the field of clustering, in the field of correlation clustering, they are rather neglected. With our methods LUCK [18] and LUCKe [23], introduced in Chapter 3, we allow to transfer benefits from distance-based clustering algorithms to correlation clustering. Using LUCK(e), noise-handling clustering algorithms can find arbitrarily oriented and arbitrary dimensional subspace clusters while still offering benefits from their robustness w.r.t. noise. Especially unfavorable noise can form "chains" that connect disparate clusters, particularly when using density-based algorithms. We tackle this problem with our chain-detection methods [46, 47] introduced in Section 1.3 and described in detail in Chapter 3.

Evaluation

Evaluating and comparing clustering algorithms proves difficult since clustering is such a versatile task. For evaluating a clustering algorithm its runtime, usability, degree of necessary expert knowledge, simplicity of implementation, and many more aspects can be relevant. But most often, the quality of a clustering algorithm refers to the quality of its results for users as well as researchers. To evaluate the quality of clustering results we can use *external* or *internal* evaluation measures.

External evaluation One possibility to evaluate the quality of a clustering is to compare its results on real-world data sets to a *"ground truth"* or *labelling*. As the "ground truth" is usually determined by an expert in the field the data stems from, the process of labeling data is an expensive task. Thus, the number of available labeled data sets is limited and developing clustering algorithms solely based on a few labeled data sets can lead to overfitting. Depending on the purpose of the clustering or the field the labeling expert comes from, different labellings may be seen as the "ground truth". Moreover, experts are human, and humans may err. Thus, the correctness of the labellings and their value for evaluation is sometimes questionable.

Furthermore, if the "ground truth" or desirable *classes* are already known, the clustering task is sort of obsolete. Hence, this type of evaluation is mainly used for research purposes and the general evaluation of an algorithm rather than for the evaluation of a certain clustering result. Synthetic data sets can support examining the quality of clustering algorithms depending on certain aspects of the data and in most cases automatically provide a ground truth. As the generation of data sets with properties worth examining can be a cumbersome task, data generators can ease the workload of researchers in the field. Thus, we develop in Section 2.2 an easy-to-use generator for data containing subspace clusters [20] that is also available online.

Given a reliable ground truth, we can examine clustering results with a plethora of *external evaluation measures*. Different external evaluation measures used for clustering consider different aspects as important and use different techniques. Most common techniques are based on set matching, counting pairs, or entropy [6]. Often, a value between 0 and 1 is desirable, thus there are normalized versions of some of the measures. Furthermore, adjustments for chance can set the quality in proportion to a random assignment of objects to clusters. Exact descriptions, advantages and disadvantages of commonly used external measures can be found in diverse surveys [74, 76, 77]. In this thesis, we mostly use the wide spread Normalized Mutual Information (NMI), adjusted rand index (ARI), as well as precision and recall as external evaluation measures.

Internal evaluation With *internal evaluation measures* the quality of a clustering can be indicated without the necessity of labeling the data first. Like that, internal evaluation measures can also be used as stopping criteria during the execution of clustering algorithms. Internal evaluation measures for clustering usually regard *compactness* and *separation* [62]. Compactness evaluates the similarity of objects belonging to the same

cluster and separation describes the disparity between objects of different clusters. The calculation of internal evaluation measures can, e.g., be based on distances to a centroid, pair-wise distances, or statistical methods like variance or entropy. However, all internal quality measures rely on some model of a "good" clustering, which means that only algorithms that find clusters of a type fitting the respective model achieve good scores.

We develop an internal quality measure for correlation clustering in Section 3.1 and an internal quality measure for ordering in Section 5.2.

1.2 Subspace Clustering

The amount and dimensionality of gathered data increased significantly over the last decades: nowadays, sensor networks are widespread, new examination methods in chemistry and molecular biology are developed steadily, and social networks produce a multitude of various data by millions of users every day. Even though we gather more information than before, discovering knowledge in databases does not get easier. The more attributes we gather, the more may be irrelevant for some data mining tasks. For different groups of data points, different features may be relevant, which is often referred to as *local feature relevance*. The more dimensions a data set has, the more different meaningful groupings could exist. The *empty space phenomenon* describes that for a fixed number of points the number of empty grid cells grows exponentially with respect to the dimensionality such that "most" of the high-dimensional space is empty. Especially the quality of grid-based and density-based clustering algorithms can suffer from this effect. With increasing dimensionality of the data the impact of further disadvantageous properties for clustering increases, too, and is often referred to as the *curse of dimensionality*, first mentioned by Bellman [24] and elaborated in a plethora of subsequent literature (e.g., [59, 26, 82]). It encompasses several aspects of high-dimensional data that are unfavorable for traditional data mining tasks. For example, the "discrimination between the nearest and the farthest neighbor becomes rather poor in high-dimensional space" [59] as mathematically described below. Thus, the usefulness of common distance measures like the Euclidean distance becomes questionable with increasing dimensionality. For a (independent and identically distributed) data set X with d dimensions and a distance measure $dist$ it holds that:

$$\lim_{d\to\infty} \frac{\max_{x,y\in X} dist(x,y) - \min_{x,y\in X} dist(x,y)}{\min_{x,y\in X} dist(x,y)} = 1 \tag{1.1}$$

This decreased discriminability of distances in high-dimensional data means, that "objects are almost equi-distant, and clustering based on any such similarity assessment is meaningless" [9].

Thus, common clustering algorithms are usually not able to find meaningful clusterings in high-dimensional data, which sparked the emergence of clustering algorithms specifically aimed at high-dimensional data. Throughout the thesis, we use the term *subspace clustering* as a generic term for all clustering methods that at some point regard only a subset of attributes to tackle this problem. Subspace clustering algorithms provide, additionally to

the grouping of the data into clusters, an associated subspace, i.e., a subset of attributes, for every cluster. Points of a cluster are close or similar if they are projected onto this subspace.

Types of subspace clustering The field of subspace clustering is sometimes further differentiated according to different properties:

- The notion of subspaces and the number of clusters a point can get assigned to constitutes the categories *projected clustering*, *soft projected clustering*, *subspace clustering*, and *hybrid approaches* [59]. Even though this differentiation is common especially in early publications in the field, the exact terminology is neither used accurately in current research [59] nor notably intuitive (e.g., the main difference between *projected clustering* and *subspace clustering* is the number of clusters a point can get assigned to), thus we do not use this differentiation further.

- Depending on a hard or soft assignment of dimensions to subspace clusters, we can differentiate between *hard subspace clustering* and *soft subspace clustering*: Assuming each cluster belongs to an exact subspace of the full space is the foundation for *hard subspace clustering*, while assigning weights to all dimensions according to their relevance for a cluster is called *soft subspace clustering* [34](analogously to hard and soft clustering as described in Section 1.1).

- According to their algorithmic approach most algorithms in the field of subspace clustering can be assigned to the group of *bottom-up* or *top-down* approaches [71, 59]. For pruning, *bottom-up* approaches often use the *downward closure property*, describing that if a d-dimensional space is dense (i.e., it contains at least a certain number of points per volume), then its $d-1$-dimensional projections are also dense. Top-down approaches often rely on the *locality assumption*, implying that members of a subspace cluster are close even in the full-dimensional space.

- For very broad definitions of subspace clustering also the orientation of subspaces can serve as a categorization aspect: all previously mentioned methods refer to axis-parallel subspaces. In contrast, arbitrarily oriented subspace clustering, also called *correlation clustering*, allows combining different attributes from the original space to only one attribute in the subspace. Most often, *linear correlations* between attributes, characterized by groups of points lying on (or close to) hyperplanes or lower dimensional arbitrarily oriented subspaces in the full space are regarded. For further details on correlation clustering, see Section 1.3. In this thesis, we do not include correlation clustering when we refer to subspace clustering as it differs significantly from the previously defined subareas.

Extensive enumerations, descriptions, and assignments to subcategories of a plethora of subspace clustering algorithms can be found in diverse surveys on the topic [34, 59, 60, 71]. All those approaches have in common that they consider the relevance of attributes for each cluster in order to reduce difficulties connected to the *curse of dimensionality*.

Grid-based Approaches One of the first approaches to subspace clustering, namely CLIQUE [5], was developed in 1998 and builds the foundation for various grid-based bottom-up subspace clustering algorithms, like, e.g., ENCLUS [31], or MAFIA [67]. Grid-based approaches have several advantages, e.g., they can handle large data sets efficiently and are able to find density-based, potentially non-convex clusters. However, for a given cell width the number of grid cells grows exponentially with the number of dimensions and due to the *empty space problem* most grid-cells do not contain any points in very high-dimensional data. Additionally, choosing the "best" cell width for clustering is not easy and equidistant grids are often unsuitable for data sets containing clusters of different densities. Our grid-based approach Grace [17] tackles these problems by automatically adapting a grid to the data based on the distribution of points in one-dimensional projections of the data. Grace partitions each axis prevalently at positions where there are strong changes in the density of the data. In this way, the resulting non-equidistant grid incorporates the inherent structure of the data. We consider differently important dimensions by partitioning the data independently of the axis at the respectively most suitable position. This incremental approach potentially yields a different number of divisions for each axis. Thus, Grace positions itself between subspace clustering and clustering, as the type of cluster (full-dimensional or subspace cluster) depends on the data and is chosen automatically. Moreover, Grace limits the number of overall grid cells dependent on the number of points, circumventing an exponential number of grid cells with respect to the number of dimensions. In contrast to other algorithms using an elaborated or adaptive grid, like, e.g., OptiGrid [49], Grace can find clusters of non-convex shape. While OptiGrid interprets dense grid-cells as clusters, Grace connects adjacent dense grid cells, providing high-dimensional density-based clusters. All details on Grace can be found in Section 2.1 ([17]).

When developing new subspace clustering algorithms, we encounter another quite neglected problem: the evaluation of their quality. While there are plenty of data sets labeled according to their clusters, finding real-world data with given ground truth for the clusters as well as the subspaces that belong to the clusters is hard. Labeling data is a cumbersome task, especially working on high-dimensional data, and knowing the correct subspace for each cluster requires expert knowledge. As an alternative to real-world data, synthetic data allows examining algorithms in detail regarding diverse properties of the data. Thus, most subspace clustering algorithms are evaluated based on synthetic data sets where the ground truth is unambiguous and existent. Creating synthetic data with data generators allows having a multitude of labeled data sets available without the risk of overfitting on a few data sets throughout the development process. In Section 2.2 we elaborate in detail on existing generators, their availability, and the need for a flexible and at the same time easy-to-use generator for data sets containing subspace clusters. Our data generator [20] as described in Section 2.2 is publicly available and simplifies not only the development process of new subspace clustering algorithms but also supports reproducibility and comparability among algorithms in the field.

Subspace search One of the main challenges in the field of subspace clustering is to identify the relevant attributes for each cluster. As the clusters are not known beforehand, the points forming a cluster as well as the dimensions which are relevant for the respective cluster need to be detected simultaneously. Some dimensions may not be relevant at all, while some dimensions may be relevant for all clusters. A point could belong to different clusters if it is regarded in different subspaces (compare to the "traditional" subspace clustering in the narrower sense), while some points may not belong to any clusters, so-called outliers, which we explain in more detail in Section 1.4.

Subspace search is a rather small research field aimed at finding and scoring subspaces in order to simplify and improve clustering. Taking each dimension's relevance into account when calculating the distance between two points allows purely distance-based clustering algorithms to find subspace clusters that would not be detectable in the original full space. If, as common in high-dimensional real-world data sets, only relatively few dimensions are relevant, the effect described in Equation 1.1 is weakened, and clusters become more meaningful as the impact of noise and irrelevant dimensions is reduced.

Incorporating the importance of dimensions can enable clustering algorithms to find subspace clusters, as already shown in previous research. For example, RIS [52] ranks subspaces according to their "interestingness" as preprocessing step for arbitrary clustering algorithms, which then operate only on an interesting subspace. But regarding the most common related algorithms RIS [52], SURFING [12], and SCHISM [75], none of them seems appropriate in everyday use as we elaborate in Section 2.3. Especially for high-dimensional data, "good" parameters are difficult to choose even for experts, and non-linear complexity of the algorithms results in infeasible runtimes for every day use cases. However, incorporating the importance of subspaces has great potential to improve the field of subspace clustering as advantages of highly elaborated clustering algorithms can directly be transferred to the field of subspace clustering. Thus, we develop KISS [15], a **k**NN-based **I**mportance **S**core of **S**ubspaces, in Section 2.3. It works fully automatically and its runtime complexity is only linear in the number of dimensions and $O(n \cdot \log n)$ in the number of points. This favorable combination of properties is not reached by any of the previously mentioned algorithms in this field. Commonly, subspace clustering and subspace search algorithms are based on the assumption that for points of different clusters different attributes are relevant. For KISS, we refine this idea of the *local feature relevance* further and assume that the relevance of attributes can not only differ for points of different clusters, but for all points. KISS is based on the idea that a dimension is relevant for a point if the point is probably part of a cluster that lies in a subspace containing this dimension. This probability is connected to the occurrence of points in the sets of one-dimensional kNN regarding different attributes, as explained in detail in Section 2.3. Regarding the kNN in only one-dimensional projections allows abdicating from distance measures operating on high-dimensional data that become less meaningful for increasing dimensionality. KISS allows transferring advantages from clustering to the field of subspace clustering, while being fast, explainable, and mitigating several aspects of the curse of dimensionality.

Contributions With our clustering algorithm Grace [17] we develop a method to limit the number of grid cells and tackling the empty space problem in Section 2.1. We present our available, easy-to-use generator for data containing subspace clusters in Section 2.2 ([20]). In Section 2.3, we develop the scoring method KISS [15]. KISS indicates the importance of every dimension for each point individually, enabling clustering methods to find subspace clusters.

1.3 Correlation Clustering

Regarding high-dimensional data, there may be not only irrelevant attributes but also redundant or interdependent attributes. Where the former are handled via *feature selection* and subspace clustering as explained in the previous section, we regard the latter in the following.

On the one hand, attributes correlated to other attributes could be summarized to decrease the dimensionality of the data and with it the complexity. This is referred to as *feature extraction*. On the other hand, merging several attributes can impair the explainability of subsequent results as the gathered attributes usually have a real-world meaning, where attributes composed of other attributes are rarely intuitively interpretable for humans. Furthermore, depending on the data, finding the correlations may be the focus of interest as they may provide further insights into the data, independently of finding groups contained in the data.

Principles of Correlation Clustering Points of a *correlation cluster* are close when projected onto the corresponding arbitrarily oriented subspace. This projection can be done using the principal components of the cluster. Correlation clusters may not only reveal differently strong correlations between different attributes but also the number of involved respectively correlated attributes, i.e., the dimensionality of the related hyperplanes may differ.

There are several notions of correlation clustering:

- Attributes can correlate in diverse ways and for diverse reasons. Even though for most use cases correlated attributes are linearly correlated, for some use cases data sets may contain, e.g., hyperparaboloid [57] or periodically correlated data [56]. In the following, we regard exclusively *linear correlation clustering*.

- Even though the correlation of attributes does not depend on the spatial proximity of points constituting the correlation, many algorithms in the field aim at finding density-based correlation clusters resp. *correlation connected clusters* [27] , e.g., 4C [27], COPAC [2], and HiCO [3]. This probably originates in the notion of "classical" clusters, such that the found clusters are basically clusters of points revealing a correlation between some attributes.

- In the field of graph mining, the term correlation clustering is also used to describe graph partitioning without knowing the number of clusters [10], but is not directly connected to the task we regard in this thesis.

Even though finding global correlations between attributes is a quite straight-forward task, it becomes difficult if the data contains several clusters. To find correlations over the full data set most frequently *principal component analysis* (PCA) [51] is used.

Principal component analysis (PCA) [51] is a central element in correlation clustering. The principal components of a data set are the eigenvectors of the covariance matrix of the data and give the direction of the main variance of the points. The eigenvalues indicate the strength of correlation between attributes in the complete data space. However, if the data set contains several clusters constituting potentially different correlations between attributes as shown in Figure 1.2, these cannot be detected directly by PCA.

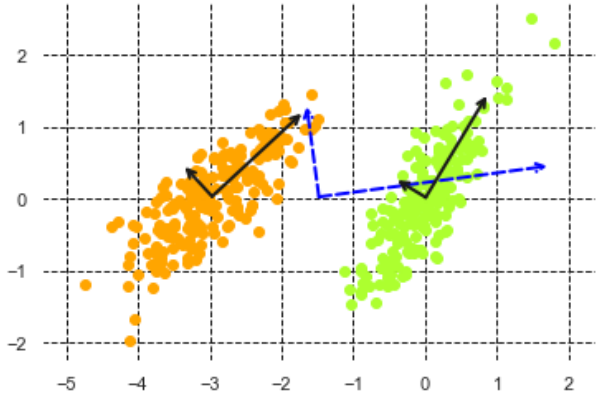


Figure 1.2: For this data set PCA yields the principal components indicated by the blue dashed arrows. The black solid arrows correspond to the principal components of the single clusters.

Evaluation Evaluating the quality of a correlation clustering proves difficult. There are virtually no completely labeled real-world data sets containing correlation clusters, thus an external evaluation can only be performed using synthetic data. Furthermore, there are no internal evaluation measures for correlation clustering. Internal evaluation measures for clustering as introduced in Section 1.1 cannot be applied to correlation clusters, as the fundamental models differ. Thus, we develop SRE [54] (*S*um of subspace *R*econstruction *E*rrors), the first internal evaluation measure for arbitrarily oriented subspace clustering in Section 3.1. The SRE is based on ideas originating from the field of autoencoders, assuming that points belonging to a correlation cluster have a low reconstruction loss after "encoding" and "decoding" them. The "encoding" and "decoding" corresponds in this case to projecting them onto their associated arbitrarily oriented subspace and retransforming them into the original full-dimensional space. Besides the reconstruction loss, the SRE also considers the subspace dimensionality and the number of clusters.

A bridge from clustering to correlation clustering Independently of the evaluation, the task of correlation clustering itself is complex. Similarly to subspace clustering as introduced in Section 1.2, correlation clustering algorithms simultaneously detect points building a cluster as well as the arbitrarily oriented subspace in which the points' projections are close. There are many algorithms designed to find correlation clusters, but, as described in Sections 3.2 and 3.3, most of them have several disadvantages. For example, they often need hyperparameters that are unintuitive and hard to choose well, have a high complexity (e.g., CASH [1] is in worst case exponentially complex), or cannot find correlations of different dimensionalities or correlation clusters that are not density connected. Especially for very high-dimensional data, the concept of density-connection becomes questionable regarding the curse of dimensionality and loss of meaningfulness for distance measures. Nevertheless, it depends on the use case which notion of correlation clusters is desired by the user.

With our work LUCK [18] resp. its successor LUCKe [23] (see Chapter 3) we build a general bridge between the two fields clustering and correlation clustering. Arbitrary purely distance-based clustering algorithms can be applied in order to find correlation clusters if they use distances as calculated by LUCKe instead of the usual Euclidean distance matrix. For that, we developed distance measures that are described in detail in Sections 3.2 and 3.3. They assign small values to pairs of points that are probably part of the same correlation, such that common distance-based clustering algorithms assign them subsequently to the same cluster. Where LUCK uses an orientation vector to allow finding one-dimensional linear correlations, LUCKe is based on local PCA (i.e., a PCA on the k nearest neighbors) and allows finding correlation clusters of arbitrary dimensionality. For that, we regard the similarity of directions of vectors indicating the main orientation of the kNN to each other as well as to the vector connecting the points. The more similar, the smaller are the values of our distance measure, indicating a high probability for the points to belong to the same correlation cluster. Our sophisticated distance measures are based on scalar products and are described in detail together with an analysis of their properties in Sections 3.2 and 3.3.

Some correlation clustering algorithms like 4C [27], COPAC [2], or ORCLUS [4] already combine PCA with the clustering algorithms DBSCAN [36], or k-Means [63, 64]. Nevertheless, the underlying clustering algorithm is not interchangeable and the results do not fit our above description of a correlation cluster. E.g., the DBSCAN-based approaches find *correlation connected clusters* [27], i.e., "a dense region of points in the d-dimensional feature space having at least one principal axis with low variation along this axis" [27]. In contrast, LUCK(e) is not dependent on the spatial proximity of points belonging to a correlation cluster in high dimensionality. ORCLUS needs user inputs regarding the dimensionality of the clusters as well as their number, while LUCKe automatically takes all principal components as well as their eigenvalues into account. LUCKe is the first method that allows to generically transfer advantages from the plethora of distance-based clustering algorithms directly to the field of correlation clustering.

PCA and the single-link effect in clustering The other way around, we can use PCA, a central technique in the field of correlation clustering, to improve clustering. A common problem for density-based algorithms is that disadvantageously distributed noise can connect individual clusters, which is also known as the single-link effect. As only a few points suffice to destroy an otherwise clear cluster structure by forming a *chain* between at least two clusters, removing them from the clustering process improves the results significantly. In our papers [46, 47] introduced in Section 3 we show how to detect those points forming a chain between clusters using PCA. The method originally designed for DBSCAN [47] can also detect chains between clusters found by any clustering algorithm [46]. Those unwanted chains consist of merely enough noise points to be considered dense and their intrinsic dimensionality tends to be lower than the intrinsic dimensionality of actual clusters in the data set. Our methods [46, 47] use this assumption to detect points that may be part of a chain using PCA and subsequently check if a cluster of potential chain points indeed connects to other clusters. Only then those chain points are excluded from the data set for clustering.

Contributions We develop SRE [54], the first internal evaluation measure for correlation clustering in Section 3.1. With LUCK [18] and its successor LUCKe [23] we create a way to use any distance-based clustering algorithm to find correlation clusters in Sections 3.2 and 3.3. That enables us to transfer a plethora of advantages and benefits from clustering to the field of correlation clustering. The other way around, we use the central correlation clustering method PCA to solve problems caused by the single-link effect for (especially density-based) clustering algorithms in noisy data, see Sections 3.4 and 3.5 ([46, 47].

1.4 Outlier Detection

An outlier is "an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism" [45]. In real-world applications, *outlier detection* respectively *anomaly detection* can deliver valuable insights in the data: typical examples are fraud detection, hints to medical problems, or finding flaws in production pipelines. Outliers can corrupt diverse data mining results as many fundamental methods are not robust against them, e.g., the mean of a distribution can be shifted significantly by only one outlier, where the less frequently used median is more robust against singular very large values deviating from the rest of the data.

An outlier can be seen as a point that does not belong to any cluster [37]. However, many clustering algorithms assign *each* point to a cluster, distorting important features of the clustering. For example, only one outlier can lead to a cluster center returned by k-Means [63] being "outside" of the original cluster, as seen in Figure 1.3. Hence, there is a plethora of algorithms designed to detect outliers and several approaches exclude outliers from the data set before or during clustering to improve clustering results, e.g., k-Means−− [30]. However, especially without knowing the clusters beforehand, finding those outliers can be difficult for a variety of reasons:

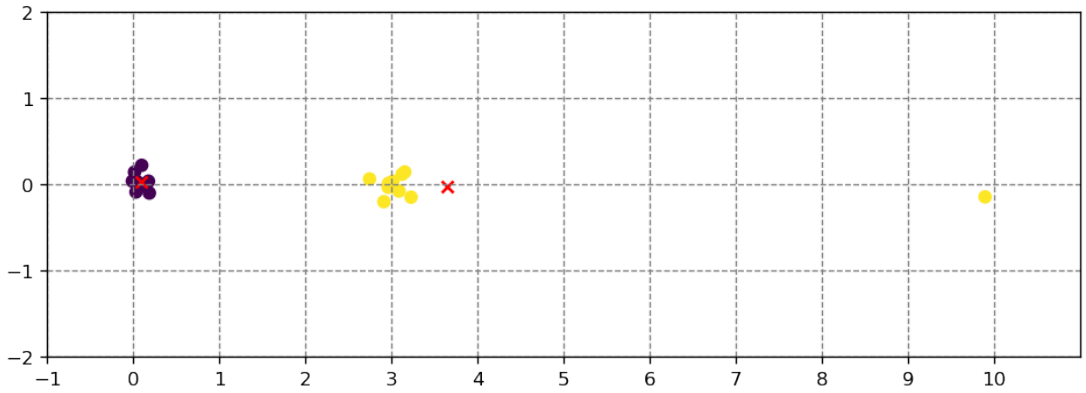


Figure 1.3: Data set colored according to clusters found by k-Means. Red crosses indicate cluster centers, i.e., the mean value of all points assigned to a cluster. Only one outlier (on the right) can shift the cluster center to a non intuitive position.

- As outliers are unexpected occurrences, there is usually no previous knowledge about them, making it hard to define what we are actually looking for.
- Depending on the use case, a certain degree of deviation might be expected and normal, where the same deviation may be considered an outlier for a different use case.
- It is difficult to differentiate between inconvenient noise and interesting outliers, especially in noisy or high-dimensional data sets containing several irrelevant dimensions
- Several aspects of the *curse of dimensionality* (see Section 1.2) exacerbate outlier detection further [82] in high-dimensional data .
- Other aspects include the origin of the anomalies, the time and context in which the data was gathered, and the availability of data sets with labeled outliers [29].

There are several fundamental approaches and a multitude of methods to detect outliers as well as several surveys categorizing them and listing advantages and disadvantages [29, 50, 73, 81, 82]. We can, e.g., differentiate between local (contextual) and global outliers, between scoring and labeling of outliers, unsupervised and supervised outlier detection, and many more.

Several outlier detection algorithms need the number or percentage of outliers as user input. As users usually want to find outliers without knowing them or the amount of outliers beforehand, a degree of outlierness can provide more information than a binary decision while simultaneously requiring less user input. Additionally, outliers are extraordinary occurrences and are thus usually examined further after their detection. Knowing and understanding why an object is detected as outlier helps subsequent examination, thus the explainable and clear algorithms are desirable. Additionally, as outlier detection and clustering are related to some extent, using similar ideas to clustering algorithms can

provide valuable results, as shown by outlier detection algorithms using k-Means (e.g., KMOR [41]), kNN (e.g., ODIN [44]), or the concept of density (e.g., LOF [28]).

We incorporate these requirements and ideas in our approach MORe++ [19], that gives a explainable score of outlierness based on k-Means clustering. In MORe++ we take advantage of clustering to improve outlier detection. By regarding outliers in every dimension separately instead of the full space we weaken some effects of the curse of dimensionality. As projecting onto only one dimension can mask interesting contextual outliers in the full space, we first partition all points according to their clusters as found by k-Means++ [8], a successor of k-Means [63, 64] and only then look at their one-dimensional projections. That gives us an outlier score based on the number of dimensions in which a point is an one-dimensional outlier. For high-dimensional data, this score can be more useful than a hard labeling, especially in the presence of subspace clusters, and is intuitively explainable also for non-specialists in the field. The use of histograms allows not only an additional speed-up but also refraining from high-dimensional distance measures as soon as clusters are found. Thus, our algorithm is able to process data sets containing many high-dimensional points relatively fast. All details on MORe++ can be found in Section 4.1 ([19]).

Taking a closer look at angle-based outlier detection approaches like ABOD [61] we see another potential for synergy with the clustering task. ABOD is the first outlier detection algorithm based on the observation that the angle enclosing all other points in the data set is rather small for global outliers compared to points lying in the middle of the dataspace. We refine this idea developing our clustering approach ABC [21]: Regarding the angle enclosing only the k nearest neighbors (kNN) of a point we note that points around the center of a cluster have comparatively high values where points at the border of a cluster have relatively low values. We use this to detect the points at the borders of clusters which we call *border points*. Typically, only a small fraction of the data set are border points, thus clustering them is significantly faster than clustering the complete data set. However, the border points alone can describe the clusters sufficiently. We can use adaptions of DBSCAN or hierarchical single-linkage clustering to summarize all points bordering a cluster. For that, we define a distance measure incorporating the previously calculated angles, which also supports separating close clusters. Given the clustering of the border points, the points in the inner of the cluster do not have to be regarded complexly but can simply be assigned to the cluster of the nearest border point. All details on ABC can be found in Section 4.2. ABC is the first algorithm that defines border points based on angles for clustering, where the usage of angles instead of distances tends to be more suitable for high-dimensional data [61].

Contributions We develop MORe++ [19], a fast and explainable outlier detection method for high-dimensional data that benefits from preceding clustering of the data in Section 4.1. We adapt the idea of regarding angles enclosing other points from the outlier detection method ABOD to enable detecting points bordering a cluster in Section 4.2. The resulting clustering algorithm ABC [21] is fast and suitable for high-dimensional data.

1.5 Ordering

In contrast to the previously regarded research areas, the ordering of input data is a rather neglected field from a data miner's point of view. Even though every algorithm uses *some* ordering of input data, its influence on the results is rarely investigated. Especially for clustering tasks in high-dimensional data, where the notion of similarity becomes difficult, meaningful orderings are valuable. If data is already ordered such that similar points are near to each other, the clustering task is simplified significantly as having an ordered list instead of a *set* of points brings several advantageous properties with it: the complexity decreases with the number of potential partitionings, ordered input leads to stable quality of results, and reproducibility increases. But also for other tasks, ordering of the input plays an important role, e.g., when working with sets of points as objects, as often used for data sets of 3d objects, when working on graphs, or for visual analytics. An order brings structure into the data, and structure helps to understand and learn from it.

Of course, there exists already a multitude of possibilities how to order tabular data, e.g., lexicographic ordering, or locality preserving orderings like Z-order [70] or Hilbert curves [48]. With a depth-first or breadth-first tree traversal index structures like R-trees [43] and its successors (e.g., R*-tree [14]), or k-d-trees [25] can be used to provide a meaningful ordering.

Also for ordering non-tabular data, especially nodes of a graph, a lot of effort has been made already. Famous methods to order the nodes of a graph are, e.g, by degree, depth-first or breadth-first graph traversal (where the root node has to be determined), or the Cuthill-McKee [32] algorithm minimizing the bandwidth of the graph's adjacency matrix. Nevertheless, all of these orderings have some drawbacks, especially w.r.t. clustering tasks as well as reproducibility, as elaborated in detail in our paper [16] presented in Section 5.1.

Depending on the reason why data should be ordered, and the goals that need to be achieved, ordering is a difficult task. A major problem in finding a good ordering is that there are virtually no internal quality indicators for orderings (besides the one we created, see below [22]). But even given such an indicator, finding a good ordering is complex: For a data set of n points, there are $n!$ possible orderings, so that simple brute-force methods are not applicable in everyday use.

Thus, we investigated orderings and diverse aspects of their impact in the field of data mining in our paper [16] and defined some necessary properties for internal measures of orderliness. Details elaborating on the problem, possible approaches, existing solutions as well as relations to other research areas can be found in Chapter 5.

Ordering data can simplify clustering, as we show with our clustering method CirClu. It orders the nodes of a graph meaningfully and subsequently partitions them in only linear time. For that, we used an observation from visual analytics approaches, where nodes of a graph are arranged in a circle [13, 65]. Where most works in this field aim at a well interpretable visualization of the graph or the minimization of edge crossings in the graph, we focus on a property suitable for clustering: having similar nodes as close as possible, i.e., having connected nodes as close as possible. For that, we evenly arrange all nodes of a graph on a circle and aim at reducing the edge lengths. In this context, we develop a

measure for the orderliness of nodes in a graph [22], called Circle Index (CI). It indicates if connected nodes are close to each other when arranged evenly on a unit circle. For that, the ratio between average edge length and a lower bound for the average edge length is regarded. The CI can be minimized with an iterative approach using some geometric basic knowledge, as explained in Section 5.2. For a graph $G = (V, E)$ with $|V|$ vertices and $|E|$ edges the the described approach is quite efficient with a complexity of $O(i \cdot |E| \cdot |V|)$. In the resulting ordering similar nodes that are likely to belong to the same cluster are near to each other, which is an advantageous base for clustering as we show with our clustering approach CirClu [22]: Keeping the order of the nodes reduces the number of possible bipartitionings from exponentially many to only linear many. As similar nodes are already near to each other, incrementally performing adapted normalized minimum cuts (which are then only linearly complex) leads to a comparably good clustering. Furthermore, our experiments on labeled real-world data show that a good quality of the ordering as measured by CI is related to a good quality of clustering based on this ordering. All details on the clustering algorithm as well as the Circle Index can be found in Chapter 5.

Contributions We reveal the need for research in the field of ordering data with respect to subsequent data mining tasks in Section 5.1 ([16]). We develop the Circle Index [22], a measure for the orderliness of nodes in a graph in Section 5.2. Optimizing this measure yields a beneficial order for subsequent clustering tasks, which we demonstrated with our new clustering algorithm CirClu [22].

1.6 Conclusion

In our research we investigated clustering with regard to a broad range of related tasks: subspace clustering, correlation clustering, outlier detection, and ordering. We examined synergies between them, found crossovers, refined basic assumptions, and enabled general transfer between some of the fields and clustering.

By using ideas from different research areas we were able to solve common problems in the fields that had no solution, yet. For example, we tackled the single-link effect occurring in the field of clustering by using a method from linear correlation clustering to detect chains of noise connecting clusters ([46, 47]).

We created possibilities to find correlation clusters ([18, 23]) as well as subspace clusters ([15]) with simple clustering algorithms. This reunification of research areas allows to transfer advantages of elaborated clustering techniques to the field of subspace clustering and correlation clustering.

Investigating benefits of refining common assumptions yielded useful and promising methods. For example, KISS [15] is based on the assumption that for each individual point different dimensions can be important, while most algorithms in the field interpret the *local feature relevance* with respect to groups of points. Also, we refined the idea of ABOD [61]. Instead of regarding the angle enclosing all points of the data set, we only regard the angle enclosing the kNN of a point. This adjustment allows us to differentiate

not only between global outliers and other points, but between inner points, border points, and outliers [21].

We did some fundamental, necessary work with the creation of our data generator [20], the development of the Circle Index [22] and the SRE [54]. Our generator for data containing subspace clusters facilitates and accelerates the work of scientists in the field of subspace clustering who want to evaluate and investigate newly developed algorithms on synthetic data. Reproducibly generatable data sets are a foundation of reproducible and proper research, especially when comparing different algorithms. Also the Circle Index, an internal measure for orderliness of nodes in a graph, enhances proper research by allowing to evaluate the ordering of nodes in a graph without a given ground truth. A low Circle Index indicates a good basis for subsequent clustering tasks. Similarly, internal evaluation measures are necessary to evaluate correlation clusterings on unlabeled real-world data. With the SRE we developed such a measure.

A broad range of future work can profit from our achievements. Research on the benefits of different orderings for data mining tasks is still in an early state of development. Suitable orderings of data can accelerate algorithms, improve their results, and increase reproducibility as well as flexibility in all fields working with data. Furthermore, LUCK(e) allows to transfer a variety of advantages of distance-based clustering algorithms to the field of correlation clustering. Thus, combining LUCK(e) with elaborated and advanced clustering algorithms may solve some open problems in the field of correlation clustering. For example, clustering algorithms that are robust to noise or use sophisticated acceleration methods can be used to detect correlation clusters in noisy real-world data sets while being faster than current methods. Similarly, combining KISS with such clustering algorithms may improve results in the field of subspace clustering analogously.

Requirements and demands on clustering algorithms may differ heavily depending on the area of application. Furthermore, selecting parameters that produce helpful results is a task that often requires expert knowledge not only in the field of application but also regarding the clustering algorithm itself. Incorporating the user in interactive clustering approaches can yield highly specialized results for diverse clustering tasks. Additionally, interactive approaches usually give a visual feedback that allows to select appropriate parameters without the need for expert knowledge in the field of data mining. Our first steps into this direction yielded promising results in the form of clustering-related analysis tools [53, 55, 69]. Thus, research on the edges between clustering and interactive algorithms as well as visual analytics could create further synergies in future work.

In this thesis, we simultaneously regarded related research areas as well as their similarities and differences which led to a deeper understanding of the areas. For the task of clustering, we were able to create synergies with related areas and enabled valuable advancements in data mining.

Chapter 2

Between Clustering and Subspace Clustering

This chapter includes the following publications:

1. Anna Beer, Daniyal Kazempour, Julian Busch, Alexander Tekles, and Thomas Seidl. "Grace - Limiting the Number of Grid Cells for Clustering High-Dimensional Data". In: *Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen" (LWDA)*. 2020, pp. 11–22. URL: http://ceur-ws.org/Vol-2738/LWDA2020_paper_11.pdf

2. Anna Beer, Nadine Sarah Schüler, and Thomas Seidl. "A Generator for Subspace Clusters". In: *Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen" (LWDA)*. vol. 2454. CEUR Workshop Proceedings. 2019, pp. 69–73. URL: http://ceur-ws.org/Vol-2454/paper_29.pdf

3. Anna Beer, Ekaterina Allerborn, Valentin Hartmann, and Thomas Seidl. "KISS - A fast kNN-based Importance Score for Subspaces". In: *Proceedings of the 24th International Conference on Extending Database Technology (EDBT)*. 2021, pp. 391–396. DOI: 10.5441/002/edbt.2021.40

2.1 "Grace - Limiting the Number of Grid Cells for Clustering High-Dimensional Data"

Publication: Anna Beer, Daniyal Kazempour, Julian Busch, Alexander Tekles, and Thomas Seidl. "Grace - Limiting the Number of Grid Cells for Clustering High-Dimensional Data". In: *Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen" (LWDA)*. 2020, pp. 11–22. URL: http://ceur-ws.org/Vol-2738/LWDA2020_paper_11.pdf

Statement of Originality: I had the idea and conceptualized the paper. Based on my instructions, Alexander Tekles implemented the method and performed the experiments within the context of his bachelor's thesis. Daniyal Kazempour wrote the related work section. Julian Busch helped shortening and improving the paper. The idea was discussed with Thomas Seidl, especially in the early development phase.

Erratum: Page 5, line 5: "A maximum of 50 bins" → "A *minimum* of 50 bins"

Grace – Limiting the Number of Grid Cells for Clustering High-Dimensional Data

Anna Beer, Daniyal Kazempour, Julian Busch,
Alexander Tekles, and Thomas Seidl

Ludwig-Maximilians-Universität München, Munich, Germany
{beer,kazempour,busch,seidl}@dbs.ifi.lmu.de
alexander.tekles@campus.lmu.de

Abstract. Using grid-based clustering algorithms on high-dimensional data has the advantage of being able to summarize datapoints into cells, but usually produces an exponential number of grid cells. In this paper we introduce Grace (using a *Gr*id which is *a*daptive for *c*luste*r*ing), a clustering algorithm which limits the number of cells produced depending on the number of points in the dataset. A non-equidistant grid is constructed based on the distribution of points in one-dimensional projections of the data. A density threshold is automatically deduced from the data and used to detect dense cells, which are later combined to clusters. The adaptive grid structure makes an efficient but still accurate clustering of multidimensional data possible. Experiments with synthetic as well as real-world data sets of various size and dimensionality confirm these properties.

Keywords: Grid-based, Clustering, High-dimensional

1 Introduction

Clustering is one of the most important and well investigated unsupervised data mining tasks. Nevertheless, some problems related to the curse of dimensionality are still not solved. Grid based approaches suffer not only from the exponentially increasing number of cells in relation to the number of dimensions, but also from the incoherence between data and grid structure. As many real-world datasets have high dimensional feature spaces, being able to handle many dimensions is quite important for clustering algorithms.

Even though subspace clustering algorithms focus on high-dimensional data, they assume clusters to be in a low dimensional subspace of the data and are thus not suitable to find clusters lying in the full-dimensional space. Density based approaches on the other hand find clusters in full-dimensional space where all dimensions are equally important, but cannot handle high-dimensional data. Thus,

efficient clustering of high-dimensional data without any a priori knowledge is still a huge challenge.

Hence we developed a grid-based clustering approach for high-dimensional data. It works fully automatically and finds clusters in full-dimensional data space without creating an exponential number of cells. The grid adapts itself to the data in regards of cell size as well as individual number of cells per dimension by using information from one-dimensional projections of the data. This leads not only to an adequate quality of the clustering results, but at the same time facilitates high efficiency of the algorithm.

Our main contributions are as follows:

- We develop Grace, a new grid-based clustering algorithm.
- By constructing the adaptive grid gradually, we are, to the best of our knowledge, the first ones to circumvent an exponential number of grid cells in relation to the number of dimensions.
- Grace is efficient and detects clusters of arbitrary shape in high-dimensional space.

The rest of the paper is structured as follows. Section 2 provides an overview of related work in the field of density-based and grid-based clustering. The algorithm itself is described in Section 3 and evaluated theoretically as well as empirically in Section 4. A brief conclusion is finally provided in Section 5.

2 Related Work

Since there exists a wealth of literature in the field of density-based as well as gird-based clustering, this section aims to provide an overview on some of the existing methods. We shall provide a brief elaboration on the core ideas behind each of the methods revealing the distinctive properties of our method in contrast to the competitors.

2.1 Density-based Clustering Approaches

Density-based clustering methods detect dense regions which are enclosed and separated by sparse regions and are thus suitable to find arbitrarily shaped clusters. Most density-based methods rely on local densities based on distances in the full-dimensional data space. The most common density-based approach is DBSCAN [7], which considers points with at least $minPts$ points in their ε-range as core points. Core points are connected if their distance is lower than ε and form a cluster. Not-core points either lie in the ε-range of a core point and get assigned to the cluster of the core point, or are declared noise. DBSCAN is quite sensitive to ε and $minPts$, which are two parameters hard to guess for a user without detailed knowledge of the data. OPTICS [3] improves DBSCAN by introducing a reachability plot based on $minPts$, on which users can see the cluster structure and choose appropriate ε.

DENCLUE [11] uses local densities to compute an overall density function, the maxima of which constitute density attractors. Every object is connected to such a density attractor by means of a hill-climbing procedure. A threshold ξ gives the minimum density level for a density-attractor which allows to find noise. To accelerate the calculation of local densities, a simple grid with the same cell width 2σ in all dimensions is used, where σ is a user given parameter.

Other density-based clustering algorithms usually build on the approaches presented so far and aim to improve or extend them. HDBSCAN [5] for example extends DBSCAN to a hierarchical approach allowing different levels of density that can detect clusters of different density or nested clusters, overcoming the aforementioned issue of one global hyperparameter setting. DeLiClu [1] and SOPTICS [15] represent algorithms with purposes similar to OPTICS, but with improvements regarding their efficiency. Likewise, DENCLUE 2.0 [9] is a straightforward improvement of DENCLUE with reduced runtime complexity.

2.2 Grid-based and Subspace Clustering Approaches

Grid-based clustering methods generally partition the data into cells of different densities by dividing each dimension into several intervals. Those cells can then be connected into clusters without having to look at each data point again, which decreases the runtime. The results are often highly dependent on the structure of the constructed grid and most algorithms require users to set the defining parameters. STING [14] proposes a quite interesting hierarchical grid structure based on statistical information, but is neither used for clustering, nor does it deliver exact values, but rather approximations. Also, the distribution type of the data has to be known or ascertained by hypothesis tests.

Most grid-based clustering techniques produce subspace clusterings i.e. they detect subspaces of a high-dimensional data space which contain clusters of the given data. Grid-based approaches are well-suited for this task because they can easily exploit the monotonicity of the clustering criterion regarding dimensionality. This criterion implies that a k-dimensional cell is dense only if every (k - 1)-dimensional projection of the cell is also dense, given a constant density threshold for the number of objects in a cell.

One of the first clustering algorithms to implement a grid-based subspace clustering was CLIQUE [2]. After constructing a grid with the same number of equidistant intervals in each dimension and identifying the dense cells, CLIQUE employs a bottom-up approach to find subspaces with dense regions by joining cells in k-dimensional spaces to candidate cells in $(k + 1)$ dimensions. If the number of objects in such a candidate cell exceeds a given density threshold, the corresponding $(k+1)$-dimensional space is considered a relevant subspace. After detecting the relevant subspaces, CLIQUE connects adjacent dense cells in their corresponding subspaces.

However, CLIQUE does not take the data distribution into account for generating the grid structure or for finding the dense regions. One subspace clustering approach that considers the data distribution beforehand is FIRES, which detects clusters on the basis of a greedy heuristics merging one-dimensional clusters

in order to find approximations of subspace clusters. This significantly reduces the runtime complexity of FIRES compared to CLIQUE.

While FIRES does not employ a grid structure at all, MAFIA [8] incorporates data distribution by using an *adaptive grid* in order to produce a better partitioning of the dimensions and reduce the number of grid cells. MAFIA determines the intervals in each dimension on the basis of one-dimensional histograms. Adjacent bins of a histogram are joined if they have approximately the same frequency. This yields larger intervals in the dimensions, each with roughly constant (one-dimensional) density. Nevertheless, MAFIA requires two parameters that may have a significant impact on the results and there is no guaranteed bound on the number of grid cells. On its adaptive grid, MAFIA proceeds like CLIQUE.

A general framework for clustering high-dimensional data on the basis of an adaptive grid is provided by OptiGrid [10] which recursively splits the data set by means of separating hyperplanes which should cut through low-density regions and separate high-density regions. The resulting cells already represent clusters, given a sufficiently high density. Though different approaches exist for selecting suitable hyperplanes [10, 6], these methods are not able to detect arbitrarily shaped clusters since generated cells already represent clusters and are not combined. Further, these methods require setting parameters whose impact on the result is difficult to assess a priori.

Further grid-based methods include SCHISM [17] which addresses the question of how to define and detect statistically interesting subspaces in high-dimensional data. As a measure for interestingness, the authors rely on the Chernoff-Hoeffding bound and use it for pruning. WaveCluster [18] relies on discrete wavelet transformation. The data is mapped to the frequency domain where clusters are then found by detecting dense regions. The method is insensitive to outliers and has a runtime complexity linear in the number of data objects. Among the most recent grid-based methods, ITGC [4] is an information theoretic approach regarding clustering as a data compression task. As such, neighboring grid cells are merged if it is beneficial with respect to compression costs.

3 Efficient Grid-based Clustering of Multi-dimensional Spaces

In this section we describe Grace in detail. In 3.1 we explain how the non-equidistant regular grid is generated dependent on the respective dataset. Section 3.2 shows how dense grid cells are combined to form clusters.

3.1 Generation of Adaptive Grids

The grid generation process is designed to limit the number of generated cells depending on the number of data points N and still allow for an accurate detection of clusters. For that we first estimate the density of each dimension

separately and then split the data space iteratively based on these estimations until the number of cells exceeds $N \cdot \log(N)$. To reduce runtime for calculating local changes in one-dimensional densities, we consider a histogram with $b = max(50, \sqrt{N/d})$ equi-width bins for every dimension instead of all points separately. A maximum of 50 bins for each histogram has shown to produce appropriate grid structures for various data distributions and different numbers of points N. Higher N imply possibly more complex shaped clusters requiring a higher granularity of the histogram. A higher number of dimensions d in contrast results in a lower number of bins, since high dimensionality implies less expressiveness of distances and less bins allow for higher deviations.

Estimation of Local Changes in One-Dimensional Densities Next, we compute for each bin in all one-dimensional histograms a *local change indicator* to express the local change of the one-dimensional density. To this end, we measure local changes in density as differences of frequencies between areas left and right to a particular bin and additionally set them in relation to the frequencies in their respective areas to distinguish random variations from relevant density shifts. The relevant neighborhood left and right of a particular bin is determined dynamically based on the frequency f covered by this area. The idea is to add less bins if the local density is already high, such that the separating hyperplanes will be lying closer together. Adjacent bins left and right are added iteratively until f exceeds a threshold t which is adjusted after each step. The threshold primarily depends on f and N such that a certain fraction of all objects needs to lie within the neighborhood range to stop expanding it. To avoid building large low-density cells, the neighborhood frequency is further weighted with the width of the current neighborhood range h, leading to a threshold

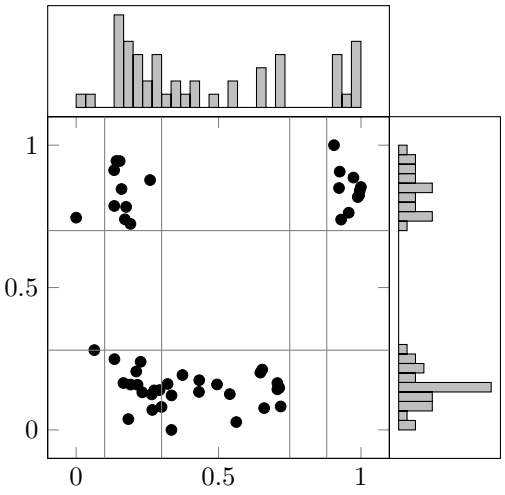


Fig. 1: Two-dimensional data with histograms as approximations of the one-dimensional data projections and edges chosen respectively

$$t = \frac{1}{h \cdot (1/b) \cdot (N/d)} \cdot N. \tag{1}$$

These weighted frequencies are used both for computing the differences between left and right areas as well as for determining the neighborhood area. As a

consequence, the density close to a bin has more impact on these computations than more distant bins. Figure 1 illustrates the generation of an adaptive grid based on the one-dimensional data projections as described so far.

Selection of Intervals After the one-dimensional histograms are computed and a change indicator is assigned to each bin, separating hyperplanes are selected iteratively until the number of cells generated by these planes is greater than or equal to $N \cdot \log(N)$. If d is high, not all dimensions are considered in order to limit the number of cells, i.e. some dimensions are not split by a separating plane. A dimension is only considered if at least two edges are chosen for this dimension. With only one edge in a dimension, i.e. two intervals, all data points would lie in the same interval or in adjacent intervals with respect to this dimension. In both cases, the corresponding dimension would have no informative content. This grid generation method yields at max $3 \cdot N \cdot \log(N)$ cells (see Theorem 1).

Theorem 1. *Given a d-dimensional data set with N objects. A regular grid in the d-dimensional space that is constructed by iteratively adding cutting hyperplanes, consists of at most 3N cells, if the grid generation process is stopped as soon as the number of cells is greater than or equal to N.*

Proof. Suppose h_{dim} edges are already chosen for dimension $dim \in 1, ..., d$. If $h_k \leq 1$, dimension k is not yet considered due to the minimum of two edges in a dimension for it to be considered. Let $D_c \subseteq 1, ..., d$ be the set of indices of the dimensions already considered. Dimension k is divided into $b_k = (h_k + 1)$ intervals. The number of cells c in the grid is computed by multiplying the number of intervals in all dimensions:

$$c = \prod_{dim \in D_c} b_{dim}$$

When adding a hyperplane separating dimension k to increase the number of cells from c to c', three cases can occur. Before adding the separating plane, $c < N$ holds.

Case 1: $h_k > 1$

If $h_k > 1$, the number of intervals on the corresponding coordinate axis increases by one as well. This yields:

$$\begin{aligned} c' &= \prod_{dim \in D_c} b'_{dim} = (b_k + 1) \cdot \prod_{dim \in D_c \setminus k} b_{dim} \\ &= \Big[b_k \cdot \prod_{dim \in D_c \setminus k} b_{dim} \Big] + \Big[1 \cdot \prod_{dim \in D_c \setminus k} b_{dim} \Big] \\ &\leq c + c = 2 \cdot c < 2 \cdot N < 3 \cdot N \end{aligned}$$

Case 2: $h_k = 0$

If $h_k = 0$, the number of cells does not increase at all, as dimension k is not considered yet and will not after this iteration. For the relationship between c and c' it holds, that: $c' = c < N < 3N$

Case 3: $h_k = 1$

If $h_k = 1$, k is a new dimension to be considered, as $h'_k = h_k + 1 = 2$ and $b'_k = b_k + 1 = 3$ respectively. The number of cells therefore increases by the factor 3. $h'_k = 2 \Rightarrow k \in D_c$

$$c' = \prod_{dim \in D_c} b'_{dim} = b_k \cdot \prod_{dim \in D_c \setminus k} b_{dim} = 3 \cdot \prod_{dim \in D_c} b_{dim} = 3 \cdot c < 3 \cdot N$$

□

Every bin of the initial histogram represents a potential edge for splitting. In every iteration, the bin with the maximum local change indicator is chosen as the next edge. To choose edges closer to cluster borders, we make a small adjustment after selecting an edge: The edge is shifted in one direction as long as two successive bins have a frequency difference below 5% and we choose the direction to which the edge needs to be shifted less. After selecting an edge, we discard all bins within the neighborhood if the selected bin from the set of potential edges. This step avoids high granularity of the grid in areas of density changes. Algorithm 1 summarizes the grid creation.

Algorithm 1 CreateGrid

```
b ← max(50, √(N/d))
for all dimensions do
    generate histogram with b bins
end for
for all dimensions do
    for all bins of histogram do
        determine neighborhood range
        compute local change indicator
    end for
end for
sort the bins of all histograms w.r.t. the local change indicator
moreEdgesNeeded ← true
selectedEdges ← {}
while moreEdgesNeeded do
    select the bin with highest local change indicator
    shift the bin if the adjacent bins have approximately similar frequencies
    add the edge to selectedEdges
    discard the bins within the neighborhood range of the selected bin from the set of
    potential edges
    if grid generated by selectedEdges contains more than N · log(N) cells then
        moreEdgesNeeded ← false
    end if
end while
```

3.2 Simple Connection of Dense Grid Cells to Clusters

Given the generated grid, the next steps involve detection of dense grid cells and subsequent combination of adjacent dense cells. For Grace, we apply wider notion of adjacency than existing works: Coordinates may differ at most by one in all dimensions – compared to a narrow notion, where the coordinates of adjacent bins may differ by one only in exactly one dimension.

A cell of volume V is considered dense, if it contains more than $minPts/V$ points for a $minPts$ given by the user. To avoid setting the density threshold too high for clusters with lower density, we first determine the most dense cells. To this end, we identify all cells containing more points that they would in expectation assuming a uniform distribution. In a second step, we discard these cells and detect the remaining dense cells with lower density using a new threshold.

Detection of dense cells is straightforward. After discretizing all data points to the grid, the number of data points in each grid cell is counted and compared to the threshold of the particular grid cell. Given an adequate grid structure, the number of dense cells p is usually much lower than N. Adjacent dense cells are now connected to form clusters by extracting connected components from the graph represented by the symmetric $p \times p$ adjacency matrix M with $M_{i,j}$ if and only if cells i and j differ by exactly one dimension. Adjacent cells can be found by sorting the dataset in every dimension and then iterating over the dimensions.

3.3 Connection of Diagonally Adjacent Grid Cells

So far, only adjacent cells in the narrower sense have been connected. To connect cells adjacent in the wider sense, i.e., diagonally adjacent cells, we add additional helper cells next to dense cells. Given a cell $\tilde{a}$, that is either a dense cell or a previously added helper cell with coordinates $(a_1, a_2, ..., a_d)$ and the order of dimensions considered for the current sorting of the coordinates $d_{i_1}, d_{i_2}, ..., d_{i_d}$, a new helper cell $(b_1, b_2, ..., b_d)$ with $b_k = a_k \forall k \in \{i_1, ..., i_{d-1}\}$ and $b_{i_d} = a_{i_d} + 1$ is now added to the set of helper cells if it has not yet been added before. New helper cells are not considered in the same iteration they were added, but in subsequent iterations they are treated just like the original dense cells. This ensures to find exactly all connections between originally adjacent dense cells in the wider sense.

Theorem 2. *Given a d-dimensional grid with a set P of dense cells and an initially empty set of helper cells H, both of whom are iterated d times. In every iteration i, a cell b with coordinates $b_k = a_k \forall k \in \{1, ..., d\} \setminus i$ for each cell $a \in P \cup H$ that has no adjacent cell with the coordinates of b is added to the set of helper cells after the current iteration. With this procedure, two dense cells that are adjacent in the wider sense can be connected, either directly or indirectly with the help of other cells in $P \cup H$, by just applying the notion of adjacency in the narrow sense.*

Proof. Given two dense cells a and b with coordinates $(a_1, a_2, \ldots, a_d)$ and $(b_1, b_2, ..., b_d)$, respectively.

Case 1: a and b are adjacent in the narrow sense
Adjacency in the narrow sense implies adjacency in the wider sense by definition, thus the two cells are also adjacent in the wider sense.

Case 2: $a_i \in \{b_i, b_i - 1\}, \forall i \in \{1, ..., d\}$
Suppose the cells' coordinates differ in dimensions $\{j_1, j_2, ..., j_m\} \subseteq \{1, 2, ..., d\}$ with $j_s < j_t \ \forall s < t$. In iteration j_1, the cell $a_{(1)}$ with coordinates $(a_1, ..., a_{j_1} + 1, ..., a_d)$ is either added as helper cell or already a dense cell. Since this cell differs by one from a in exactly one dimension, it is adjacent to a in the narrow sense. In iteration j_2, the cell $a_{(2)}$ with coordinates $(a_1, ..., a_{j_1} + 1, ..., a_{j_2} + 1, ..., a_d)$ is again either added as helper cell or already a dense cell. The new cell is now adjacent to the previously added cell $a_{(1)}$. This step is then repeated for all $j \in \{j_1, j_2, ..., j_{m-1}\}$. Finally, the cell $a_{(m-1)}$ with coordinates $(a_1, ..., a_{j_1} + 1, ..., a_{j_2} + 1, ..., a_{j_{m-1}}, ..., a_d)$ is either added as helper cell or already a dense cell. $a_{(m-1)}$ differs in exactly one dimension from b, so that $a_{(m-1)}$ and b are adjacent in the narrow sense and thus a and b.

Case 3: $a_i \in \{b_i, b_i + 1\}, \forall i \in \{1, ..., d\}$
Switching a and b converts this case to the same as case 2.

Case 4: $a_i \in \{b_i, b_i - 1\}, \forall i \in I \subset \{1, ..., d\}$ and $a_j \in \{b_j, b_j + 1\} \forall j \in J = \{j_1, j_2, ..., j_m\} \subset \{1, ..., d\} \setminus I$
In this case, two chains of adjacent cells can be constructed, each following the same idea as in case 2 or in case 3 respectively. The first chain starts from cell a, considering all dimensions with $a_i = b_i + 1$, which corresponds to case 2. The second chain starts from cell b, considering all dimension with $a_i = b_i - 1$, which corresponds to case 3. Finally, without loss of generality, the coordinates of the last cell $\tilde{a}$ in the first chain are of the form $(\tilde{a}_1, \tilde{a}_2, ..., \tilde{a}_d)$ with $\tilde{a}_i = \tilde{a}_i + 1 = b_i \forall i \in I$ and $\tilde{a}_k = b_k \forall k \in \{1, ..., d\} \setminus I$. The coordinates of the last cell $\tilde{b}$ in the second chain are of the form $\tilde{b}_1, \tilde{b}_2, ..., \tilde{b}_d$ with $\tilde{b}_j = \tilde{b}_j + 1 = a_j \forall j \in \{j_1, ..., j_{m-1}\}$, $\tilde{b}_{j_m} = a_{j_m} + 1$ and $\tilde{b}_k = a_k \forall k \in \{1, ..., d\} \setminus J$. Now, these two last cells of both chains differ only in dimension j_m by one, so that they are connected in iteration j_m.

□

4 Evaluation

We investigate Grace from a theoretical as well as from an empirical point of view. In Section 4.1 we calculate the runtime complexity and in 4.2 we present and discuss several experiments based on synthetic as well as real world data sets and compare the results with DBSCAN and CLIQUE.

4.1 Complexity Analysis

The histograms for each dimension can be computed in time $O(N \cdot d)$. For each of the $b \cdot d$ histogram bins, the local change indicator can be computed in constant

time, leading to $O(b \cdot d)$. Having $b \leq \sqrt{N \cdot d}$, we get $O(\sqrt{N \cdot d} \cdot d) = O(\sqrt{N} \cdot d^2)$. Finding separating hyperplanes requires sorting the $b \cdot d$ local change indicators, which can be done in time $O(b \cdot d \cdot \log(b \cdot d)) = O(\sqrt{N \cdot d} \cdot d \cdot \log(\sqrt{N \cdot d} \cdot d)) = O(N \cdot d^2)$.

The dense cells are determined by counting for every point the occurrences of each coordinate combination, which is in $O(N \cdot d)$. Dense cells can be combined efficiently by sorting them in every dimension to identify adjacent grid cells in that dimension. Since the maximum number of dense cells is $N \cdot \log(N)$, the complexity of sorting the dense cells is $O(N \cdot \log(N) \cdot \log(N \cdot \log(N)) \cdot d) = O(N \cdot \log^2(N) \cdot d)$. Adjacent dense cells can be identified in each dimension by comparing consecutive cells in the sorted order with complexity $O(N \cdot d)$. Thus, all connections between dense grid cells can be detected in time $O(N \cdot \log^2(N) \cdot d)$. Connected components can be identified in time $O(N^2 \cdot \log^2(N))$. In total, the complexity is thus

$$O(N \cdot d + \sqrt{N} \cdot d^2 + N \cdot d^2 + N \cdot \log^2(N) \cdot d + N^2 \cdot \log^2(N)) = O(N^2 \cdot \log^2(N) \cdot d^2).$$

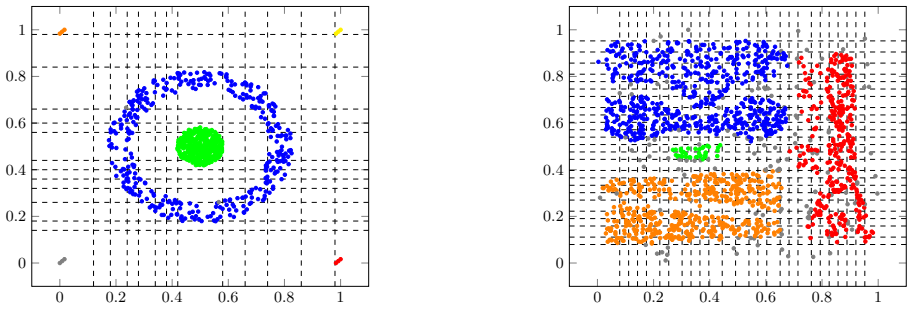
Note, that the number of dense cells p which is responsible for the $N^2 \cdot \log^2(N)$ part is usually far lower than N.

4.2 Empirical

The following experiments have been conducted on a Linux machine with a commodity hardware featuring a 2.0 GHz CPU with two cores and 3.6 GB RAM. As Grace uses elements from density-based as well as grid-based clustering, we compare our results to those DBSCAN and CLIQUE. Where Grace works fully automatically, DBSCAN and CLIQUE both need two parameters, for which those yielding the best results were chosen. For DBSCAN we used the scikit-learn Python library implementation and for CLIQUE the implementation from the data mining framework ELKI [16].

For a first visual interpretation, we show the effectivity of Grace working on two simple two-dimensional synthetic data sets containing density-based clusters and compare the results to those of DBSCAN. Figure 2 shows the clustering result for "TARGET" [19] with $N = 770$, consisting of four small clusters distributed at opposing corners as seen in Figure 2, one of them being detected as noise (bottom left cluster). Apart from the detected noise cluster, all other clusters are detected correctly by Grace. With a proper parameter setting, DBSCAN is capable of detecting all clusters, too. "CLUTO-T8-8K" contains 8000 two-dimensional objects [12]. Applied to this data set, the Grace found some, but not all clusters similar to DBSCAN.

Another synthetic data set with $N = 101,000$ and $d = 9$ containing three spherical clusters and $1,000$ noise objects has been created to show the scalability of Grace. The clusters of this data set are found in mere 4.5 seconds, where, due to excessive memory consumption, neither DBSCAN nor CLIQUE could be applied to this data set with the machine used here. To compare at least CLIQUE with the proposed approach in high dimensions, another data set with



(a) TARGET dataset (b) CLUTO-T8-8K

Fig. 2: Clustering results using an adaptive grid (represented by the dashed lines). Noise is colored gray.

$N = 100,000$ and $d = 8$, also containing three spherical clusters, is clustered by the two algorithms. They both detect all three clusters, where Grace was almost three times as fast as CLIQUE (1.7 vs 4.8 seconds).

Finally, a data set derived from the "VICON" data set containing physical action data measuring human activity [13] has been tested. The actions are measured by means of nine sensors on different body parts, each emitting three-dimensional spatial data. In summary, this yields a data set with 27 dimensions. For this experiment, the two actions punch and handshake have been merged into one data set with 5045 objects. Again, neither the DBSCAN implementation nor the CLIQUE implementation used can be applied to this data set on the machine used due to excessive memory consumption. Grace detected an accurate clustering within 232ms, with one cluster being detected 100% and the other cluster being split with 92% of it being grouped in one cluster.

5 Conclusion

Grace finds clusters in multidimensional data spaces where points build a cluster if they are close in all dimensions. It generates an adaptive grid structure that makes it possible to reduce the runtime complexity significantly for multidimensional data spaces compared to similar grid-based approaches. The experimental evaluation has shown that the algorithm outperforms DBSCAN and CLIQUE for large data sets and high dimensions. Grace works fully automatically and can be applied to datasets of various sizes, dimensionalities, and cluster densities. For clustering high-dimensional data with all dimensions being relevant for forming the clusters, it is an efficient alternative to established algorithms. It is moreover a possibility to get some first insights if no information about the data is available yet, since no expert knowledge about the data is needed beforehand due to the absence of any parameters. In future work noise should be handled separately and we are also investigating the suitability for anytime results. The

grid construction is promising for many other applications, and could, e.g., be applied in context of arbitrarily oriented correlation clusters.

Acknowledgments

This work has been partially 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. Achtert, E., Böhm, C., Kröger, P.: Deliclu: Boosting robustness, completeness, usability, and efficiency of hierarchical clustering by a closest pair ranking. In: PAKDD (2006)
2. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. SIGMOD (1998)
3. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure. In: SIGMOD (1999)
4. Behzadi, S., Hinterhauser, H., Plant, C.: Itgc: Information-theoretic grid-based clustering. In: EDBT (2019)
5. Campello, R., Moulavi, D., Sander, J.: Density-based clustering based on hierarchical density estimates. In: Pei, J., Tseng, V., Cao, L., Motoda, H., Xu, G. (eds.) PAKDD (2013)
6. Chang, J.W., Jin, D.S.: A new cell-based clustering method for large, high-dimensional data in data mining applications. In: SAC (2002)
7. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD (1996)
8. Goil, S., Nagesh, H., Choudhary, A.: Mafia: Efficient and scalable subspace clustering for very large data sets. In: KDD (1999)
9. Hinneburg, A., Gabriel, H.H.: Denclue 2.0: Fast clustering based on kernel density estimation. In: IDA (2007)
10. Hinneburg, A., Keim, D.: Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In: VLDB (1999)
11. Hinneburg, A., Keim, D.: An efficient approach to clustering in multimedia databases with noise. In: KDD (1998)
12. Karypis, G., Han, E.H., Kumar, V.: Chameleon: A hierarchical clustering algorithm using dynamic modeling. IEEE Computer (1999)
13. Lichman, M.: UCI machine learning repository (2013), http://archive.ics.uci.edu/ml
14. Muntz, R., Wang, W., Yang, J.: Sting: A statistical information grid approach to spatial data mining. In: VLDB (1997)
15. Schneider, J., Vlachos, M.: Scalable density-based clustering with quality guarantees using random projections. DMKD (2017)
16. Schubert, E., Zimek, A.: Elki: A large open-source library for data analysis-elki release 0.7. 5" heidelberg". arXiv preprint arXiv:1902.03616 (2019)
17. Sequeira, K., Zaki, M.: Schism: A new approach for interesting subspace mining. In: ICDM (2004)
18. Sheikholeslami, G., Chatterjee, S., Zhang, A.: Wavecluster: A multi-resolution clustering approach for very large spatial databases. In: VLDB (1998)
19. Ultsch, A.: Clustering with som, u*c. In: WSOM (2005)

2.2 “A Generator for Subspace Clusters”

Publication: Anna Beer, Nadine Sarah Schüler, and Thomas Seidl. “A Generator for Subspace Clusters”. In: *Proceedings of the Conference on “Lernen, Wissen, Daten, Analysen” (LWDA)*. vol. 2454. CEUR Workshop Proceedings. 2019, pp. 69–73. URL: `http://ceur-ws.org/Vol-2454/paper_29.pdf`

Statement of Originality: While supervising Nadine Schüler’s master’s thesis we noted the necessity of the generator described in the paper. I conceptualized the generator, Nadine Schüler implemented it, and we discussed the paper with Thomas Seidl.

A Generator for Subspace Clusters

Anna Beer, Nadine Sarah Schüler, and Thomas Seidl

LMU Munich
{beer,seidl}@dbs.ifi.lmu.de
n.schueler@campus.lmu.de

Abstract. We introduce a generator for data containing subspace clusters which is accurately tunable and adjustable to the needs of developers. It is online available and allows to give a plethora of characteristics the data should contain, while it is simultaneously able to generate meaningful data containing subspace clusters with a minimum of input data.

Keywords: Data Generator · Subspace Clustering · Reproducibility

1 Introduction

Developing algorithms in the field of data mining is usually an iterative process in which a main idea is implemented and then tested on several use-cases or experiments containing a ground truth. Depending on the results of those, the algorithm is modified and a loop of alternately testing and improving the algorithm starts. If the same data or only a few data sets are used in several iterations of this cycle, we create overfitting algorithms. The fields in which such subspace clusters can occur are manifold and especially for gene expression data or other data with medical background, clusters are most often found only in meaningful subspaces. Nevertheless, the number of labeled datasets is limited, and datasets containing labeled subspace clusters are rare. So, instead of using the few real world labeled datasets to develop and improve a subspace clustering algorithm, artificial datasets, of which the ground-truth is known by construction, are often used. Additionally, we can generate datasets in such a way, that they emphasize the advantages of the algorithm and help to detect diverse properties which possibly emerged in the development process. Data generators simplify the cumbersome process of constructing new datasets by hand, and allow building reproducible data sets, which are versatile enough to produce a non-overfitting algorithm in the above described development cycle. Nevertheless, there are only few publicly available data generators and none for generating data containing subspace clusters, even though some are used in diverse subspace clustering papers, as described in Section 2. Thus, we developed a generator for data containing subspace clusters, which allows to determine a multitude of parameters and is described in Section 3. Section 4 concludes this short paper and gives ideas for future work.

2 Related Work

The quality of most subspace clustering algorithms presented in the last years is shown using synthetic data, the construction of which is usually not well described or not reproducible at all. Most authors created very elementary data generators, leading to a multitude of generators with too little setting options to construct datasets with reasonably predictable characteristics. Looking at a multitude of subspace clustering related papers, we found the following to describe their data generation process best: SubClu [KKK04], SURFING [BPR+04], CLIQUE [AGGR98], which uses the generator described in [ZM97], and a review of diverse subspace clustering algorithms [PHL04]. Further, ResCu [MAG+09] and INSCY [AKMS08] use the same generator as [KKK04]. While all of those generators allow the user to set the number of points and dimensionality of the dataset as well as the number and dimensionality of clusters explicitly or implicitly, some crucial aspects are missing in each. E.g., the density or variance of clusters can be set in SubClu and SURFING, but not in [PHL04] or CLIQUE. CLIQUE constructs clusters differently to the other generators, as the user defines hypercubes in which the uniformly distributed points are more dense than in the surrounding areas. Surprisingly, generating data with noise is only provided by the generator from CLIQUE. The other generators construct, similarly to ours, some Gaussian distributed clusters and have different properties: in SubClu and SURFING, no cluster can be clustered in the full dimensional space, but the authors do not describe how this is reached. In [PHL04], the values of the relevant dimensions for each instance in a cluster can be restricted, leading to hypercube-shaped clusters.

MDCGen [IZFZ], which is probably the most recent and a very elaborated generator especially designed for multidimensional data and also subspace clustering, does not provide the possibility that a point can belong to multiple clusters at once. Additionally, there are data generators introduced independently from the field of subspace clustering, but to the best of our knowledge none of them is able to construct data containing subspace clusters of arbitrary dimensionality. [MLG+13] gives an overview over some data generators for big data benchmarking, like Hibench, LinkBench, CloudSuite, TPC-DS, YCSB, BigBench and BigDataBench, and BDGS. MUDD [SP04] is a generator similar to those. They are designed to create big data sets with similar properties as some given real world data, but users cannot specify enough details to be able to expose the advantages and disadvantages of their algorithms in development. RAIL [KBS19] is an interactive generator concentrating on producing linear correlated data, but allows only constructing 3-dimensional datasets containing 2-dimensional planes.

3 The Generator

In contrast to the data generators described in Section 2, the work here presented offers to define a plethora of characteristics of the dataset to be constructed while

simultaneously allowing to generate meaningful datasets containing subspace clusters without having to think about parameters too much: It requires only three parameters for the general set-up: The number of points n, the number of dimensions dim and m, a flag determining if it is possible for a point to belong to more than one subspace cluster. If given only those three parameters, we proceed as follows: To restrict the number of subspaces generated we use a fixed number of cluster centers, determined by a random number k between 1 and $\sqrt{n}$. The clusters are then randomly allocated to a number of subspaces $< k$ and the number of points as well as the number of dimensions of all subspaces clusters is drawn randomly from a uniform distribution within the given limits.

Users can specify the properties of the data further by giving information for every subspace S, namely the number of points, dimensionality, and number of clusters in S. Additionally, the variance of each cluster can be given. Figure 1 shows how subspaces can be distributed. In this example, there are four different subspaces, of which the first contains two clusters, the second and third contain one cluster each, and the fourth contains three clusters. The last two points belong to no cluster at all, while all other points are in two clusters in different subspaces. If $m = false$, a point may only belong to exactly one cluster, we insert the given subspaces into the $n \times dim$ matrix as long as there are sufficient points not assigned yet. Points and dimensions not assigned to belong to a certain subspace cluster, are filled with uniformly distributed noise data and a 0 in the label-matrix giving the cluster-assignments. If $m = true$, subspaces are first assigned in the same way as described above, before points are assigned to a second subspace and obtain a second cluster membership (see Figure 1). This is again assigned by going through the points and if there are enough unassigned dimensions to meet the requested subspace dimensionality this point will become a member of the second subspace in addition to the first one. When the points belong to a subspace cluster, the values are drawn from a appropriate multidimensional Gaussian distribution function, the center and standard deviation of which can be given by users. The remaining values are again drawn from a uniform distribution function. Uniformly distributed noise points can be added. Our generator is online available under https://github.com/NanniSchueler/SubCluGen.git and outputs the data matrix as well as the label matrix.

4 Conclusion

In summary, we introduced a data generator especially designed for subspace clusters. It expects only three parameters: the size and dimensionality of the dataset as well as as boolean value determining if a point can belong to clusters in different subspaces. With that a fast construction of data is possible. Simultaneously, reproducible datasets with very specific properties can be designed by users to test algorithms they are developing for diverse characteristics. The generator is easy to use and we plan to extend it with even more possibilities, like, e.g., non-axis parallel subspace clusters or other distributions instead of Gaussian, in future work. Also a combination with RAIL or some of the men-

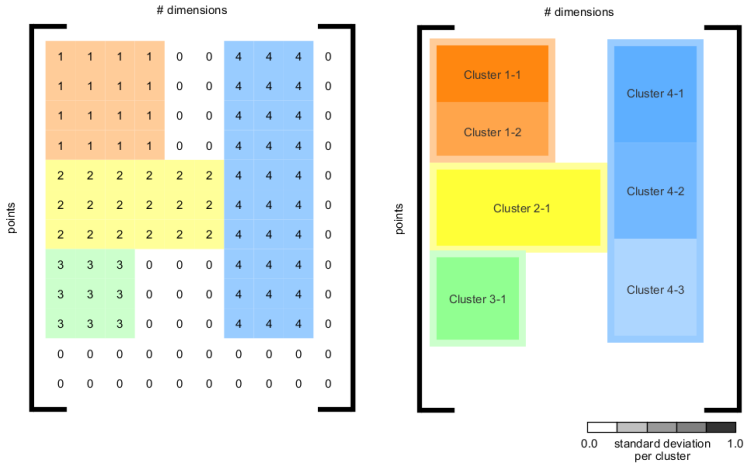


Fig. 1: Left: Label-Matrix as given by the generator as output, 0 implies uniformly distributed data, where numbers 1 to 4 imply the subspace affinity. On the right, the exact cluster affinity can be seen as well as the variance of the clusters implied by colour saturation.

tioned generators taking real world data into account could deliver a variety of reproducible datasets containing the desired properties for testing and developing.

Acknowledgement

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

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

[AKMS08] Ira Assent, Ralph Krieger, Emmanuel Müller, and Thomas Seidl. Inscy: Indexing subspace clusters with in-process-removal of redundancy. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 719–724. IEEE, 2008.

[BPR+04] Christian Baumgartner, Claudia Plant, K Railing, H-P Kriegel, and Peer Kroger. Subspace selection for clustering high-dimensional data. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 11–18. IEEE, 2004.

[IZFZ] Félix Iglesias, Tanja Zseby, Daniel Ferreira, and Arthur Zimek. Mdcgen: Multidimensional dataset generator for clustering. *Journal of Classification*, pages 1–20.

[KBS19] Daniyal Kazempour, Anna Beer, and Thomas Seidl. Data on rails: On interactive generation of artificial linear correlated data. In *International Conference on Human-Computer Interaction*, pages 184–189. Springer, 2019.

[KKK04] Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. Density-connected subspace clustering for high-dimensional data. In *Proceedings of the 2004 SIAM international conference on data mining*, pages 246–256. SIAM, 2004.

[MAG+09] Emmanuel Müller, Ira Assent, Stephan Günnemann, Ralph Krieger, and Thomas Seidl. Relevant subspace clustering: Mining the most interesting non-redundant concepts in high dimensional data. In *2009 Ninth IEEE International Conference on Data Mining*, pages 377–386. IEEE, 2009.

[MLG+13] Zijian Ming, Chunjie Luo, Wanling Gao, Rui Han, Qiang Yang, Lei Wang, and Jianfeng Zhan. Bdgs: A scalable big data generator suite in big data benchmarking. In *Advancing Big Data Benchmarks*, pages 138–154. Springer, 2013.

[PHL04] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: a review. *Acm Sigkdd Explorations Newsletter*, 6(1):90–105, 2004.

[SP04] John M Stephens and Meikel Poess. Mudd: a multi-dimensional data generator. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 104–109. ACM, 2004.

[ZM97] Mohamed Zaït and Hammou Messatfa. A comparative study of clustering methods. *Future Generation Computer Systems*, 13(2-3):149–159, 1997.

2.3 “KISS - A fast kNN-based Importance Score for Subspaces”

Publication: Anna Beer, Ekaterina Allerborn, Valentin Hartmann, and Thomas Seidl. “KISS - A fast kNN-based Importance Score for Subspaces”. In: *Proceedings of the 24th International Conference on Extending Database Technology (EDBT)*. 2021, pp. 391–396. DOI: 10.5441/002/edbt.2021.40

Statement of Originality: I had the idea and conceptualized the paper. Based on my instructions, Ekaterina Allerborn implemented the method and performed the experiments within the context of her master’s thesis. We refined some details in collaboration. Valentin Hartmann provided some mathematical perspectives. There were discussions on the idea with Thomas Seidl throughout the development process.

Short Paper

KISS — A fast kNN-based Importance Score for Subspaces

Anna Beer
LMU Munich
Munich, Germany
beer@dbs.ifi.lmu.de

Ekaterina Allerborn
LMU Munich
Munich, Germany

Valentin Hartmann
EPFL
Lausanne, Switzerland
valentin.hartmann@epfl.ch

Thomas Seidl
LMU Munich
Munich, Germany
seidl@dbs.ifi.lmu.de

ABSTRACT

In high-dimensional datasets some dimensions or attributes can be more important than others. Whereas most algorithms neglect one or more dimensions for all points of a dataset or at least for all points of a certain cluster together, our method KISS (***k***NN-based **I**mportance **S**core of **S**ubspaces) detects the most important dimensions for each point individually. It is fully unsupervised and does not depend on distorted multidimensional distance measures. Instead, the k nearest neighbors (kNN) in one-dimensional projections of the data points are used to calculate the score for every dimension's importance. Experiments across a variety of settings show that those scores reflect well the structure of the data. KISS can be used for subspace clustering. What sets it apart from other methods for this task is its runtime, which is linear in the number of dimensions and $O(n \log(n))$ in the number of points, as opposed to quadratic or even exponential runtimes for previous algorithms.

1 INTRODUCTION

As sensors in fields like biology and chemistry become more and more sophisticated, websites collect more and more data about their users, and IoT and manufacturing devices get equipped with sensors that allow for predictive maintenance, the amount, granularity and dimensionality of data increases. In order to still be able to analyze the data in a meaningful way and in a reasonable time, one often needs to reduce at least one of the three; this paper will focus on the dimensionality. The more dimensions there are, the more of them are not important and distort further data mining tasks. The more dimensions, the longer it takes to process them all: the running time of many algorithms increases exponentially with the number of dimensions, especially of those designed for fewer dimensions. But not only that: many distance measures become more and more useless with an increasing number of dimensions [4]. Thus, instead of dragging along all dimensions of a point, many methods focus on working on only a small subset of the dimensions. The dimensions that a point is reduced to should, of course, be the ones that capture the most relevant information that this point contains. But to learn those important dimensions proves difficult: users do not want to waste time studying the data and its features thoroughly before applying a data mining algorithm. However, most algorithms still require user input that needs expert knowledge, or even require the user to label data by hand. In addition, the number of possibly important subspaces is exponential, making it a complex and time-consuming task to find the most important one. As datasets grow in size and contain data from different sources, one part of a dataset might differ a lot from a different part. Nonetheless, methods for dimensionality reduction typically try to find a common subspace for all data points, which can potentially be completely unsuited for heterogeneous data. Often, every single point has its own properties and thus the importance of a subspace may vary for each point, as shown in Fig. 1: for objects 1,4,7 in the first column the texture may be relevant, whereas it does not seem to be important for the other objects, since every other texture only occurs once. Also, for objects 1,2,3 in the first row and the quadrangles 4 and 5 in the second row the number of corners may be important. The color could be the best attribute to distinguish objects 1,5, and 9 in the diagonal from the others. Thus, for object 1 all three considered dimensions — number of corners, color, and texture — may be relevant, while there are other objects in the same dataset for which not all of those dimensions are important, e.g., for object 7 only the texture is relevant.

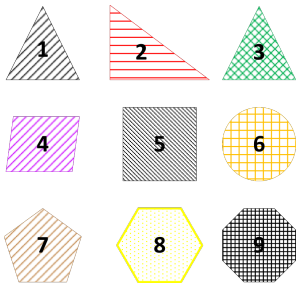


Figure 1: For different objects, different attributes or subspaces can be relevant: texture, number of corners, color, or a subset of those dimensions may be important.

A method to score the importance and expressiveness of each dimension for every point of a dataset individually without requiring any user input that scales to high dimensionalities would solve the problems mentioned above. In this paper, we develop KISS, a ***k***NN-based **I**mportance **S**core of **S**ubspaces, which fulfills all of these requirements. KISS can detect the most important subspace for a point fast and reliably in highly noisy data and data where only few dimensions are important per point.

One of the fundamental considerations that led to KISS is that those dimensions are most expressive for a point whose values lie in a cluster. We use the observation that if a point lies in a cluster in a certain subspace, the kNN of the projections of this point onto each dimension of this subspace intersect heavily. Since kNN in one-dimensional projections of the data can be computed fast, we

can efficiently calculate a score indicating the likelihood of the point lying in a cluster in the considered dimension. Usage of the kNN prevents relying on non-expressive distance measures, and there is no need for the users to know the data beforehand. KISS is deterministic, simple, fully unsupervised, and scalable w.r.t. the number of points as well as to the number of dimensions. It is easy to implement and reliably detects important dimensions for individual points fast. Our main contributions are as follows:

- We develop KISS, an importance scoring for every dimension for each individual point.
- KISS is fast w.r.t. both the number of points as well as the number of dimensions.
- KISS is fully unsupervised
- KISS does not rely on any multidimensional distance measure that gets useless for a high number of dimensions.

2 RELATED WORK

The problem of finding a global important subspace for all points has been addressed in previous work, which we introduce in Section 2.1. We restrict ourselves to algorithms that, like KISS (and contrary to, e.g., PCA or FOSSCLU), work in the standard basis of the vector space, as it simplifies getting insights into the data, which KISS was developed for.

2.1 Subspace Search

There exists some work on scoring of dimensions, where RIS, SURFING, and SCHISM are some of the most common algorithms.

RIS [6] produces a ranked list of all dimensions using a density-based quality criterion ("interestingness") that requires multiple parameters, which are set based on heuristic methods. The rating is only a relative comparison between different dimensions of the same dataset and is the same for all points.

SURFING [3] is a bottom-up approach that also returns the most "interesting" subspaces of a dataset. It is, like KISS, based on kNN, declaring subspaces as interesting in which "the k-nn-distances of the objects differ significantly from each other" [3]. The kNN distances are computed w.r.t. the subspaces, making their expressiveness dependent on the dimensionality of the subspaces. The algorithm has a runtime complexity of $O(mn^2)$, where n is the number of points and m is the number of different subspaces analyzed, which is 2^D in the worst case, making it much less scalable regarding both the number of points as well as the number of dimensions. Additionally, the minimum cluster size k has to be specified by the user.

SCHISM [8] extends the CLIQUE [2] principle and looks at the density of grid cells using an adaptive threshold function τ given by the user and applying the Chernoff-Hoeffding bound. It uses several preprocessing steps and requires three user given parameters u, τ, and ξ. Like RIS, and in contrast to KISS, it calculates a global score for "interesting" subspaces that is not adapted to individual points.

Although the dimension weightings at first glance seem to be suitable for comparing with KISS, such a comparison proves difficult: These dimension scoring methods do not return important subspaces for each point individually, or require at least two parameters set by the user, making it hard to objectively evaluate without overoptimism. But most notably, they are far more complex: The fastest of them, RIS, has runtime at least quadratic in the number of dimensions as well as the number of points. SCHISM is only linear in the number of points, but exponential in the number of dimensions. SURFING is quadratic in the number of points and exponential in the number of dimensions.

2.2 Subspace Clustering

We do not perform any clustering in this paper, but since we define "important dimensions" as dimensions in which a point lies in a cluster, there is a relation to the field of subspace clustering. Even though subspace clustering algorithms also deliver important dimensions in a way, their focus is different from KISS. Whereas those algorithms often need to perform a complete clustering of the dataset, we aim to get the relevant subspaces directly, individually for every point. We do not need to know the precise clusters to find important dimensions. Also, most of those algorithms rely on parameters that are not easy to set. We found that especially our first two goals, being fully unsupervised, and returning individual scores for different points, are to the best of our knowledge not achieved simultaneously by any other algorithm in this field. COSA and DISH are most related to our work, since they both consider subspaces for individual points:

COSA [5] finds important subspaces individually for each point using the kNN. A hierarchical clustering is applied based on a dimension weighting matrix and the relevant dimensions can be calculated based on the dimension weights of the respective cluster members. Despite the similarities to KISS, there are two major differences: First, users need to set a not quite intuitive parameter λ, which gives the "strength of incentive for clustering on more dimensions" [7]. Second, the kNN are calculated in the full-dimensional space, making COSA vulnerable to the loss of expressiveness of distance measures in high dimensions.

DiSH [1] is a density-based algorithm that finds cluster hierarchies and nested clusters. It has two parameters: a smoothing factor μ representing the minimum number of points in a cluster and ε for ε-range queries. Even though DiSH also uses only one-dimensional range queries and delivers subspace preference vectors for every point, the nesting of subspaces makes it impossible to determine the distinctly important subspaces. Also, the vectors are only calculated in an intermediate step, and depend on the parameter choices.

2.3 Possible Competitors

Finding suitable methods to compare KISS to is difficult: there are a number of subspace clustering algorithms, but they perform clustering, and not detection of the most important subspaces. For some algorithms one could extract the important subspaces of a point by looking at the subspace of the cluster the point was assigned to. This assignment, however, can only be obtained after an expensive clustering of the complete dataset. Some algorithms like, e.g., RIS or SURFING rank the subspaces in a similar way as KISS scores them, but they deliver one ranking for the complete dataset, not for each point individually. To the best of our knowledge, there is no algorithm yet that fulfills all of the requirements we impose. In particular, returning individual dimension ratings for each point and being completely unsupervised are very rare properties. Nevertheless, to at least have *some* point of reference, we exemplarily compare against CLIQUE, which is a grid-based bottom-up approach for subspace clustering. It requires two parameters, ξ and τ, which determine the number of intervals every dimension is partitioned into and the density threshold. We try out different parameter settings, showing that the results are very

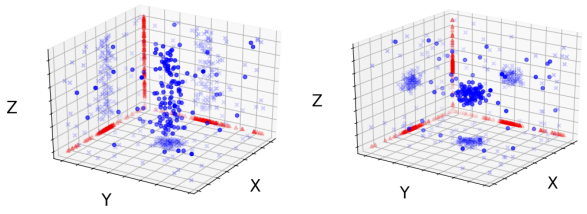


Figure 2: Projections of a 2-dimensional resp. 3-dimensional cluster. Blue crosses are projections onto the 2-dimensional subspace, red triangles projections onto one dimension.

sensitive to the parameter choices, whereas for KISS no parameters need to be tuned. In addition, we use the quality criterion of SURFING to obtain scores for every dimension and compare KISS to them. However, those are global scores for all points and not individual ones like those computed by KISS.

3 KISS

This section presents our newly developed dimension score KISS. We first describe the basic idea to use the k nearest neighbors in one-dimensional projections of the n data points to be able to compute KISS, which gives a scoring for the importance of every dimension for each individual point. Section 3.2 motivates our idea mathematically. In Section 3.3 we develop the exact formula of KISS, and present the complete KISS-based algorithm to obtain the most important subspace of a point. We analyze the complexity of our algorithm in Section 3.4.

3.1 Idea: Using One-dimensional kNN

If a point lies in a cluster in a d-dimensional subspace, most of the kNN of this point in the projection of the dataset onto those d dimensions will be members of that cluster, too. If we look at only one of those d dimensions (cf. the red triangles in Fig. 2), we still see the cluster structure: the cluster on the left lies in dimensions X and Y, and the red triangles on the according axes show a clear cluster structure. Projected onto those axes, most of the one-dimensional kNN of a point will lie in the same (original) d-dimensional cluster as the point itself.

Following this observation, for a given point p, we count for every other point q in how many dimensions it belongs to the one-dimensional kNN of p, giving us a Point Score $PS(p,q)$. A high Point Score means that q is likely to be contained in the same (higher-dimensional) cluster as p, meaning that the dimensions that they share carry more importance for p than the others. Summing up the Point Scores for each dimension individually gives us a measure for the importance of each dimension, where we account for outliers by incorporating the one-dimensional distances to the kNN of a point.

3.2 Mathematical Perspective

In the following we give some theoretical insights which support our idea. We denote cluster indices by superscripts and dimension indices by subscripts, and see clusters as collections of points drawn from a common probability distribution over $\mathbb{R}^D$.

Consider a cluster C^1 with center c^1 in dimensions $\{1,\dots,l\}$ (without loss of generality). We will consider the neighborhood of points in dimension 1. Let C^2 be a different cluster with center c^2 that overlaps with cluster C^1 in dimension 1, and assume that for all points $r^1 \in C^1$ and all $r^2 \in C^2$ we have $\Pr(|r_1^1 - c_1| \leq \varepsilon) \geq \delta$ and $\Pr(|r_1^2 - c_1| \leq \varepsilon) \geq \delta$, respectively, where $\varepsilon > 0$ and δ is a value close to 1. Furthermore, C^1 and C^2 should not overlap and be sufficiently far apart in the dimension that they share: $|c_1^1 - c_1^2| \leq 4\varepsilon + \varepsilon'$ for an arbitrarily small $\varepsilon' > 0$. This is, e.g., the case for all shared dimensions with high probability if c^1 and c^2 are uniform samples from a sufficiently large set in $\mathbb{R}^D$.
Let $p^1, \tilde{p}^1 \in C^1$, and $p^2 \in C^2$. In addition, let q be a point that does not lie in any cluster in dimension 1 and is drawn from a somewhat uniform distribution on a large enough interval. Precisely, we require it to fulfill $\Pr(|q - c_{1,2}| \leq 3\varepsilon) \leq \delta'$, where δ' is close to 0.
We now consider the distance of p^1 to the other three points in dimension 1.
For the point from the same cluster, we get

$$\begin{aligned}\Pr(|p_1^1 - \tilde{p}_1^1| \leq 2\varepsilon) &\geq \Pr(|p_1^1 - c_1^1| + |c_1^1 - \tilde{p}_1^1| \leq 2\varepsilon)\\ &\geq \Pr(|p_1^1 - c_1^1| \leq \varepsilon)\Pr(|c_1^1 - \tilde{p}_1^1| \leq \varepsilon) \geq \delta^2 \approx 1.\end{aligned}$$

The point from the other cluster yields

$$\begin{aligned}\Pr(|p_1^1 - p_1^2| \leq 2\varepsilon) &\leq \Pr(|p_1^1 - c_1^2| > \varepsilon \text{ or } |p_1^2 - c_1^1| > \varepsilon)\\ &\leq \Pr(|p_1^1 - c_1^2| > \varepsilon) + \Pr(|p_1^2 - c_1^1| > \varepsilon) = 2(1-\delta) \approx 0,\end{aligned}$$

where for the first step we observed that at least one of p_1^1, p_1^2 needs to lie outside of the ε-interval around its cluster's center, and applied a simple union bound to obtain the second inequality. Finally, for the point that does not lie in any cluster in dimension 1, we have, using the same arguments as above,

$$\begin{aligned}\Pr(|p_1^1 - q_1| \leq 2\varepsilon) &\leq \Pr(|p_1^1 - c_1^1| > \varepsilon \text{ or } |q_1 - c_1^1| \leq 3\varepsilon)\\ &\leq \Pr(|p_1^1 - c_1^1| > \varepsilon) + \Pr(|q_1 - c_1^1| \leq 3\varepsilon) \leq (1-\delta) + \delta' \approx 0.\end{aligned}$$

Thus, if k is chosen smaller than the size of C^1, the kNN of p_1^1 will almost exclusively consist of points from C^1. Thus,

$$\mathbb{E}(|\{i \in \{1,\dots,l\} \mid \tilde{p}_i^1 \text{ is part of the } k\text{-NN of } p_i^1\}|) \approx l\frac{k}{|C^1|} > l\frac{k}{n},$$

where the last quantity corresponds to a uniform distribution of points.

3.3 The Full Algorithm

KISS, the score indicating the importance of a dimension d for a point p in a dataset DB, depends on the k nearest neighbors (kNN) of p in the one-dimensional projection onto d: $kNN_p(d)$. Note that their number can be larger than k in case of ties, since we chose the deterministic variant of kNN.

The more often a point occurs in the sets of one-dimensional nearest neighbors of p, the closer related it is to p, which we capture in the *Point Score* $PS(p,q)$, where $\mathbb{1}$ is the indicator function: $PS(p,q) = \sum_{d=1}^{D} \mathbb{1}\{q \in kNN_p(d)\}$.

Higher values for k lead to more accurate scores, as shown in Fig. 3. However, the runtime of our algorithm depends on k and we would like to keep it $O(n\log(n))$ in the number of points (see our complexity analysis in Section 3.4). For this reason, k is set to $\sqrt{n}$, which is also in line with previous literature [5].

The importance a dimension d has for a point p depends not only on the intersection of the kNN in this dimension with the kNN in the other dimensions, but also on the distance of those kNN. Otherwise, the important dimensions for outliers would be distorted. Thus, the farther away a point in the kNN is, the less influence it should have on the importance of the respective dimension, which is why we divide the Point Score of each

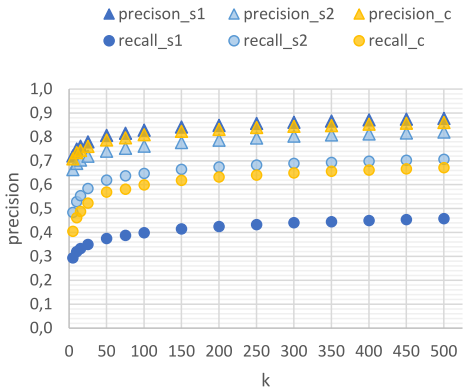


Figure 3: Results for different values of k. _s1, _s2, and _c denote different binarization methods, see Sec. 4.

$q \in kNN_p(d)$ by the distance between the corresponding projections of q and p. Additionally, the computed value is divided by the neighborhood size to account for ties among the nearest neighbors:

$$KISS'(p,d) = \frac{1}{|kNN_p(d)|} \sum_{q \in kNN_p(d)} \frac{1}{dist(p_d, q_d)} PS(p,q) \quad (1)$$

Finally, KISS is normalized for every point by dividing every value by the highest KISS occurring for the respective point:

$$KISS(p,d) = \frac{KISS'(p,d)}{\max_{e \in \{1,\ldots,D\}} KISS'(p,e)}. \quad (2)$$

This gives a value between 0 and 1 and allows for a meaningful comparison between different points of a dataset.

3.4 Complexity

The calculation of the kNN of all points in all dimensions needs $O(D*k*n + D*n*\log(n))$ steps, where D is the number of dimensions in the data set DB of size $|DB| = n$. Computing the kNN in one dimension can be performed efficiently by sorting the points w.r.t. this dimension and going to the left and right of the query point in the sorted list. Given the kNN of every point for every dimension, all Point Scores $PS(p,\cdot)$ w.r.t. a point p can be calculated in $O(D*k)$ by iterating through the k nearest neighbors of p in all D dimensions, keeping track of the scores via a hashmap where they get continuously updated. This has to be done for all n points, resulting in $O(n*D*k)$. For calculating the KISS for a point p and a dimension d, we need to sum up the Point Scores of all of p's kNN in d divided by their (one-dimensional) distance in this dimension, which can be done in $O(1)$. The summation can be performed in $O(k)$. We want to compute the KISS for all points and all dimensions, thus we get $O(n*D*k)$.

The KISS for all dimensions and all points can hence be computed in time $O(D*k*n+D*n*\log(n)+n*D*k+n*D*k) = O(D*n*(k+\log(n))$, which is linear in the dimension D and close to linear in the size of the dataset n. Runtime experiments confirmed this behavior, but were omitted due to space constraints.

4 EXPERIMENTAL EVALUATION

In Section 4.1 we introduce a technical tool that is needed to validate our method against a ground truth. In Section 4.2 we describe our experiments, and summarize the results in Section 4.3.

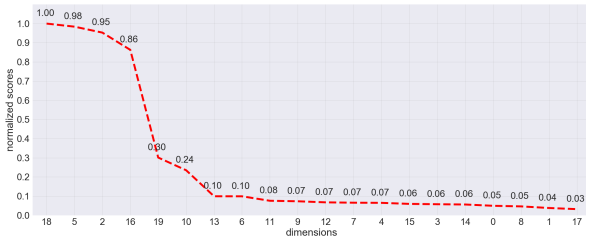


Figure 4: Typically distributed scores for different dimensions for a point p, sorted by descending normalized score.

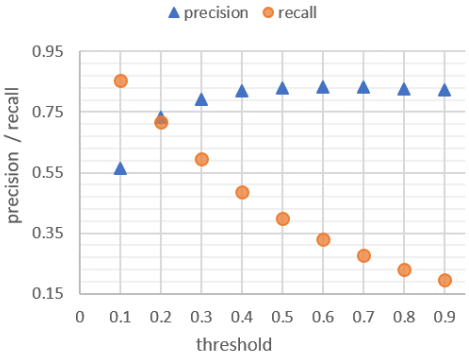


Figure 5: Precision and recall using simple binarization with different thresholds.

4.1 Binarization

To be able to validate our results and because for certain applications a division of the dimensions into important and unimportant ones can be needed, we suggest two possibilities to binarize the values obtained by KISS. Ordered by score value, a typical distribution of the scores for a point is shown in Fig. 4. If a point lies in a cluster, the KISS of the according dimensions clearly differs from the KISS of unimportant dimensions.

The naïve approach "simple binarization" of using a fixed threshold for the normalized score based on which we set the score to either 0 or 1, already delivers good results, as we show in Section 4.2. Fig. 5 shows recall and precision for our base case experiment and different values for the threshold, where 0.2 offers a good trade-off between the two. We performed the other experiments with the thresholds 0.5 and 0.2, denoted by _s1 and _s2, respectively.

Additionally, we developed a more sophisticated approach — "complex binarization" —, which comes with an only negligable increase in runtime, to improve our results even further. Here, we look for the most appropriate cut position in the ranked scores: e.g., for the KISS distribution depicted in Fig. 4 it could be dimension 12 since the scores for all dimensions to the right of it are significantly lower than the ones to the left of it. Our approach for detecting this cut position in the score ranking consists of first setting the importance of each dimension to 1 and then lowering it to 0 if its KISS lies below one of the three thresholds described below.

We set all parameters required for the complex binarization to the same reasonable values we give below for all experiments. Both strategies are introduced mainly to be able to validate our results against a binary ground truth. Note that the parameter values rely on the data being scaled to the D-dimensional unit

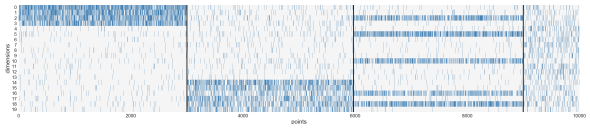


Figure 6: Transpose of the binary matrix for the base case dataset computed using KISS with complex binarization. The subspace boundaries are depicted as black lines.

hypercube. The three thresholds for the complex binarization are:

(1) *normalized threshold* t_n: If $KISS(p,d) < t_v = 0.1$, then $KISS(p,d)$ is set to 0.
(2) *unnormalized threshold* t_u: Because we normalize the KISS of each dimension by dividing by the largest KISS for this point, even a point that is just random noise has at least one dimension with score 1. However, the unnormalized value of dimensions with a high normalized KISS for this noise point will be significantly lower than the unnormalized values of dimensions with a high normlized KISS for a point that lies in a cluster. Thus, in addition to setting a threshold for the normalized KISS, we also set one for the unnormalized KISS' (cf. Equation 1): if $KISS'(p,d) < t_u$, the score for d is set to 0, where t_u equals the difference between mean and minimum of all unnormalized scores.
(3) *descent threshold* t_d: The descent threshold controls the decline between consecutive KISS values. If $\frac{KISS(p,e)-KISS(p,d)}{KISS(p,e)} > t_d = 0.7$, where $KISS(p,e)$ is the next largest KISS value of p, then all KISS values smaller than or equal to $KISS(p,d)$ become 0.

The result of the binarization can be expressed in a binary matrix as in Fig. 6.

Empirically, setting t_n significantly lower than 0.5 and t_d rather high allows for detecting more relevant dimensions and therefore detecting subspaces of higher dimensionality. t_u affects mostly how well outliers are detected. However, setting this parameter too high leads to very restricted binarized scores and can possibly decrease the detection rate of the important dimensions of the cluster points. In general, the parameters allow us to trade precision for recall. KISS is supposed to be used in settings where working with the original data without a significant reduction of the dimensionality is infeasible, either due to limited human capacity when manually analyzing the data or due to non-favorable dependence of a downstream task's performance on the number of dimensions. Hence we can live with a mediocre recall if in return the precision is high, allowing us to get rid of many dimensions, which is why we mainly focus on achieving a high precision.

In real-world settings where one needs a binary division of the dimensions, one typically has a (computational or storage) budget of dimensions one can deal with in the downstream task, and would binarize in a way so that exactly this many dimensions are labeled as important.

4.2 Experiments

We test with both the simple and complex binarization of KISS and denote the corresponding values with the abbreviations _s and _c, respectively. To have ground truth values for the importance of all dimensions, we generated data containing subspace clusters with possibly overlapping subspaces. The clusters are

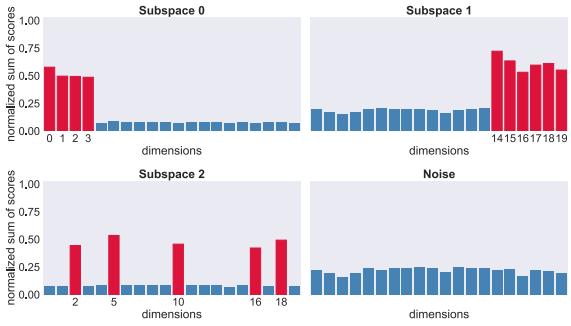


Figure 7: Average KISS for all points, partitioned according to ground-truth based important subspaces. Red bars show dimensions containing clusters.

Gaussians with mean randomly drawn from the uniform distribution on the D-dimensional unit hypercube. The values for the dimensions of a point that do not lie in a cluster are uniformly distributed in the hypercube [1].

Looking at the distribution of KISS per important (according to the ground truth) subspace in Fig. 7, we already see the correlation to the cluster subspaces: the average scores for the important dimensions (red bars) are visibly higher than those for unimportant dimensions (blue bars). Noise points that do not lie in any cluster are shown in the lower right diagram: the average KISS values do not differ much. The precision we achieve for both types of binarization are good, as can be seen in Fig. 8.

To the best of our knowledge there are no alternatives yet to KISS (see Section 2.3). However, with CLIQUE and SURFING we compare KISS to representative algorithms for subspace clustering and for subspace search. The comparison makes the disadvantages of having to set parameters as well as the benefit of individual scores in contrast to a global ranking clear.

Among others, Fig. 8 shows results obtained with CLIQUE for different parameter configurations. Even though CLIQUE was able to obtain high recall values, the precision was even for the best parameter settings much lower than for KISS (for both binarization methods). We also see that the results heavily depend on the choice of CLIQUE's parameters, with precision ranging from 35% to 77% and recall from 53% to 87%. Fig. 8 further includes the results obtained by binarizing the "quality" of each particular dimension as computed by SURFING for different values of k (in the same way as we binarize the KISS values). The classification performance of SURFING is very dependent on the parameter choice as well. With a good choice, it is able to achieve a high recall, but, as expected, the precision values are rather low, since the quality assignments are the same for all points, which does not match the ground truth.

Starting from this base case, we altered one parameter of the data in each of the following subsections to investigate KISS' behaviour w.r.t. this parameter.

4.2.1 Number of points. Changing the number of points n did not affect the precision, recall or accuracy of KISS significantly.

[1]The settings for our base case dataset are as follows: number of points $n = 10000$, dimensionality of point p: $dim(p) = 20$, percentage of noise $noise = 0.1$, set of dimensions in subspace S_i: $S_0 = \{0 \ldots 3\}$, $S_1 = \{14 \ldots 19\}$, $S_2 = \{2, 5, 10, 16, 18\}$, percentage of points w.r.t. n lying in subspace S_i: $|S_i| = [0.3, 0.3, 0.3]$, dimensionality of S_i: $dim(S_i) = [4, 6, 5]$, number of clusters in subspace S_i: $n_c(S_i) = [1, 2, 1]$, variance of cluster C_i: $var(C_i) = [1.5, 1.0, 1.3]$.

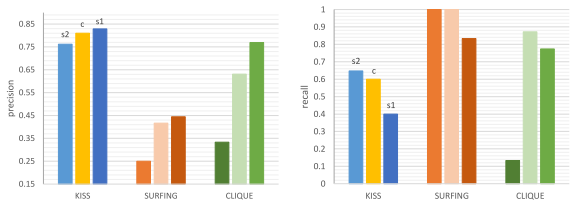


Figure 8: Results for KISS, SURFING, and CLIQUE for the base case dataset and different parameters ξ and τ resp. k. Same color indicates same parameters.

The three values deviated by at most 3% for $n \in \{5\,000, 10\,000, 25\,000, 50\,000, 75\,000, 100\,000\}$.

4.2.2 Number of dimensions. With growing number of dimensions (while keeping the ratio of dimensions lying in important subspaces the same) the recall decreases, but the precision, which we put more emphasis on, stays high.

4.2.3 Noise. Increasing the percentage of pure noise points leads to less points in each cluster, thus precision drops. Nevertheless, the decrease of quality is slow, and up to 50% of data can be pure noise points before precision falls below 75% (for the complex binarization).

4.2.4 Number of subspaces. We examine KISS for up to 10 different subspaces and obtain good results with precisions above 73% in all cases. Additionally, recall as well as accuracy diminish only slightly with increasing number of subspaces.

4.2.5 Subspace size ratio. We tested several size ratios between the three base case subspaces our dataset consists of. In our base case, every subspace contains one third of the non-noise data points. We set the share of instances in the first subspace to values $\{0.4, 0.5, 0.6, 0.7, 0.8\}$, while dividing the remaining points equally among the other two subspaces (and additionally keeping the 10% noise of the base case). The quality of the scores hardly changed: we received precision values for the simple binarization between 83% and 85%, and between 84% and 86% for the complex binarization. Recall values ranged between 39% and 41%, and 47% and 49%, respectively, showing that the size ratio of the subspaces does not constitute a problem for KISS. Thus, even subspaces containing only very few points of the complete dataset can be found as easily as bigger subspaces.

4.2.6 Number of clusters per subspace. With an increasing number of clusters, precision as well as recall decrease, since there are fewer points per cluster that could help identify a point in the cluster.

4.2.7 Density of clusters. We tested several density settings for the clusters. When all subspaces contain similarly dense clusters, the quality decreases with lower density (i.e., higher standard deviation). If each subspace contains differently dense clusters, the results are rather determined by the average density than by the lowest or highest occurring density. Thus, a large difference in cluster density does not influence the results negatively.

[1]When adding more subspaces to the base case dataset, we use the same settings as for the original subspaces: the points are evenly distributed among the subspaces, and the cluster settings of the clusters lying in subspaces S_{0+3i}, S_{1+3i}, S_{2+3i} correspond to the settings of the clusters lying in subspaces S_0, S_1, S_2.

4.3 Summary of Results

Our experiments show that KISS achieves a high precision and reasonable recall across a wide range of settings. With a high number of subspaces or clusters the performance starts to degrade, but KISS is robust to noise and can deal with high numbers of points as well as clusters of different density. We would like to point out that the experiments only show a small part of KISS' capabilities, since the original KISS is a continuous value, which we just binarized here, and likely not even optimally.

5 CONCLUSION AND FUTURE WORK

We developed KISS, a scoring that assigns an importance value to each dimension of each point of a dataset. It is scalable, does not suffer from the curse of dimensionality, since it replaces multi-dimensional distance measures by one-dimensional ones, and does not require significant user involvement to set parameters. Its runtime is linear in the dimensionality and close to linear in the number of points, setting it apart from similar methods.

KISS has numerous applications, both as a tool to get an insight into datasets as well as a foundation for data mining applications, in particular to accelerate downstream tasks or to make them more robust to noise. We are currently working on some of the most immediate extensions: (1) performing clustering using especially the most relevant dimensions for each point; and (2), using KISS for outlier and noise detection, following the observation that points that have a low KISS in every dimension are typically in none of those in a cluster. We encourage the usage of KISS for preprocessing data and gaining knowledge in an early stage of a data anlysis process, since it is simple, fast, delivers good results and does not require parameter tuning.

ACKNOWLEDGMENTS

This work has been partially 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] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, Ina Müller-Gorman, and Arthur Zimek. 2007. Detection and visualization of subspace cluster hierarchies. In *International Conference on Database Systems for Advanced Applications*. Springer, 152–163.

[2] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. 1998. *Automatic subspace clustering of high dimensional data for data mining applications*. Vol. 27. ACM.

[3] Christian Baumgartner, Claudia Plant, K Railing, H-P Kriegel, and Peer Kroger. 2004. Subspace selection for clustering high-dimensional data. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*. IEEE, 11–18.

[4] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When is "nearest neighbor" meaningful?. In *International conference on database theory*. Springer, 217–235.

[5] Jerome H Friedman and Jacqueline J Meulman. 2004. Clustering objects on subsets of attributes (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 66, 4 (2004), 815–849.

[6] Karin Kailing, Hans-Peter Kriegel, Peer Kroeger, and Stefanie Wanka. 2003. Ranking interesting subspaces for clustering high dimensional data. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 241–252.

[7] Lance Parsons, Ehtesham Haque, and Huan Liu. 2004. Subspace clustering for high dimensional data: a review. *Acm Sigkdd Explorations Newsletter* 6, 1 (2004), 90–105.

[8] Karlton Sequeira and Mohammed Zaki. 2004. SCHISM: A new approach for interesting subspace mining. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE, 186–193.

Chapter 3

Between Clustering and Correlation Clustering

This chapter includes the following publications and drafts:

1. Daniyal Kazempour, Anna Beer, Peer Kröger, and Thomas Seidl. "I fold you so! An internal evaluation measure for arbitrary oriented subspace clustering". In: *2020 International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2020, pp. 316–323. DOI: 10.1109/ICDMW51313.2020.00051

2. Anna Beer, Daniyal Kazempour, Lisa Stephan, and Thomas Seidl. "LUCK- Linear Correlation Clustering Using Cluster Algorithms and a KNN Based Distance Function". In: *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*. SSDBM '19. New York, NY, USA: Association for Computing Machinery, 2019, 181—184. DOI: 10.1145/3335783.3335801

3. Anna Beer, Lisa Stephan, and Thomas Seidl. "LUCKe — Connecting Clustering and Correlation Clustering". In: *2021 International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2021, pp. 431–440. DOI: 10.1109/ICDMW53433.2021.00059

4. Janis Held, Anna Beer, and Thomas Seidl. "Chain-detection for DBSCAN". in: *BTW 2019 - Datenbanksysteme für Business, Technologie und Web (Workshops)*. Gesellschaft für Informatik, Bonn, 2019, pp. 173–183. DOI: 10.18420/btw2019-ws-18

5. Janis Held, Anna Beer, and Thomas Seidl. "Chain-detection Between Clusters". In: *Datenbank-Spektrum* 19.3 (2019), pp. 219–230. DOI: 10.1007/s13222-019-00324-9

3.1 "I fold you so! An internal evaluation measure for arbitrary oriented subspace clustering"

Publication: Daniyal Kazempour, Anna Beer, Peer Kröger, and Thomas Seidl. "I fold you so! An internal evaluation measure for arbitrary oriented subspace clustering". In: *2020 International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2020, pp. 316–323. DOI: 10.1109/ICDMW51313.2020.00051

Statement of Originality: The concept was developed and implemented by Daniyal Kazempour. I formalized the method mathematically and helped improving the implementation. The experiments were designed in collaboration. Peer Kröger suggested further improvements. The main idea as well as experiments were discussed with Thomas Seidl.

I fold you so! An internal evaluation measure for arbitrary oriented subspace clustering

Daniyal Kazempour, Anna Beer, Peer Kröger, Thomas Seidl
Ludwig-Maximilians-University Munich
{kazempour, beer, kroeger, seidl} @dbs.ifi.lmu.de

2020 International Conference on Data Mining Workshops (ICDMW) | 978-1-7281-9012-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDMW51313.2020.00051

***Abstract*—In this work we propose SRE, the first internal evaluation measure for arbitrary oriented subspace clustering results. For this purpose we present a new perspective on the subspace clustering task: the goal we formalize is to compute a clustering which represents the original dataset by minimizing the reconstruction loss from the obtained subspaces, while at the same time minimizing the dimensionality as well as the number of clusters. A fundamental feature of our approach is that it is model-agnostic, i.e., it is independent of the characteristics of any specific subspace clustering method. It is scale invariant and mathematically founded. The experiments show that the SRE scoring better assesses the quality of an arbitrarily oriented subspace clustering compared to commonly used external evaluation measures.**

I. Introduction

Among the greatest challenges in developing unsupervised machine learning techniques, particularly clustering methods, is the evaluation of the results. In most works, the results of a new algorithm are compared against those of other algorithms based on a ground truth for the datasets they are tested on. This ground truth is not only rarely given for available datasets, but contradicts the goal of unsupervised algorithms. In cases where no ground truth is given or appropriate, internal quality measures are used. However, internal evaluation criteria assume that clusters have certain properties and measure how well these specific properties are fulfilled. Depending on the assumptions behind the measures, there is a strong bias towards certain types of clustering models, which we will elaborate on in Section II.

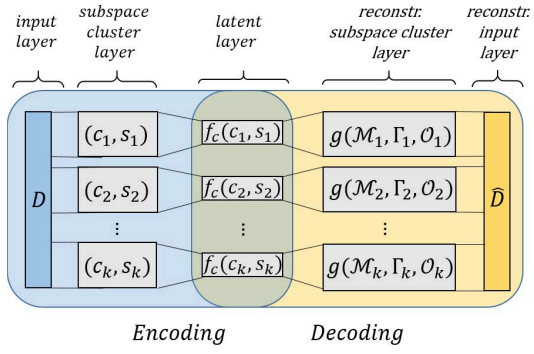


Fig. 1. Architecture of subspace clustering as a reconstruction task.

A very recent class of clustering methods is designed to detect clusters in arbitrarily oriented subspaces of a high-dimensional dataset, also known as correlation clustering methods[1] [1]. The main idea behind this type of clustering methods is to detect groups of objects that have a similar correlation among a given set of features and, thus, are located on a low-dimensional linear subspace E in ambient space. These objects are in fact similar to each other when projected onto the subspace perpendicular to E. While there exists a thorough evaluation of axis-parallel subspace clustering methods (a variant of the general problem described above assuming attribute independence) based on a plethora of different external criteria [2], there is, to the best of our knowledge, no internal quality measure for arbitrarily-oriented subspace clustering methods.

In this work, we introduce SRE (*S*um of subspace *R*econstruction *E*rrors), the first internal evaluation criterion for subspace clustering. It performs well as we show with experiments in Section IV and scores aspects of a clustering which are neglected by external evaluation measures yet important in context of arbitrarily oriented subspace clustering. It is model-agnostic, i.e., it does not have a bias towards a given clustering model (density-based, Hough-based, ICA-based arbitrarily oriented subspace clustering etc.). To achieve this, we propose a different perspective on the subspace clustering problem. Inspired by the concept of autoencoders [3] which have been also proposed for dimensionality reduction [4] we re-define subspace clustering as a task in which the objective is to partition a given dataset in ambient space. This segmentation is performed in such a way that the obtained subspace for each of the clusters allows a reconstruction of the data from this latent space with a minimized loss while at the same time reducing the model complexity, i.e. reducing the number of dimensions for the subspace and reducing the number of clusters. Autoencoders learn a lower-dimensional embedding into the latent space, which minimizes the reconstruction error regarding the original ambient space. The latent space of an autoencoder is an arbitrarily shaped manifold, whereas the result of a subspace clustering is a set of subspace clusters, where each is described by its individual arbitrarily oriented linear subspace. Thus, using our new definition of subspace clustering, it delivers a piecewise linear approximation of an arbitrarily shaped manifold within the data. The non-neural autoencoding architecture of our proposed method is illustrated in Figure 1. The approximation loss together with

[1]Not to be confused with correlation clustering in context of graph mining.

2375-9259/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDMW51313.2020.00051

regularization parameters for the number of clusters and the number of subspaces builds the new internal quality measure SRE. At this point one may object that by relying on the assumption that data is located on or around manifolds, we would have some kind of an "external" criterion. This however is not the case, since the manifold assumption addresses an *internal* property, like the silhouette-coefficient [5] also relies on the internal model assumption that the clusters are convex. The remainder of this paper is structured as follows: In Section II we introduce related work, in Section III we give a formalization for our new perspective on subspace clustering and the consequential internal quality measurement *SRE*, establishing a link to autoencoders. After investigating some properties of *SRE*, we describe the conducted experiments in Section IV and discuss in this context the feasibility of our internal measure. We conclude our work with the lessons learned and future prospects. In summary, our contributions are as follows: **(1)** We propose a model-agnostic internal evaluation measure for subspace clustering **(2)** We re-define the subspace clustering problem as an optimization task with the objective to minimize the reconstruction error from a piecewise linear approximation of a non-linear manifold, while at the same time reducing the model complexity.

II. Related Work

The so far existing internal evaluation measures such as the silhouette coefficient [5], the density-based validation index [6] or modularity [7] are limited to their respective underlying models, and as such come with their model bias. There exist however further internal evaluation measures such as the Davies-Bouldin score [8]. This score measures the average of the pairwise ratio of compactness and separation. Similarly the Calinski-Harabasz Score [9] and Dunn-Index [10], like many other existing internal evaluation measures, rely on properties such like compactness and separation. One drawback which comes with them, is the fact that for the compactness of the clusters the average distance of objects to their cluster centers is computed. This reliance on cluster centers introduces a certain bias. Further these internal evaluation measures neglect aspects like reconstruction quality, number of clusters or number of dimensions rendering them inadequate for our needs. Our SRE measure differs from the existing methods by integrating the reconstruction loss as well as the model complexity (number of clusters and dimensionality of subspaces). As such, we do not compare in this work SRE against other internal evaluation measures, since they would yield scores which may favor individual algorithms that rely on density or convex clusters, but not favor clusters which are "tight" around an arbitrarily oriented subspace and which do not penalize increased model complexity. In a very recent work [11], the authors introduce an approach for a holistic assessment of clustering algorithms with respect to their structure discovery capabilities. In that work they rely on criteria, namely stability, structure and consistency. In their work, the notions of stability and consistency rely on a density threshold ϱ. This introduces unfortunately a model bias, since subspace clustering algorithms which rely on density would be in favor. In the work of [2], the authors propose an evaluation method which is founded on three major paradigms, namely (a) cell-based, (b) density-based and (c) clustering-oriented paradigms. All three paradigms are incorporated in the framework of an *external* measure for subspace clustering which relies on ground truth. The method is limited to axis-parallel subspace clusterings. The authors state in the introduction of their work, that it would be beyond the scope of their work to include arbitrarily oriented subspace clustering algorithms.

III. Subspace Clustering as Manifold Learning

Seeking for an algorithm-agnostic internal evaluation measure for subspace clustering leads us to autoencoders, which inspired this work. According to [12][Ch. 14], an autoencoder is a neural network which is aimed at copying its input to its output by encoding the input first to a lower-dimensional latent layer (bottleneck) $h = f(x)$ and decoding it to its full-dimensional representation $g(f(x)) = x$. The limitation of the dimensionality of the latent layer is necessary to learn lower-dimensional representations of the ambient space. The learning process itself is expressed as a minimization of a loss function such as $L(x, g(f(x)))$, where L is a penalty function depending on the dissimilarity between the reconstruction $g(f(x))$ and the ambient space representation of x. Bridging over to manifold learning, according to [12], autoencoders rely on the assumption that data is located on or around lower-dimensional manifolds or subsets of manifolds. A manifold is characterized by its set of tangent planes where each of the tangent planes represents a local euclidean space. Autoencoders learn such manifolds while balancing between two aspects: (1) achieving a good approximation of a manifold (minimizing reconstruction error) and (2) satisfying regularizations such as, e.g., the dimensionality of the latent layer. It is noteworthy at this point that if we use a linear decoder and the mean squared error as a loss function, the autoencoder (if undercomplete) learns the same subspace as a PCA. Under this aspect, a PCA learns one global linear approximation of a potentially non-linear manifold of the ambient space. However, autoencoders with non-linear encoder and decoder are capable of learning non-linear manifolds. If we apply local PCA [13] on the data in ambient space, we obtain a set of linear lower-dimensional subspaces and the objects projected to their respective subspaces. Arbitrarily oriented subspace clustering on the other hand yields a set of lower-dimensional subspaces with their corresponding objects projected to them. These arbitrarily-oriented subspaces can be described by a set of principal components like in ORCLUS [14] or through HNF representation of hyperplanes as in CASH [15]. Subspace clustering algorithms are distinct among themselves. While e.g. ORCLUS minimizes the so called energy of a clustering, CASH maximizes compactness, two terms which not necessarily comply with each other. Further ORCLUS optimizes by the number of partitions k and the dimensionality l while CASH optimizes by the minimum number of objects per hyperplane and maximum allowed deviation from it. At this point we

ask: What is the common goal that not only those two, but any subspace clustering algorithm in general can be tailored at to optimize for? The answer is: Expressing data through a set of lower-dimensional subspaces. Just like autoencoders or local PCA aim at encoding the ambient space in such a way that the reconstruction from the latent representation is minimized, we re-phrase the subspace clustering task with the goal in mind to construct an internal evaluation measure:

Definition 1 (Subspace clustering as dimensionality-reduction and manifold learning task)**.** *Given a dataset $\mathcal{D}$ in ambient space, the task of a subspace clustering algorithm is to yield a clustering $\mathcal{C} = \{c_0, c_1, ..., c_k\}$ and their corresponding set of subspaces $\mathcal{S} = \{s_0, s_1, .., s_k\}$ such that the loss $\mathcal{L}(\mathcal{D}, g(f(C, S)))$ is minimized, where $(C, S) := \{(c_i, s_i)\}$, $f(C, S)$ corresponds to an encoding of the dataset and $g(f(C, S))$ corresponds to the decoding of the dataset.*

Definition 2 (Encoding of a subspace cluster)**.** *Given is a subspace cluster c_i as part of a $n \times d$-dimensional data matrix in a subspace s_i with dimensionality l, where d denotes the full dimensionality of the ambient space and k denotes the number of clusters. Further the index $i \in [0, k]$ denotes the corresponding cluster id. First, the data is centered by subtracting $\mathcal{O}_i$, the mean of all points in c_i. Then the PCA, which is based on the covariance matrix of all points in c_i, provides the ordered eigenvalues $\lambda_0, \ldots, \lambda_d$ and respective eigenvectors $\gamma_0, \ldots, \gamma_d$, last of which are known as the principal components of c_i. The subspace cluster is projected onto the first (i.e., those belonging to the largest eigenvalues) l principal components $\gamma_0, \ldots, \gamma_l$ by multiplying it with the $l \times d$-dimensional part of the eigenvector matrix $\Gamma_i = (\gamma_0, \ldots, \gamma_l)^T$: $\mathcal{M}_i = c_i \times \Gamma_i^T$. The complete encoding of a subspace cluster is then this l-dimensional projection of the points $\mathcal{M}_i$ together with Γ_i and the origin in the ambient space $\mathcal{O}$. Thus,*

$$f_c(c_i, s_i) = (\mathcal{M}_i, \Gamma_i, \mathcal{O}_i)$$

Definition 3 (Piecewise Linear Approximation (PLA))**.** *The PLA of a dataset $\mathcal{D}$ using results of a subspace clustering $(C, S) = \{(c_i, s_i)\}$ corresponds to the encoding $f(C, S)$ of the dataset, which is the combination of the encodings of all subspace clusters $f_c(c_i, s_i)$.*

Definition 4 (Decoding of a subspace cluster)**.** *Given the matrices $(\mathcal{M}_i, \Gamma_i, \mathcal{O}_i)$ of the encoding function as described in Definition 2 the decoding $\hat{\mathcal{D}}$ is obtained by transforming the encoded points back into the original data space: $\hat{\mathcal{D}} = \mathcal{M}_i \times \Gamma_i + \mathcal{O}_i$. In other words, the decoding function of a cluster is*

$$g_c(\mathcal{M}_i, \Gamma_i, \mathcal{O}_i) = \mathcal{M}_i \times \Gamma_i + \mathcal{O}_i$$

Based on these new definitions we want to investigate some connections between the piecewise linear approximation based on subspace clusterings and autoencoders. For autoencoders or neural networks in general, the so called universal approximation theorem [16] guarantees that a feed-forward neural network with at least one hidden layer is capable to approximate any function φ with a non-linear manifold F so that $\forall x, \forall \varepsilon > 0 : |F(x) - \varphi(x)| < \varepsilon$, if sufficient hidden units are provided. That is useful for the case that all points of a dataset were created by a common function φ, or in other terms: that all points were generated by a single process which are located on or around a non-linear manifold. But that is not necessarily the case in real world data and that is when subspace clustering is advantageous. We can formulate the corresponding theorem for subspace clustering, which is based on the points in the dataset and not on the function creating them:

Theorem 1 (Adapted Approximation Theorem)**.** *Every dataset $\mathcal{D}$ on a potentially non-linear manifold can be described by a piecewise linear approximation based on a subspace clustering (C, S) so that: $\forall \varepsilon > 0 : \mathcal{L}(\mathcal{D}, g(f(C, S))) < \varepsilon$, if sufficient subspace clusters of sufficient dimensionality are allowed.*

As a sketch the theorem can be proved using the original universal approximation theorem: for every function φ generating a dataset $\mathcal{D}$ there is a manifold F approximating φ as described above. Since a manifold is local Euclidean, there exists a neighbourhood for every data point which is homeomorph to an open subset of $\mathbb{R}^n$. These subsets correspond to the eigenspaces of the subspace clusters as described in Definition 2, and thus φ resp. $\mathcal{D}$ can also be approximated with the PLA as described in Definition 3. A *trivial solution* to approximate $\mathcal{D}$ could put every point in a single, full-dimensional cluster and thus maximize the number of clusters $|(C, S)|$ as well as their dimensionality $dim(c_i)$. The number of clusters $|(C, S)|$ and their dimensionality $dim(c_i)$ plays a fundamental role in Theorem 1, as a trivial solution could maximize both variables and put every point in a single, full-dimensional cluster. The arbitrariness of those two variables corresponds to the arbitrary number of hidden units of the neural network in the universal approximation theorem. This leads to a diversification of this one abstract variable "number of hidden units" to the two variables "number of groups in the dataset" and "number of relevant attributes for each group". Nevertheless, both variables should be taken into account when using it as optimization criterion or as evaluation measure, see Section III-A.

A. Manifold learning as internal measure

Just as in [12] it is stated that autoencoders need a balancing between minimizing the reconstruction error and satisfying regularizations such as the dimensionality of the latent layer in order to learn a meaningful lower-dimensional representation of the data. We introduce here constraints which are indispensable for the subspace clustering task.

Definition 5 (Reconstruction Loss)**.** *The reconstruction loss of a single cluster is the sum of all squared distances between the original points x_j and the reconstructed points $\hat{x}_j$ of the cluster. Normalizing by the number of elements in the cluster leads to invariance regarding the number of points in the*

cluster (and thus in the dataset)

$$\mathcal{L}_c(c_i, g(f_c(c_i, s_i))) = \frac{1}{|c_i|} \sum_{j=1}^{|c_i|} dist(x_j, \hat{x}_j)^2$$

As distance function we use the Euclidean distance divided by the square root of the number of dimensions to stay scale invariant (see Section III-B) w.r.t. the number of dimensions: $dist(x, \hat{x})^2 = \frac{1}{d} \sum_{i=1}^{d} (x^i - \hat{x}^i)^2$*, where* x^i *denotes the value of* x *in the* i*-th attribute. The reconstruction loss of the dataset is then*

$$\mathcal{L}(\mathcal{D}, g(f(C, S))) = \sum_{i=1}^{|(C,S)|} \mathcal{L}_c(c_i, g(f_c(c_i, s_i)))$$

As we want to prevent that the subspace clustering simply learns the identity function as described after Theorem 1 we introduce the possibility to penalize high dimensionalities of subspace clusters as well as a high number of clusters. For the former, the hyperparameter α is multiplied with the subspace dimensionality[2] l and the product is added to the cluster loss $\mathcal{L}$. Here, $\alpha = 0$ means that we do not put any sparsity constraint on the dimensionality of the latent layer, while the larger α the higher we penalize the dimensionality. For the latter, the hyperparameter β is multiplied with the number of clusters $|(C, S)|$ and also added to $\mathcal{L}$, leading to the total reconstruction loss:

$$\mathcal{L}_{total}(\mathcal{D}, g(f(C, S))) = \mathcal{L}(\mathcal{D}, g(f(C, S))) + \alpha \cdot l + \beta \cdot |(C, S)|$$

Figure 1 points out the connection between our new definition of subspace clustering and classic autoencoders. Equivalent to the encoding step of an autoencoder we perform the subspace clustering and with that a PLA of the original data. In the latent layer each cluster is projected onto its own low dimensional eigenspace according to the principal components and the dimensionality as given by the clustering. The decoding corresponds to the re-transformation to the original space. At this point there are certain questions which may arise, regarding the hyperparameters α and β: (1) Couldn't both hyperparameters be regarded as an "external" penalty? Here the answer is: it depends. If both are set equal ($\alpha = \beta$) then both influences (number of clusters and dimensionality of subspaces) are treated equally. However, the external character of α and β do not make this approach an external evaluation measure, since it still relies on the internal assumption that non-linear manifolds are piecewise-linearly approximated in such a way that the deviation from the linear approximations is minimized. (2) Does a hyperparameter not introduce uncertainty in the evaluation? It depends in one aspect on the sensitivity of the hyperparameters. The authors in [11] also introduce a hyperparameter ϱ in their internal measure and claim that it is insensitive. In this work we conducted experiments (Experiment 4) with the purpose to investigate the sensitivity. Further, this "uncertainty" can, depending on the use-case of the scientists, be a degree of control. In autoencoders the users can set the dimensionality of the latent layer (aka "the bottleneck). In our architecture, the scientists can also, if they have a certain background knowledge explicitly set and therefore control the weights of latent space dimensionality and cluster cardinality, emphasizing besides the reconstruction loss either the importance of the number of clusters or the importance of a low-dimensional subspace.

[2]If there are different dimensionalities the median is used in the experiments.

B. Properties of the internal evaluation measure

In the following we investigate the SRE w.r.t. some desirable properties for internal evaluation measurements.

a) Scale invariance regarding number of dimensions: If the quality of the reconstruction per dimension is fixed, i.e., if the mean and the variance of the dimension-wise distance between original point and reconstructed point is fixed, then the number of dimensions of the dataset does not influence the SRE $\mathcal{L}$. The main idea of the proof, which is here omitted for brevity, is that the distance between original and reconstructed points we defined in Definition 5 is independent of their dimensionality in contrast to, e.g., the Euclidean distance. In practice, the premises are fulfilled if the ratio between noise dimenions and important dimensions per cluster stays the same.

b) Scale invariance regarding number of points : SRE is independent of the number of points in a dataset for a fixed number of clusters, since the cluster loss $\mathcal{L}_c$ is not the accumulated, but the average loss of all points in that cluster.

c) Scale invariance regarding number of clusters: As the loss can be minimized by increasing the number of clusters for describing the dataset, $\mathcal{L}$ is dependent on the number of clusters. Since it depends on the structure of the dataset to what extent additional clusters can reduce the reconstruction loss, there is no universal normalization we could improve the SRE with regarding this aspect, which is why β is chosen according to the dataset. Additionally, the optimal number of clusters depends on the usecase, so here β allows the user to tailor the quality function to the respective goal of the clustering.

d) Comparability: Basically, there are three types of experiments when scientists need an internal evaluation measurement. They want to compare (1) different algorithms on the same dataset, (2) different parameter settings of one algorithm on a particular dataset, or (3) the applicability of an algorithm on different datasets. SRE is most suitable for type 1 and through α and β it is also useful for type 2. Type 3, the comparison of the performance of an algorithm on different datasets, can be difficult since the reconstructability of a dataset depends on its structure. However, we should compare these results on different datasets with other algorithms' results anyway.

IV. Experiments

In this section we describe and provide the results of several experiments, conducted on generated data as well as on popular real world datasets Iris, Wine, Breast Cancer, and Digits

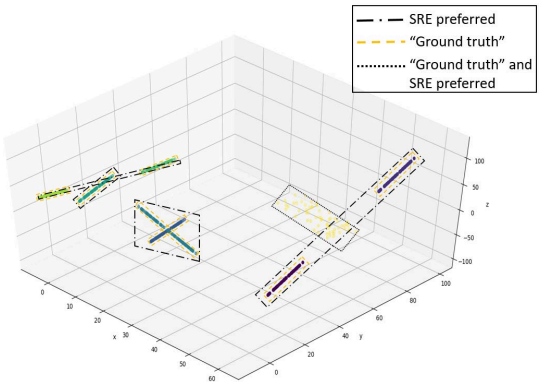


Fig. 2. Synthetic dataset for comparing SRE against a set of external evaluation measures. Best viewed in color.

as provided by the sklearn[3] framework. The dimensionality, number of objects per dataset and number of classes can be seen on the sklearn dataset webpage[3].

A. Experiment 1: SRE vs. External Evaluation Measures

We introduce this first experiment by asking: What makes an evaluation measure actually convincing? The reader may ask also in this context: why should one bother with SRE, if we already have a ground truth with our labeled data? Why not just relying on external evaluation measures such as NMI, ARI etc. like it is done in the majority of previous research so far? In order to approach these questions we have constructed an artificial dataset (3D), with the purpose to make the clustering results as comprehensive as possible. The dataset consists of eight clusters according to our defined "ground truth" indicated by dashed lines and consists of five clusters according to our evaluation model indicated by alternating dashed and dotted lines as shown in Figure 2. Each of the ground truth clusters contain between 60 and 80 objects.

We compare SRE ($\alpha = \beta = 0.5$) against external evaluation measures such as the normalized mutual information (NMI), the adjusted rand index (ARI), the homogenity score (HS), the completeness score (CS), the Fowlkes Mallows score (FMS) and the micro F1 score (F1S). For all mentioned external evaluation measures holds that a score of 1 corresponds to the best possible result, while a score of 0 corresponds to a poor clustering. For the SRE it holds that the lower the score, the better. The experiments were conducted on the artificial dataset, running ORCLUS [14] and CASH [15]. Both algorithms were executed on a hyperparameter grid. We have chosen the best clustering result based on the criterion of obtaining the highest average of the external quality measures: $\frac{NMI+ARI+HS+CS+FMS+F1S}{5}$. The results can be seen in Figure 3 and 4.

In Figure 3 the top part shows bar charts with the external measure results and the SRE. In the bottom we have scatter plots where one can see the clustering results. The left bar

[3]https://scikit-learn.org/stable/datasets/index.html

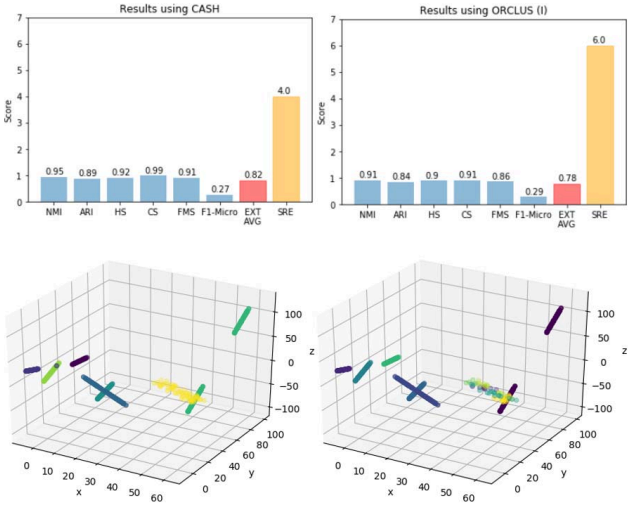


Fig. 3. Synthetic dataset for comparing SRE against a set of external evaluation measures. Best viewed in color.

chart and scatter plot shows the results for CASH, the right scatter plot shows the results for ORCLUS. For CASH we can observe that the average of external quality measures yielded 0.82. Why didn't we obtain a value of 1.0 or at least close to 1.0? One reason for that observation is the poor value of the F1 score. Besides that, there is one major aspect which prevents the other external measures to achieve a score of 1: the two rightmost linear segments are detected as one line despite the fact that according to the ground truth they should be two separate clusters. According to the model behind SRE, assigning both distant lines to the same cluster is favorable, since it results in fewer number of clusters while having no impact on the loss, since both clusters are perfectly located on the same line. At this point one drawback of using external measures becomes visible: neglecting properties of subspace clustering. The results of ORCLUS (Figure 3 right) yield an average external score of 0.78 and therefore just by 0.04 worse compared to the CASH result. Comparing both clustering results (CASH and ORCLUS) in the bottom plots it can be observed that ORCLUS fragmented the planar cluster into four smaller planar clusters. This fragmentation does not seem to affect the external scores significantly, which is however from the perspective of the underlying model of SRE a massive difference. The hyperparameter settings of ORCLUS which led to this clustering are $k = 10, l = 2$. The number of clusters has a high impact on the SRE score, as well as the fact that all clusters are of dimensionality $l = 2$. By purely observing the average scores (0.82 for CASH and 0.78 for ORCLUS) one may think that both clustering results are almost equally good. By looking at the SRE scores (4.0 for CASH and 6.0 for ORCLUS) one can see that the result of CASH is by 33% better compared to the ORCLUS clustering. Through this case we get a first impression that relying on external evaluation measures only, may lead us to draw either wrong conclusions, or at least conclusions which neglect the

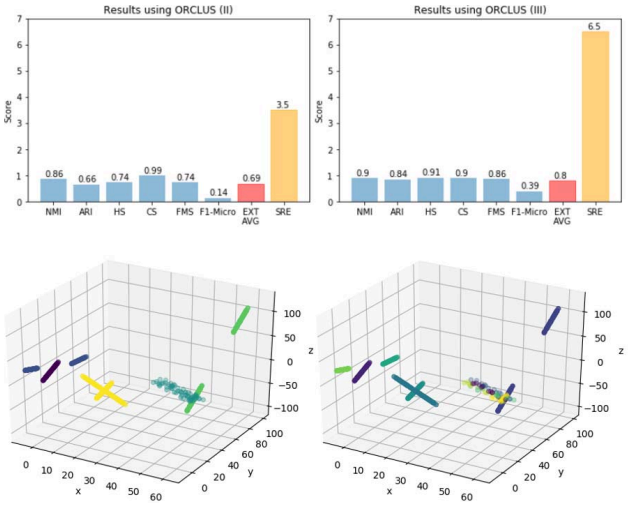


Fig. 4. Synthetic dataset for comparing SRE against a set of external evaluation measures on ORCLUS clustering results. Best viewed in color.

principles of arbitrarily oriented subspace clustering.

So far we have looked at the external evaluation scores and the corresponding SRE score of a clustering. We investigate now the question: Does SRE correspond to the external evaluation scores? For this purpose we run ORCLUS again on our artificial dataset with the same parameter settings as before. After 20 runs with the hyperparameters $k = 10$, $l = 2$ we obtain a clustering as seen in Figure 4 (left). Here the average external score is 0.69 and thus by 0.09 worse compared to the previous ORCLUS run (ORCLUS (I) Figure 4) and by 0.13 worse compared to CASH. Solely based on the external scores one would state that this clustering is of clearly worse quality than the others. Albeit the external scores are worse, the SRE is with 3.5 much lower, i.e., better, compared to the first ORCLUS run with an SRE of 6.0 (42%) and even by 12.5% better compared to the CASH result. How can we actually explain these massive discrepancies? Regarding the SRE, the discrepancies can be understood by taking a closer look at the resulting clustering (Figure 4 left, bottom). ORCLUS has detected five clusters of dimensionality $l = 2$. Compared to the first ORCLUS run, which detected 10 clusters by fragmenting the bottom planar cluster, we have a reduction in number of clusters by 50%. Further the two intersecting lines have been detected as a single planar cluster. For the underlying model of SRE, this clustering (1 cluster, dimensionality of 2) is equally scored as detecting both separately (2 clusters, dimensionality of 1), since both lines are located on a plane, the reconstruction loss in both cases is 0. External evaluation measures, however, are incapable to handle such cases of equivalence, leading to a drop of their respective scores. This issue comes additionally to the previously elaborated fact that both distant linear clusters on the right are detected as one, despite the "ground truth" expects them both to be regarded as separate clusters. Finally in our last run (ORCLUS (III)) with $k = 20$, $l = 2$ we obtain a clustering with an average external score of 0.8, thus being almost en par with the CASH result (0.82). Here again, by solely looking at the external average scores one may be tempted to state that the ORCLUS (III) clustering would be almost as good as the CASH clustering. A close look at the clustering itself (Figure 4, right, bottom) reveals that while the average external score is high, it again has fragmented the planar cluster in many smaller ones. While the external measures fail to capture this fragmentation the SRE score of 6.5 reveals that the result is by about 39% worse compared to the result of CASH (4.0). Referring to the initial question of this experiment of why one should bother using SRE when we could as well just use external measures, one answer to that is: Even if we have labels, and even if we want to trust a "ground truth", external quality measures fail to capture fundamental concepts of an arbitrarily oriented subspace clustering: taking into account the dimensionality and the number of clusters besides the loss.

B. Experiment 2: Pure Reconstruction Quality

How do the algorithms behave on different datasets if we do not apply any regularization at all, which means $\alpha = \beta = 0$? Do we observe common patterns among different datasets regarding loss, number of dimensions or number of clusters? And do the algorithms exploit automatically a higher number of dimensions and higher number of clusters if no regularization is given? The experiment was conducted using the following algorithms: ORCLUS [14], 4C [17], COPAC [18] and LMCLUS [19]. We used the implementations from the data mining framework ELKI [20] for all algorithms. The best hyperparameters for every algorithm was found by iterating over their respective hyperparameter grid using the same step size[4] for each algorithm per experiment. The settings yielding the lowest measured loss per algorithm can be seen in the sourcecode under the following link[5].

For each of the algorithms a triple containing the lowest loss, the number of subspaces and the dimensionality of the subspaces is obtained as shown in Table I. In that table we can observe what we have expected and elaborated on in Section 2: Without any regularization, algorithms like ORCLUS, COPAC and 4C exploit a high number of dimensions as well as a high number of clusters. For the Iris dataset COPAC dominates with the lowest loss and at the same time the lowest dimensionality, but puts almost every point in a single cluster. LMCLUS exploits the dimensionality but interestingly not the number of clusters which yields one single cluster with a reconstruction loss of 0.005. Throughout all datasets LMCLUS dominates the number of clusters by requiring only one cluster, which is interesting since there is no actual parameter of the algorithm directly influencing the number of clusters except $minpoints$. On the Wine dataset ORCLUS dominates by having the smallest loss *and* the smallest number of clusters, while COPAC has also a loss of zero, but has the lowest dimensionality, at the cost of having 176 clusters. In the Breast

[4] 1 for minpts, dim, k, 0.1 for ε-range, Eigenvalue threshold δ, sensitivity threshold

[5] Sourcecode: https://www.dropbox.com/s/1nhln46cxaidael/ORTA.zip?dl=0

TABLE I

RECONSTRUCTION LOSS, DIMENSIONALITY AND NUMBER OF CLUSTERS FOR THE DETECTED SUBSPACES BY THE ALGORITHMS FOR DIFFERENT REAL-WORLD DATASETS ($\alpha = \beta = 0$). BOLD ENTRIES REPRESENT RESULTS BEING DOMINATED BY AT LEAST ONE CRITERIA (LOSS, DIM, CLUS).

	Iris			Wine			Breast			Digits		
Algorithms	Loss	Dim	Clus	Loss	Dim	Clus	Loss	Dim	Clus	Loss	Dim	Clus
ORCLUS	0.000	3	15	**0.000**	**12**	**2**	**0.000**	**25**	**5**	**0.000**	**60**	**5**
COPAC	**0.000**	**2**	**149**	**0.000**	**1**	**176**	**26.747**	**2**	**1**	**7.617**	**5**	**5**
4C	**0.000**	**3**	**144**	0.000	12	178	305.744	29	5	0.000	63	1613
LMCLUS	**0.005**	**3**	**1**	**0.019**	**8**	**1**	**0.010**	**8**	**1**	**6.809**	**11**	**1**

cancer setting, ORCLUS yields the lowest loss, but at the same time with a higher dimensionality, while COPAC has a comparably high loss but at the same time a very low dimensionality and number of clusters. Finally in the Digits dataset ORCLUS comes with a loss of zero, but achieves this by exploiting almost the full dimensionality of the dataset (64) where COPAC has a high loss, but comes again with lower number of dimensions and same number of clusters.

C. Experiment 3: Influence of Dimensionality

In this experiment we ask: How does a fixed dimensionality of the subspaces influence the number of clusters and the pure reconstruction loss $\mathcal{L}$ for the different algorithms? For that we compare $\mathcal{L}$ as well as the number of clusters of ORCLUS, 4C, LMCLUS, and a simple autoencoder AE (with a single layer, ReLU for encoding, linear activation function for decoding, 1000 epochs, and batch size 4) from Keras[6] on the wine dataset for the fixed dimensionaloities $l \in \{1, 2, 3, ..., 12\}$. The runs of the autoencoder have been repeated ten times per dimensionality, selecting the result with the lowest loss. The algorithms CASH and COPAC have been omitted in this experiment, since no maximum dimensionality of subspaces can be enforced. The results in Figure 5 (left) show, as expected, a decreasing loss with increasing dimensionality for almost all algorithms. Only 4C achieves a loss of 0 for all dimensionalities, exploiting the fact, that we have imposed no limitations on the number of clusters and thus creating an own cluster for each object. While LMCLUS provides a lower loss even at dimensionality 1, it is surpassed by ORCLUS at a dimensionality of 8 which is not visible in Figure 5 (left). The exploitation as seen by 4C demonstrates the necessity of α to regulate the number of clusters. The less effective performance of the autoencoder has to be taken with some grains of salt, since it consists of a single-layer without any further optimizations which may not reflect the full potential of this technique, whose further investigation is beyond the scope of this work.

While we have obtained the loss from each of the methods considering different number of dimensions, we ask: how many clusters did the methods yield for achieving their minimum loss? In Figure 5 (right) we can see that 4C fully exploited the fact that no limitations were imposed on the number of clusters. ORCLUS in contrast was capable to achieve low reconstruction losses mostly with two clusters. However there

[6]blog.keras.io/building-autoencoders-in-keras.html

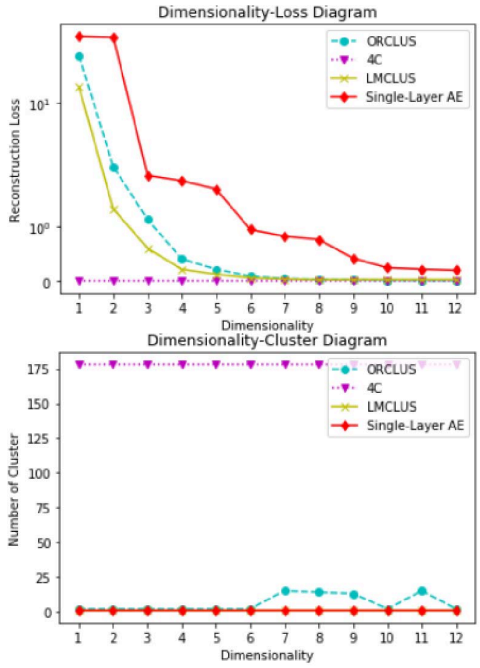


Fig. 5. Top: Loss depending on the dimensionality l on the Wine dataset. Vertical axis is in symmetric log-scale. Bottom: Number of clusters depending on the dimensionality l on the Wine dataset.

are number of dimensions in which ORCLUS exploited the number of clusters as well, achieving low reconstruction errors by yielding 12 to 15 clusters. The autoencoder has been set by default to 1 regarding the number of clusters, since it learns a single manifold. To our surprise LMCLUS yielded throughout all numbers of dimensions one single cluster. Changing the hyperparameter settings of the LMCLUS algorithms did not result in an increased number. Further investigations may be required to elucidate the reasons for this behavior, which is beyond the scope of this work.

D. Experiment 4: Regularization Sensitivity

So far we have observed the effects of having no regularization at all on the evaluation as well as the effects on the reconstruction loss by imposing limitations on the dimensionality. In this experiment we ask: How sensitive are the results with respect to the regularization terms, specifically with regards to their hyperparameters α and β? Asking what are the aspects which are important for an evaluation measure,

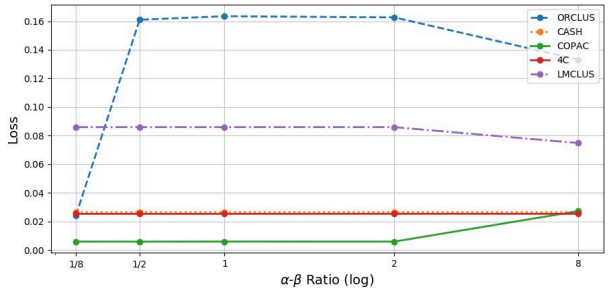


Fig. 6. Sensitivity of the loss w.r.t the hyperparameters α and β of the regularization terms on the iris dataset.

we demand that it should be insensitive towards α and β. Connected to this fact, we ask: Which impacts can we observe if we weight both of them equally? What can be observed if we set α to a high value and β to a low one, and vice-versa? For this purpose in this experiment all five subspace clustering algorithms were tested on the iris dataset with different (α, β)- settings, namely $\{(\frac{1}{9}, \frac{8}{9}), (\frac{1}{3}, \frac{2}{3}), (\frac{1}{2}, \frac{1}{2}), (\frac{2}{3}, \frac{1}{3}), (\frac{8}{9}, \frac{1}{9})\}$. The settings of the ratios of α and β have been chosen in such a way that the sum satisfies $\alpha + \beta = 1$. In Figure 6 we have on the horizontal axis the ratio α/β, and on the vertical axis the computed loss. The results reveal that for different ratios of α and β the loss remains mostly the same. Which confirms that SRE is robust with respect to different hyperparameter settings. An exception to the rule poses ORCLUS which seems sensitive on the extreme cases $\frac{1}{8}$ and 8. This is due to the fact that at $\frac{1}{8}$ more dimensions are permitted, due to a lower α and thus a lower penalty for the number of dimensions. In the following ratios ORCLUS detected optimum settings which have only one dimension but two clusters. Another aspect which this experiment reveals is that overall COPAC yields a lower reconstruction loss, followed by 4C and CASH. LMCLUS performs as the second worst algorithm and ORCLUS as the worst.

V. Conclusion

In conclusion we developed a new internal evaluation measurement for subspace clustering. We translated approaches from autoencoders such as the approximation of non-linear manifolds by a function to the concept of subspace clusters, leading to piecewise linear approximations based on PCAs of single subspace clusters. Our evaluation measure is the first one for arbitrary oriented subspace clustering and fulfills important properties like invariance regarding number of clusters as well as number of dimensions. The experiments show that SRE takes aspects into account which are neglected by external evaluation measures. It also differs from existing internal evaluation measures, since it considers aspects as reconstruction error and model complexity in terms of subspace dimensionality and number of clusters. In future work we want to investigate the two regularization parameters α and β in more detail. Further we want to enhance the autoencoding view to a data compression view, by also utilizing the Minimum Description Length (MDL) for evaluating the reconstruction loss and model complexity. We envision that this work may pave the path for a different understanding and perspective on the arbitrarily oriented subspace clustering problem as well give rise to novel internal evaluation measures.

Acknowledgements

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 responsibility for its content.

References

[1] H.-P. Kriegel, P. Kröger, and A. Zimek, "Subspace clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 4, pp. 351–364, 2012.

[2] E. Müller, S. Günnemann, I. Assent, and T. Seidl, "Evaluating clustering in subspace projections of high dimensional data," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1270–1281, 2009.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," Univ San Diego. Inst for Cognitive Science, Tech. Rep., 1985.

[4] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[5] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[6] D. Moulavi, P. A. Jaskowiak, R. J. Campello, A. Zimek, and J. Sander, "Density-based clustering validation," in *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 2014, pp. 839–847.

[7] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[8] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 224–227, 1979.

[9] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics-theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.

[10] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," 1973.

[11] F. Höppner and M. Jahnke, "Holistic assessment of structure discovery capabilities of clustering algorithms," *ECML-PKDD*, 2019.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] N. Kambhatla and T. K. Leen, "Dimension reduction by local principal component analysis," *Neural computation*, vol. 9, no. 7, pp. 1493–1516, 1997.

[14] C. C. Aggarwal and P. S. Yu, *Finding generalized projected clusters in high dimensional spaces*. ACM, 2000, vol. 29.

[15] E. Achtert, C. Böhm, J. David, P. Kröger, and A. Zimek, "Global correlation clustering based on the hough transform," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 1, no. 3, pp. 111–127, 2008.

[16] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[17] C. Böhm, K. Kailing, P. Kröger, and A. Zimek, "Computing clusters of correlation connected objects," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 2004, pp. 455–466.

[18] E. Achtert, C. Böhm, H.-P. Kriegel, P. Kröger, and A. Zimek, "Robust, complete, and efficient correlation clustering," in *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 2007, pp. 413–418.

[19] R. Haralick and R. Harpaz, "Linear manifold clustering," in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2005, pp. 132–141.

[20] E. Schubert and A. Zimek, "ELKI: A large open-source library for data analysis - ELKI release 0.7.5 "heidelberg"," *CoRR*, vol. abs/1902.03616, 2019.

3.2 "LUCK- Linear Correlation Clustering Using Cluster Algorithms and a KNN Based Distance Function"

Publication: Anna Beer, Daniyal Kazempour, Lisa Stephan, and Thomas Seidl. "LUCK- Linear Correlation Clustering Using Cluster Algorithms and a KNN Based Distance Function". In: *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*. SSDBM '19. New York, NY, USA: Association for Computing Machinery, 2019, 181—184. DOI: 10.1145/3335783.3335801

Statement of Originality: The concept was developed by Anna Beer and implemented by Lisa Stephan in context of her Bachelor's Thesis. Daniyal Kazempour wrote the related work section and provided expert knowledge throughout the process. The concept was discussed and improved with Thomas Seidl.

LUCK- Linear Correlation Clustering Using Cluster Algorithms and a kNN based Distance Function

Anna Beer
LMU Munich
Munich, Germany
beer@dbs.ifi.lmu.de

Daniyal Kazempour
LMU Munich
Munich, Germany
kazempour@dbs.ifi.lmu.de

Lisa Stephan
LMU Munich
Munich, Germany
lisa.stephan@campus.lmu.de

Thomas Seidl
LMU Munich
Munich, Germany
seidl@dbs.ifi.lmu.de

ABSTRACT

LUCK allows to use any distance-based clustering algorithm to find linear correlated data. For that a novel distance function is introduced, which takes the distribution of the kNN of points into account and corresponds to the probability of two points being part of the same linear correlation. In this work in progress we tested the distance measure with DBSCAN and k-Means comparing it to the well-known linear correlation clustering algorithms ORCLUS, 4C, COPAC, LMCLUS, and CASH, receiving good results for difficult synthetic data sets containing crossing or non-continuous correlations.

CCS CONCEPTS

• **Information systems → Clustering**;

KEYWORDS

Linear Correlation Clustering, Clustering, kNN

ACM Reference Format:
Anna Beer, Daniyal Kazempour, Lisa Stephan, and Thomas Seidl. 2019. LUCK- Linear Correlation Clustering Using Cluster Algorithms and a kNN based Distance Function. In *31st International Conference on Scientific and Statistical Database Management (SSDBM '19), July 23–25, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3335783.3335801

1 INTRODUCTION

Many algorithms trying to find linear correlations have trouble handling crossing correlations, several correlations in one dataset, or non-continuous correlations, such as shown in Fig. 1. But all those constellations are quite common in arbitrary datasets. Our new distance measure LUCK makes them easily detectable and offers a broad foundation for future work. It delivers low values for points which are probably part of the same linear correlation based on the orientation of their k nearest neighbors' distribution. LUCK can be used in any distance-based clustering algorithm, e.g. DBSCAN or k-Means, which we both test and compare to established correlation clustering algorithms like ORCLUS, 4C, COPAC, LMCLUS, and CASH. Further it is adaptable with one easy to set threshold parameter and learns from that the optimal number k of nearest neighbors to regard for the orientation.

We have only tested the most basic and common clustering algorithms with LUCK, so there is much room for improvement by trying other clustering algorithms. Nevertheless, the results are already good in comparison and we are able to detect crossing linear correlations as well as non-continuous ones. Due to lack of space we here only show the two base cases from Fig. 1, modifying their number of dimensions, percentage of noise, and jitter. We give an overview over related work in Sec. 2 and explain LUCK in detail in Sec. 3. Above mentioned experiments are performed in Sec. 4. Sec. 5 concludes this paper and gives an overview over a multitude of possible future work.

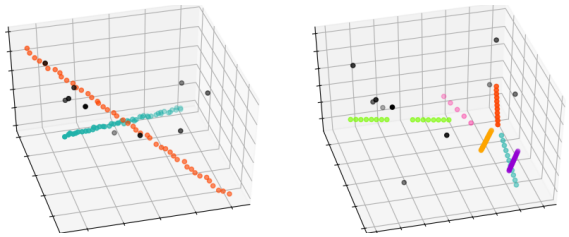


Figure 1: Base case experiments

2 RELATED WORK

Under the term correlation clustering we understand clusters which are located in interesting subspaces which are not axis-parallel but arbitrarily oriented as stated in [9]. There exists a wealth of literature in this area: Historically ORCLUS [5] was the first of its kind, followed by works like 4C [6], COPAC [2], HiCO [3], ERiC [4], LMCLUS [8] and CASH [1]. In the following we give an overview over those to which we will compare our method.

Comparative Methods. The main idea behind ORCLUS is a k-means [10] like approach. It begins with k initial seeds. Then it assigns points to clusters according to a distance function based on the eigensystem of the current cluster obtained from a PCA. ORCLUS relies on a *cluster-based locality assumption*, which means that the subspace of each cluster is learned from its cluster members in a local neighborhood. 4C in contrast combines the concept of PCA with density-based clustering such as DBSCAN [7]. 4C detects arbitrary numbers of clusters but requires a specification of the density-threshold. It is biased towards the maximal dimensionality of correlation clusters which is user specified. In contrast to ORCLUS, 4C is not relying on a cluster-based but on an *instance-based*

SSDBM '19, July 23–25, 2019, Santa Cruz, CA, USA

ACM ISBN 978-1-4503-6216-0/19/07…$15.00
https://doi.org/10.1145/3335783.3335801

locality assumption, i.e., the correlation distance measure which is specifying the subspace is actually learned from the local neighborhood of *each point* in data space. COPAC assigns a local correlation dimensionality to each object, which corresponds to the dimensionality of the respective best fitting correlation cluster. Partitioning the dataset by those dimensionalities and using the eigensystem of each partition, linearly correlated clusters of *different* dimensionality are obtained. Like ORCLUS and 4C, COPAC also relies on the locality assumption. LMCLUS differs from the previous and computs histograms of the distances of the points to each intermediate arbitrary oriented representation. The sampling which belongs to a histogram providing the best separability between a near-zero mode vs. the rest is selected and the data points are partitioned on the best separation. Like all previously mentioned methods, also LMCLUS relies on computing eigensystems and is reliant on the locality assumption. In contrast, CASH, a top-down dynamic-grid based approach, does depend on neither the locality assumption nor eigensystems. Relying on Hough transform, it is a global correlation clustering algorithm, i.e., data points within the detected linear correlated clusters are not necessarily locally dense but can be of arbitrary distance among the linear correlation.

Applied Clustering Algorithms and Delimitation. In this first approach using LUCK, we apply the probably most common clustering algorithms, DBSCAN and k-Means. DBSCAN [7] is a density-based clustering algorithm which needs two parameters ε and $minPts$, so that a point can be considered as dense, if it has at least $minPts$ many points in its ε- range. k-Means [10] is a partitioning algorithm which needs the number of clusters k. Even though most diverse distance based clustering algorithms could be used, some of which may be more suitable, we wanted to investigate the behaviour for those two basic clustering methods first. LUCK is neither grid- nor eigensystem based, and requires only one easy to set hyperparameter.

3 LUCK

With our innovative approach LUCK (**L**inear Correlation Clustering **U**sing **C**luster Algorithms and a **k**NN- based Distance Function) we investigate the possibility of using clustering algorithms to find correlation clusters. For that we introduce a new distance measure d_{corr} which we explain in Sec. 3.1.2. It gives low values for points which are probably linearly correlated, taking the k nearest neighbors (kNN) of every point into account by looking at the orientation of their distribution (which we further refer to as the point's orientation) If two points' orientations are the same, the linear correlations in which those two points lie are parallel. If the connecting line between both points has also the same direction, both points lie in the same linear correlation.We apply the well known clustering algorithms k-Means and DBSCAN using d_{corr} instead of the Euclidean distance. With that, points which have similar orientations lying in a linear correlation are clustered together. We elaborate the details in Sec. 3.3.

3.1 Distance Measure

For the distance measure we aim for low values for points which are correlated, and high values for points which are not correlated. We assume that the kNN of a point are meaningful for a point, i.e., that if a point lies in a correlation cluster, its nearest neighbors probably lie in the same correlation cluster. Thus, the orientation of the distribution of the kNN of a point deliver the orientation of a correlation cluster it probably lies in. We represent the orientation of the kNN as explained in the following Sec. 3.1.1. Based on the orientation, the distance d_{corr} between two points is calculated as explained in Sec. 3.1.2.

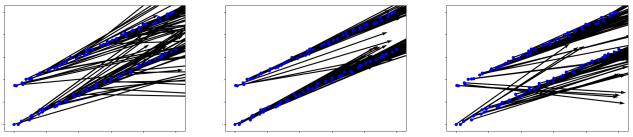


Figure 2: Orientation vectors for $k = 5$, $k = 15$, and $k = 25$

3.1.1 Orientation Vector. We calculate the orientation vector $\overrightarrow{o_p}$ of a point p with k nearest neighbors $q_1, ..., q_k$ w.r.t. the vectors $\overrightarrow{pq_1}, ..., \overrightarrow{pq_k}$ between p and its kNN as follows:

(1) We standardize all vectors $\overrightarrow{pq_i}$, so that if two kNN lie on opposite sides of p (but all three are on the same line) they do not neutralize each other. For that we invert all $\overrightarrow{pq_i}$ for which the first dimension is negative, i.e. we multiply it with -1. If the first dimension is zero, we invert $\overrightarrow{pq_i}$ if the second dimensions is negative and so on[1].
(2) We norm all vectors $\overrightarrow{pq_i}$ by dividing by their length $|\overrightarrow{pq_i}|$
(3) We calculate the mean of all standardized and normalized vectors: $\overrightarrow{o_p}' = \frac{1}{k} \sum_{i=1}^{k} \overrightarrow{pq_i}$ and normalize the resulting vector again to get the orientation $\overrightarrow{o_p}$ of a point.

3.1.2 Definition of Distance Measure. Using the orientation vectors of two points we can calculate their distances as follows.

$$d'_{corr}(p,q) = \left| |\overrightarrow{o_p} \circ \overrightarrow{o_q}| - \frac{|\overrightarrow{o_p} \circ \overrightarrow{pq}| + |\overrightarrow{o_q} \circ \overrightarrow{pq}|}{2} \right|, \quad (1)$$

where $\circ$ denotes the scalar product.

With that we almost reach the goals described before.Nevertheless, for two points being part of two parallel linear correlations the connecting line becomes more and more similar to their orientation vectors the farther away they lie from each other. Thus, we regard also the Euclidean distance between two points, resulting in our final definition for the distance d_{corr}: $d_{corr}(p,q) = d'_{corr}(p,q) \cdot |\overrightarrow{pq}|^2$

3.2 Dynamic choice of k

Fig. 2 shows the influence of the number of neighbors regarded, k, on the orientation vector. Too small values for k can lead to undesired results for correlations containing jitter, where for too high values noise points or nearby correlation clusters influence the orientation vector. As the best k is hard to choose and can vary for different points, we introduce a measure for clarity/scattering of the orientation vector in Sec. 3.2.1 and with the help of that dynamically choose k as described in section 3.2.2.

3.2.1 Scattering. If k is chosen optimally for a point which lies in a correlation, the direction vectors to its k nearest neighbors are similar to each other and scatter only little around its orientation vector $\overrightarrow{o_p}$. As measures for this scattering, like total dispersion [11]

[1]For sake of determinism the dimensions are previously sorted by the maximum occurring value in the dataset

deliver unintuitive results especially for opposed direction vectors, we define the following measure s_p for the scattering of a point p:

$$s_p = \frac{1}{k}\sum_{i=1}^{k}(1 - |\overrightarrow{pq_i} \circ \overrightarrow{o_p}|)^2, \quad (2)$$

where $\overrightarrow{pq_i}$ is the vector between p and one of its kNN q_i. It is based on the cosine distance (thus the scalar product) and regards that two opposed direction vectors still define the same linear correlation (thus the absolute value). The scattering is high if angles between $\overrightarrow{pq_i}$ and $\overrightarrow{o_p}$ are small, thus we subtract it from 1. Referring to the variance of variables we sum over the squares.

To limit possible values of s_p, the user may set a threshold τ, which is the only parameter of LUCK, determining the upper bound of scattering. Using τ allows an automatic determination of k as described in Sec. 3.2.2 and is easy to set: values may range from 0 to 1, where $\tau = 0$ means that all points in the neighborhood of a point lie on a straight line. A higher value of τ allows finding clusters with points that scatter more.

3.2.2 Determination of k. Starting with a minimal number of points $minK$, we increase k in every step and calculate the scattering s_p of every point. The first value of $k \in \{minK, ..., n-1\}$, where s_p falls below a previously defined threshold τ, is the best k for p. If s_p does not fall below τ, p is declared noise. Too high values for $minK$ can lead not only to a higher runtime, but also to less expressive $\overrightarrow{o_p}$ as nearby correlation clusters could influence the orientation of a point. Otherwise, for highly scattered correlations, a too low value for k can lead to a deceptive $\overrightarrow{o_p}$. Thus, $minK$ is calculated depending on the anticipated scattering implied by τ, where n is the size of the dataset: $minK = max(\tau n, 2)$.

3.3 Overview - Complete Algorithm

We summarize our approach in Algorithm 1: Our only input parameter is τ, from which we can calculate $minK$. For every point we calculate its orientation. For that we first calculate the optimal number of neighbors k which will be taken into account. For that we increase k while regarding the scattering s_p, which depends on the point's orientation $o_p(k)$. If the scattering does not fall below τ, the point is a noise point and does not belong to any correlation cluster. Else, its orientation is $o_p(k)$, where k is the first k for which s_p falls below τ. With that the distances between all points can be calculated. This distance matrix can be used instead of the Euclidean distance for any distance-based clustering algorithm.

Complexity. The nested for-loops (line 4 and 6 in Algorithm 1) around the calculation of the scattering s_p dominate the complexity of LUCK before applying a clustering algorithm. To calculate s_p, the orientation $\overrightarrow{o_p}$ is calculated for all k nearest neighbors, resulting in $O(k * k)$. Depending on k_m, the maximal optimal k occurring in the dataset we get an overall runtime of $O(n * k_m^3)$.

Algorithm 1 LUCK

```
1:  τ = userinput
2:  minK = max(τn, 2)
3:  // calculate orientations of all points
4:  for every point p do
5:      k_optimal(p) = 0
6:      for (k= mink; k<n; k++) do
7:          o_p(k) = calculateOrientation(p, k) (see 3.1.1)
8:          s_p = calculateScattering(p, k, o_p(k)) (see Eq. (2))
9:          if s_p < τ then
10:             k_optimal(p) = k
11:             break
12:         else
13:             k++
14:     if k_optimal(p) == 0 then
15:         p is noise
16: //calculate distance matrix
17: initialize(DistMat)
18: for every point p do
19:     for every point q do
20:         DistMat_pq = d_corr(p, q) (see 3.1.2)
21: //Apply distance-based clustering algorithm using DistMat
```

4 EXPERIMENTS

We evaluated our method with respect to several parameters: number of dimensions, noise and jitter. Our two base case datasets are rather difficult for most correlation clustering algorithms, since the first one contains two crossing lines and the second contains six lines one of which is non-continuous, as shown in Fig. 1. We use the Adjusted Rand Index (ARI) to compare our results with those of ORCLUS, 4C, COPAC, LMCLUS, and CASH, where 1 means a perfect result. For a better overview, we give all results in radar charts giving the ARI for all tested methods. Base case 1 is always on the left side, base case two on the right.

4.1 Number of dimensions

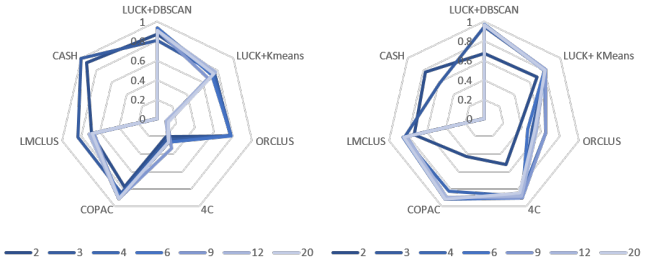


Figure 3: ARI for growing number of dimensions

Results received by LUCK+DBSCAN and LUCK+kMeans did, with increasing number of dimensions, not worsen as much as those of ORCLUS, 4C, LMCLUS (at least for base case 1) or CASH, as can be seen in Fig. 3. Particularly for CASH we were not able to obtain any results for more than 4 dimensions due to a too high memory consumption. We note, that even though 4C performs well for the second dataset regardless of the number of dimensions, it is not able to handle the crossing lines in the first dataset. For the first dataset LUCK+DBSCAN yields the best results for all dimensionalities higher than 4.The results of LUCK+Kmeans are sligthly worse due to noise which can not be detected by k-Means.

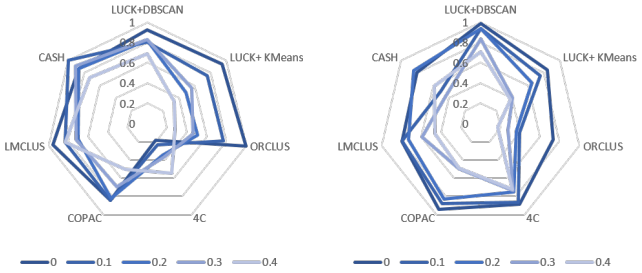


Figure 4: ARI for different percentages of noise

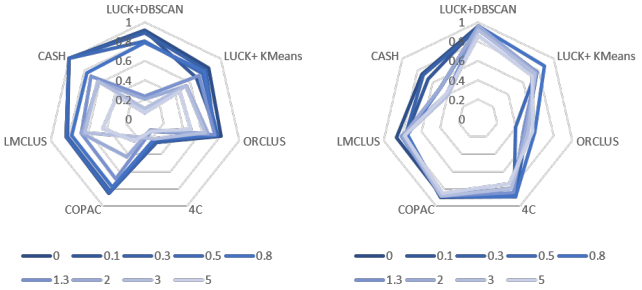


Figure 5: ARI for different levels of jitter

4.2 Noise

We tested our base case experiments with varying percentages of noise ranging from 0 to 0.4. As Fig. 4 shows, DBSCAN can cope significantly better with increasing noise than k-Means using LUCK, outperforming ORCLUS and 4C by far for the first dataset, and LMCLUS as well as CASH for the second dataset. Even for 30% noise, LMCLUS+DBSCAN reaches still an ARI of 0.83 resp. 0.84 for the first resp. the second dataset. That is the second best result after CASH for the first datset, and the best result for the second dataset. We note that ORCLUS and LMCLUS can not cope well with much noise. 4C shows an interesting behaviour for dataset 1 with crossing lines, as it gets better for increasing percentage of noise, but even then does not reach an ARI of higher than 0.55.

4.3 Jitter

We changed the variance of distribution from values between 0 up to 5, focusing on low values. Where jitter clearly worsens the results of LUCK if there are crossing lines, it has almost no influence if the lines are far away from each other. That is naturally determined by its definition- the more jitter and the nearer another cluster is, the less clear is the orientation of each point, thus the distance measure becomes less expressive. LUCK+KMeans delivers similar results as ORCLUS, and 4C is, as before, not able to detect crossing lines at all. All other algorithms cannot cope with levels of jitter higher than 1, see Fig. 5. For the second dataset, ORCLUS and CASH have both problems for any level of jitter, while LUCK+DBSCAN delivers the best results for all levels of jitter each.

5 OVERVIEW AND CONCLUSION

Our experiments show, that LUCK can handle more dimensions well, while much jitter in combination with crossing linear correlations lead to a decrease of quality. However, in datasets without crossing lines, jitter was handled well. LUCK+DBSCAN was more robust to noise than most of the tested comparative methods, especially regarding both datasets at once. Also, using LUCK enables finding correlations on diverse datasets: ORCLUS and CASH generally performed poorly for the dataset with an interrupted correlation, 4C could not handle the crossing lines. LMCLUS was mostly slightly worse than our algorithms, while the quality of COPAC was similar to ours. LUCK+DBSCAN performed better than LUCK+kMeans especially for much noise due to the noise-detection property of DBSCAN. All in all these first drafts of using the most basic clustering algorithms with our novel distance measure already delivered good results compared to established algorithms in the field of correlation clustering.

We developed, to the best of our knowledge, the first method allowing to use any distance based clustering algorithm to find linear correlations. Using DBSCAN and k-Means we did not only detect crossing linear correlations well, but also dealt with several correlations in a dataset, non-continuous correlations and multiple dimensions. Noise and jitter of different degrees were handled as well, where LUCK handles noise better in combination with DBSCAN than with k-Means, as DBSCAN itself is able to detect noise. Using k-Means-- instead of k-Means in future work could improve noise handling. More recent clustering algorithms could be used in combination with LUCK to receive even better results as well as using the principal component instead of the orientation vector. Also extending LUCK to find also spheres and higher-dimensional hyperspheres seems promising.

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] Elke Achtert, Christian Böhm, Jörn David, Peer Kröger, and Arthur Zimek. 2008. Global correlation clustering based on the Hough transform. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 1, 3 (2008), 111–127.

[2] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. 2007. Robust, complete, and efficient correlation clustering. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 413–418.

[3] Elke Achtert, Christian Bohm, Peer Kroger, and Arthur Zimek. 2006. Mining hierarchies of correlation clusters. In *Scientific and Statistical Database Management, 2006. 18th International Conference on*. IEEE, 119–128.

[4] Elke Achtert, B Christian, Hans-Peter Kriegel, Arthur Zimek, et al. 2007. On exploring complex relationships of correlation clusters. In *null*. IEEE, 7.

[5] Charu C Aggarwal and Philip S Yu. 2000. *Finding generalized projected clusters in high dimensional spaces*. Vol. 29. ACM.

[6] Christian Böhm, Karin Kailing, Peer Kröger, and Arthur Zimek. 2004. Computing clusters of correlation connected objects. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 455–466.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96. 226–231.

[8] Robert Haralick and Rave Harpaz. 2007. Linear manifold clustering in high dimensional spaces by stochastic search. *Pattern Recognition* 40, 10 (2007), 2672 – 2684.

[9] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. 2012. Subspace clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2, 4 (2012), 351–364.

[10] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[11] Seppo Mustonen. 1997. A measure for total variability in multivariate normal distribution. *Computational Statistics & Data Analysis* 23, 3 (1997), 321–334.

3.3 "LUCKe — Connecting Clustering and Correlation Clustering"

Publication: Anna Beer, Lisa Stephan, and Thomas Seidl. "LUCKe — Connecting Clustering and Correlation Clustering". In: *2021 International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2021, pp. 431–440. DOI: 10.1109/ICDMW53433.2021.00059

Statement of Originality: The concept of the previous paper [18] was refined and improved by all authors. Anna Beer led the further development process; Lisa Stephan implemented the changes and performed the experiments designed by Anna Beer. Details and basic points were discussed with Thomas Seidl.

Notes: The paper introduces the method LUCKe, a successor of the previously presented method LUCK [18]. LUCKe generalizes and improves LUCK significantly.

LUCKe — Connecting Clustering and Correlation Clustering

Anna Beer, Lisa Stephan, Thomas Seidl
LMU Munich
{beer,seidl}@dbs.ifi.lmu.de

***Abstract*—LUCKe allows any purely distance-based "classic" clustering algorithm to reliably find linear correlation clusters. An elaborated distance matrix based on the points' local PCA extracts all necessary information from high dimensional data to declare points of the same arbitrary dimensional linear correlation cluster as "similar". For that, the points' eigensystems as well as only the relevant information about their position in space, are put together. LUCKe allows transferring known benefits from the large field of basic clustering to correlation clustering. Its applicability is shown in extensive experiments with simple representatives of diverse basic clustering approaches.**

***Index Terms*—linear correlation clustering, clustering, PCA**

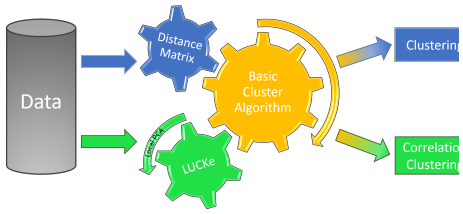


Fig. 1. Concept. Using LUCKe instead of an, e.g. Euclidean distance matrix when applying any basic clustering algorithm yields correlation clusters instead of traditional ones

I. Introduction

Even though linear correlation clustering is an established and long-known data mining task, some problems are still not solved satisfactorily. Linear correlations between attributes of a data set can be found quickly, easily, and mathematically meaningful using, e.g., principal component analysis (PCA) [1]. But PCA does not work anymore if there are several different correlation clusters in the data set. Often, real-world data does not originate from one single distribution function, but several sources, creating groups of related data points. Those can be correlated depending on different attributes, with differently strong and differently dimensional dependencies, and are called linear correlation clusters, or subspace clusters. Nevertheless, most established algorithms in this field require expert knowledge, e.g., the number of clusters we expect in the data or the number of dimensions that could be correlated to form some of those clusters. Also recent subspace clustering algorithms (e.g., [2]) require even the exact number of clusters for each subspace. But one of the reasons the world needs those algorithms is the lack of time experts have for such tasks and the amount of data we generate every day increases steadily. Even with extensive expert knowledge given, extracting correct correlation clusters is difficult: correlation clusters crossing each other or such with different dimensionalities as shown in Fig. 3 are especially hard to analyze.

In the closely related field of clustering, there are plenty of advanced, fast, and noise-detecting algorithms which we can make use of for linear correlation clustering. Some established correlation clustering algorithms already incorporate basic clustering algorithms as we explain in Sec. II. [3] introduced LUCK, a first approach to finding one-dimensional correlation clusters by combining a new type of distance function with any arbitrary distance-based clustering algorithm. The idea to regard those points in a data set as similar or near to each other, which probably lie in the same linear correlation cluster also builds the basis for the approach we introduce in this paper: *LUCKe* (**LUCK e**xtended). In contrast to LUCK, LUCKe can find linear correlations of *arbitrary* dimensionality by using local PCA and a novel similarity function. Our main idea is shown in Fig. 1: with LUCKe, any distance-based basic clustering algorithm can find correlation clusters, building a bridge between the two research areas.

Our main contributions are as follows:

- With LUCKe any purely distance-based clustering algorithm can find linear correlations instead of "classic" clusters
- Varying dimensionalities of correlation clusters are incorporated fully automatically
- In extensive experiments we show that even complex combinations of several clusters can be found easily using LUCKe, e.g., parallel or intersecting clusters.

The rest of this paper is structured as follows. In Sec. II we give an overview of related work. In Sec. III we explain and analyze our new method in detail. In Sec. IV we show with extensive experiments the abilities of our novel method LUCKe, systematically on synthetic data as well as on real-world data. Sec. V concludes the paper and gives an outlook on future work.

II. Related Work

As LUCKe builds a bridge between correlation clustering and basic clustering, we regard both: We explain the representative algorithms for basic clustering that we use in our experiments in Sec. IV. We then regard correlation clustering algorithms which already incorporate basic clustering algorithms and explain correlation clustering algorithms unrelated

2375-9259/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDMW53433.2021.00059

to basic clustering, which nevertheless serve as competitors for our results in Sec. IV.

A. Basic Clustering Algorithms

There is a multitude of basic clustering algorithms, most of which can be categorized into one of the four classes density-based, hierarchical, spectral, and centroid-based clustering. We use one representative for each applicable class in our experiments in Sec. IV, which we describe in the following.

DBSCAN [4] is the first density-based cluster algorithm and defines clusters as connected dense areas. A point is dense resp. a core point, if it has at least $minPts$ points in its ε-neighborhood. Like that, DBSCAN can not only find arbitrarily shaped clusters, but also detects noise and the idea behind it already proved to be useful for correlation clustering, e.g., in 4C [5], COPAC [6] and ERiC [7].

Agglomerative clustering [8] is a bottom-up hierarchical clustering algorithm. Starting with all points assigned to a different cluster each, the two closest clusters are merged into a larger cluster. There are different possibilities to calculate the distance between two clusters. Out of all distances between pairs of elements from two different clusters, *single linkage* uses the minimum distance, *complete linkage* uses the maximum distance, and *average linkage* uses the average distance. Another possibility is the variance-minimizing *Ward's criterion*, a type of weighted squared Euclidean distance between the centers of the merged clusters [8], [9].

Spectral clustering [10] can be applied on any graph to find the minimum normalized k-cut [11], i.e. a balanced partitioning of the graph such that edges within a partition have the highest possible edge weights and edges between different partitions have the lowest possible weights. To get a graph based on data points, usually, the kNN graph is used, which connects every point with its k nearest neighbors. There are several ways to calculate the Laplacian out of the adjacency matrix of the kNN graph [10], ways to decide how many eigenvectors of the Laplacian should be used for the final clustering, and methods to cluster the original data points based on the most important eigenvectors [12].

Centroid-based cluster algorithms like k-Means [13], [14] are not purely distance-based, but also need to compute centroids of point sets, which is, e.g., disadvantageous when working with categorical data. As a distance matrix alone is not sufficient input for these algorithms, they are not directly applicable in combination with LUCKe, but could, nevertheless, be adapted.

B. Combining Basic Clustering and PCA

We compare LUCKe with several correlation clustering algorithms in Sec. IV. ORCLUS, 4C, COPAC, and ERiC are, similar to LUCKe, PCA-based and integrate one of the well-known basic clustering algorithms such as k-Means or DBSCAN into their clustering. LMCLUS also computes the eigensystem but apart from that differs strongly from LUCKe. Additionally, we compare LUCKe to the established linear correlation clustering algorithm CASH. We give an overview over LUCK [3], the idea LUCKe is based on, here, and discuss further delimitations and differences to LUCK in Sec. III-E4.

1) Density-based methods: The complete series of density-based correlation clustering algorithms introduced here was developed by mostly the same group of authors, where each algorithm developed the basic idea of combining DBSCAN and PCA further. One of the main differences besides that LUCKe is combinable with several basic clustering algorithms, is the incorporation of PCA resp. the Mahalanobis distance: those algorithms produce inherently non-symmetric distance matrices, ignoring one point's orientation when calculating the distance between two points.

4C [5] combines PCA and DBSCAN, but in contrast to LUCKe+ DBSCAN, it is based on the assumption that points of a correlation cluster are not only highly correlated but also lie close together. Even though dense correlation clusters can give valuable insights, the correlation of points does not depend on their density. 4C even uses ε- neighborhoods, which can cause misleading principal components especially for points in sparse areas. In addition to the two parameters necessary for DBSCAN, 4C has two more input values: Users need to specify an upper bound for the dimensionality of correlation clusters λ, and a threshold δ for the eigenvalues, which is needed to select strong eigenvectors.

COPAC [6] combines PCA with a density-based clustering algorithm, similarly to 4C [5], but faster. Some further drawbacks of 4C, such as the limitation of the correlation dimensionality of detected correlation clusters, are overcome by COPAC. It divides the data set into partitions according to their local correlation dimensionality by using the local PCA.The *local correlation dimensionality* λ_P is given as the minimum number of eigenvalues explaining a certain proportion of the variance. It then applies GDBSCAN [15], a generalized form of DBSCAN, to each of the partitions. Like that, COPAC can simultaneously search for correlation clusters of different dimensionality. Furthermore, distant points that lie on a common hyperplane can be assigned to the same cluster. It has three hyperparameters: the minimum number of points in a cluster $minPts$, a neighborhood range ε, and k, the number of points used to compute the neighborhood of a point, which is used to determine the local correlation dimensionality. Partitioning the data according to the local dimensionality implicates that points from different partitions can not be assigned to the same cluster.

SSCC [16] integrates COPAC into a subspace clustering approach, allowing multilabeling.

HiCO [17] combines a distance measure that respects local correlation dimensionalities with the basic clustering algorithm OPTICS [18], which is based on DBSCAN. It generates relatively simple hierarchies of correlation clusters, where, e.g., a cluster embedded in two higher-dimensional clusters cannot be represented.

ERiC [7] also finds hierarchies of embedded correlation clusters based on the idea of COPAC, but needs two additional user-inputs: Δ specifies a certain degree of allowed deviation when checking the approximate linear dependency. δ describes

the degree of allowed jitter needed for deciding whether two subspaces are parallel.

2) Partitioning-based methods: ORCLUS [19], was the first clustering method that can find clusters not only in axes-parallel subspaces but also in arbitrarily oriented subspaces. It takes into account the eigenvectors of clusters and integrates them into a k-Means-like, iterative approach. Unlike LUCKe, it does not use all eigenvectors and can not detect clusters of higher dimensionality than l, a user input which is necessary additionally to the number of clusters k.

k-Planes [20] and the algorithm in [21] combine k-means with PCA-like approaches, but do not find correlation clusters. Instead, they aim to optimize k-Means so that clusters of different variance can be found.

3) Spectral method: Combining Spectral Clustering with local PCA yields an optimization of Spectral Clustering regarding intersecting groups of data in [22], but they do not perform correlation clustering.

4) LUCK: LUCK [3] allows finding exclusively one-dimensional linear correlations by using distance-based cluster algorithms. It uses an "orientation vector" for every point based on its kNN, where k is calculated dynamically based on a threshold τ. As the idea for LUCKe is based on LUCK, we discuss differences between LUCK and LUCKe in more detail in Sec. III-E4.

C. Other Correlation Clustering Algorithms

LMCLUS [23] considers linear manifolds as cluster centers. Histograms of the distances between sampled points and corresponding trial linear manifolds are used to partition the data set. LMCLUS has three parameters: *K* is an upper bound on the dimension of the linear manifolds in which the clusters could be embedded. *S* is the number of trial linear manifolds of a given dimensionality that are determined to find the best partitioning of a set of points. Γ is a threshold for the quality of separation and influences the composition of the clusters.

CASH [24] is based on the Hough Transform and transfers every point into a parameter space. With a grid-based approach, it finds dense regions in this parameter space, which correspond to many points lying on the same hyperplane in the original space. Like that, it refrains from using a PCA and needs only two parameters: $MinPts$ specifies the minimum number of points in a cluster and $MaxLevel$ specifies the maximum number of splits in the grid construction corresponding to the allowed degree of jitter.

III. LUCKe

In the following, we present the details of LUCKe a method that can be combined with any distance-based clustering algorithm to find correlation clusters. Algorithm 1 provides an overview of the individual steps of LUCKe.

First, the data set is scaled with min-max scaling. Secondly, LUCKe calculates eigenvalues and eigenvectors based on the kNNs for each point, see Sec. III-A We adapt the eigensystem meaningfully as described in Sec. III-B and define our similarity resp. distance measure (Sec. III-C), such that points lying in the same correlation cluster are similar resp. near to each other. With this novel idea, *basic* distance-based clustering algorithms can find correlation clusters instead of "traditional" clusters (Sec. III-D). In Sec. III-E we analyze diverse properties of the distance function.

Algorithm 1 LUCKe

Input: Data set $\mathcal{X} \in \mathbb{R}^{n \times d}$,
neighborhood size $k_{user} \in \mathbb{N}$,
Output: Distance Matrix $\mathcal{D}$

1: **function** LUCKE($\mathcal{X}$, k_{user})
2: $X = \text{scale}(\mathcal{X})$
3: // calculate eigensystem of all points
4: $k = \max(k_{user}, d)$
5: **for** every point $p \in X$ **do**
6: $N_p = \text{kNN}$
7: $E_p, V_p = \text{PCA}(N_p \cup p)$
8: **for** $(i = 1; i \leq d; i++)$ **do**
9: $\Omega(e_i) = \frac{e_i}{\sum_{j=1}^{d} e_j}$
10: $\vec{w}_i = \Omega(e_i) \cdot \vec{v}_i$
11: $W_p = (\vec{w}_1, \vec{w}_2, ..., \vec{w}_d)^T$
12: // calculate distance matrix
13: initialize(D)
14: **for** every point $p \in X$ **do**
15: **for** every point $q \in X$ **do**
16: $D_{pq} = \text{DISTANCE}\big((p, W_p), (q, W_q)\big)$ (Eq. 7)
17: **return** $Matrix$

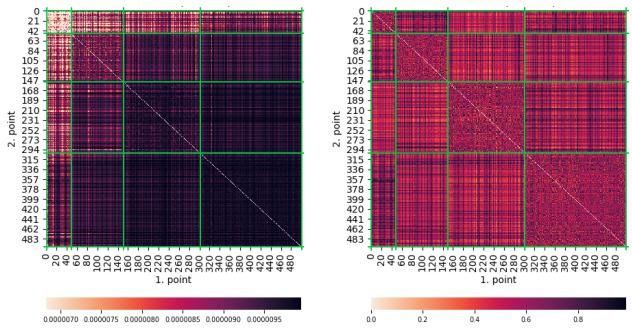
A. Computation of the Eigensystem

We perform a local PCA, (see, e.g., in [22], [25]) which is a PCA on the k nearest neighbors (kNN) of each point. It is important to consider that k should at least correspond to the dimensionality d of the data set: choosing $k < d$ results in a singular covariance matrix M [26] with at least one eigenvalue equal to zero, where the corresponding eigenvectors are not meaningful. That is connected to the fact that k points with $k < d$ always lie on a d-dimensional hyperplane. Thus, $k = \max(k_{userinput}, d) + 1$ is used, also taking into account that we include the point itself into its kNN.

B. Adaptation of the Eigensystem

We adapt the eigensystem by first normalizing the eigenvalues s.t. their sum is 1 (Sec. III-B1), and then multiplying them by their corresponding eigenvalues (Sec. III-B2).

1) Normalization of Eigenvalues: Depending on the density of the neighborhood of a point eigenvalues may differ considerably in size: the larger the k-distance, i.e., the distance to the k-th nearest neighbor, the larger are the eigenvalues of the local PCA. Eigenvalues do not only reflect the points' correlation, but rather the density of the neighborhood, as illustrated by the distance matrices in Fig. 2. They belong to a dataset consisting of points on a two-dimensional plane, which are split into four clusters of different densities. The clusters all have the same expansion and degree of correlation, but are differently



(a) Without normalization (b) With normalization

Fig. 2. Distances between points of differently dense clusters (indicated by green lines) before and after normalization

dense. The points are ordered according to their cluster and clusters are ordered ascendingly by their density for a better illustration. Note that all 4 "(classical) clusters" here belong to the same *correlation cluster*, as they lie on the same plane. Fig. 2(a) shows the pairwise distances calculated as described later in the section but based on the original eigenvalues. The intra-cluster distances (corresponding to the blocks on the diagonal of the distances matrix) increase with increasing density because of the related decreasing eigenvalues. Nevertheless, all planes are equally strongly correlated, thus it makes sense to normalize the eigenvalues and thereby eliminate the misleading influence of the density and the Euclidean distance. Comparability of different points' principal components can be reached by normalizing the eigenvalues $e_1, ..., e_d$ of every point separately with Ω as described below so that their sum is equal to 1.

$$\Omega(e_i) = \frac{e_i}{\sum_{j=1}^{d} e_j} \tag{1}$$

Normalizing the eigenvalues first leads to the distance matrix shown in Fig. 2(b), which has similar intra-cluster distances for all four clusters, allowing to detect them as the one correlation cluster they all belong to. An additional advantage of the normalization Ω is a uniform range of values, which later leads to a determinable value range of our similarity measure between 0 and 1.

2) Multiplication of Eigenvalues to Eigenvectors: To detect correlations, the relationship between the principal components of the neighborhood of a point is relevant: e.g., when ignoring the eigenvalues, two points can have the exact same eigenvectors even though they lie in perfectly linear correlations that are perpendicular to each other. Multiplying with the normalized eigenvalues results in a much more reasonable way to handle the principal components, as eigenvectors belonging to small eigenvalues have less influence. Thus, for a point with the eigenvalues $e_1, ..., e_d$ and the corresponding eigenvectors $\vec{v}_1, ..., \vec{v}_d$ a weighted eigenvector $\vec{w}_i$ is calculated as follows:

$$\vec{w}_i = \Omega(e_i) \cdot \vec{v}_i$$

C. Similarity and Distance Measure

To decide if points p and q with a similarly oriented neighborhood, i.e. a similar eigensystem, belong to the same hyperplane or to two different, similarly oriented, approximately parallel hyperplanes, the normalized connection vector between the points is regarded: $\vec{c} = \vec{pq}/|\vec{pq}|$

The normalization ensures that the Euclidean distance between the points does not affect the result, as points that lie on a common hyperplane are similar in the sense of correlation clustering even if they are spatially distant from each other. Additionally, we regard the matrix W_p resp. W_q for both points p and q, which holds the weighted eigenvectors of the respective point in its rows:

$$W = \begin{pmatrix} \vec{w}_1 & \vec{w}_2 & \dots & \vec{w}_d \end{pmatrix}^T$$

Multiplying $\vec{c}$ with the weighted eigenvector matrices W_p and W_q of both points yields similar vectors $\vec{u_p}$ and $\vec{u_q}$ if both points lie in the same correlation cluster, i.e., if the connection vector can be described by the most important principal components of both W's.

$$\vec{u}_p = W_p \cdot \vec{c}, \text{ and } \vec{u}_q = W_q \cdot \vec{c} \tag{2}$$

The entries of $\vec{u}$ correspond to the scalar product between the connection vector $\vec{c}$ and the weighted eigenvectors $\vec{w}_i$. As $|\vec{w}_i| = \Omega(e_i)$ and $\vec{w}_i$ and the unweighted eigenvector $\vec{v}_i$ differ only in length, the scalar product can be written as follows:

$$\vec{w}_i \circ \vec{c} = |\vec{w}_i||\vec{c}| \cdot cos\sphericalangle(\vec{w}_i, \vec{c}) = \Omega(e_i) \cdot cos\sphericalangle(\vec{v}_i, \vec{c}) \tag{3}$$

Thus, $\vec{u}$ is given by:

$$\vec{u} = \begin{pmatrix} \vec{w}_1 \circ \vec{c} \\ \vec{w}_2 \circ \vec{c} \\ \vdots \\ \vec{w}_d \circ \vec{c} \end{pmatrix} = \begin{pmatrix} \Omega(e_1) \cdot cos\sphericalangle(\vec{v}_1, \vec{c}) \\ \Omega(e_2) \cdot cos\sphericalangle(\vec{v}_2, \vec{c}) \\ \vdots \\ \Omega(e_d) \cdot cos\sphericalangle(\vec{v}_d, \vec{c}) \end{pmatrix} \tag{4}$$

Note, that for a fixed $\sphericalangle(\vec{v}_d, \vec{c})$, it holds that the larger e_i, the larger is the i-th value of $\vec{u}$. For a fixed e_i, it holds that the smaller $\sphericalangle(\vec{v}_d, \vec{c})$, the larger is the i-th value of $\vec{u}$. So $\vec{u}$ implies how strong the scattering of the kNN in the direction of $\vec{c}$ is. Two points belong probably to the same correlation cluster, if the kNN of both lie close to their connection vector.

Using absolute values when adding up the entries of the tensor product $T = \vec{u}_p \otimes \vec{u}_q$ yields the desired similarity between p and q while taking into account that the sign of the connection vector is arbitrary for our case.

$$sim\big((p, W_p), (q, W_q)\big) = \sum_{i=1}^{d}\sum_{j=1}^{d} \big|u_{p_i} u_{q_j}\big| = \sum_{i=1}^{d}\sum_{j=1}^{d} |t_{ij}| \tag{5}$$

where t_{ij} are the elements of the matrix $T = \vec{u}_p \otimes \vec{u}_q$.

As we in practice do not always operate on a set of points, but potentially have several points with the same coordinates

(the connection vector of which could not be normalized), we additionally define for the similarity of a point to itself:

$$sim\big((p, W_p), (p, W_p)\big) = 1 \tag{6}$$

As the resulting similarity lies always in the interval $[0, 1]$ (see Sec. III-E1) it can easily be converted into a distance measure:

$$dist\big((p, W_p), (q, W_q)\big) = 1 - sim((p, W_p), (q, W_q)) \tag{7}$$

D. Combination with Clustering Algorithms

The similarities resp. distances between all points can now serve as input for any distance-based basic clustering method, which then finds correlation clusters instead of "classic" clusters. We tested three different exemplary basic clustering algorithms: DBSCAN, Spectral Clustering, and Agglomerative Clustering. In the following, they are abbreviated in conjunction with LUCKe as follows: LUCKe+DBSCAN, LUCKe+Spectral and LUCKe+Agglomerative.

E. Properties

In the following we regard diverse properties of LUCKe: the similarity and distance measure's value range (Sec. III-E1), that it is a pseudometric (Sec. III-E2), the runtime complexity (Sec. III-E3) and differences to its conceptional predecessor LUCK (Sec. III-E4).

1) Value Range: The similarity as defined in Eq. 5 and thus also the distance (see Equation 7) is always in the interval $[0, 1]$: Regarding Equation 4 together with knowing that $|cos(\cdot, \cdot)| \leq 1$ and the definition of normalizing the eigenvalues in Equation **??** we see, that the sum of (absolute) entries of $\vec{u}$ is smaller than or equal to 1:

$$\sum_{i=1}^{d} |u_i| \overset{(4)}{=} \sum_{i=1}^{d} |\Omega(e_i) \cdot cos\sphericalangle(\vec{v}_i, \vec{c})| \leq \sum_{i=1}^{d} |\Omega(e_i)| \overset{(??)}{=} 1 \tag{8}$$

Applying this in Equation 5 directly yields the value range $[0, 1]$, which is desirable not only for easier access to the information and a high explainability for users of the distance/similarity measure but also to be able to compare similarities without looking at the whole data set.

2) Pseudometric: Some properties of a distance measure can be important. E.g., fulfilling the triangle inequality can allow acceleration via index structures, and symmetry of the distance matrix is even necessary for some cluster algorithms and other downstream tasks. Our distance measure as defined in Equation 7 fullfills both these properties. It is a pseudometric and, additionally, it is 0 if and only if two points have the same principal component with eigenvalue 1 (and thus all other eigenvalues are 0), which also corresponds to their connection vector. Proofs are left out for brevity, but can be found under https://tinyurl.com/jxw5afzs.

3) Runtime Complexity: The runtime complexity is $O(k^2 \cdot n^2)$ and is composed as follows (note, that $n \geq k \geq d$ holds): $O(n \cdot d)$ for the scaling of the data set with min-max scaling. $O(n^2 \cdot d \cdot k)$ for the kNN query for every point. $O(n \cdot k \cdot d^2)$ for calculating the $d \times d$ covariance matrix for every point. $O(n \cdot d^3)$ for the decomposition of every covariance matrix for the PCA. $O(n \cdot d^2)$ for scaling the eigenvalues and multiplying them with their eigenvectors. $O(n^2 \cdot d)$ for computing the connection vector between every two points. $O(n^2 \cdot d^2)$ for the multiplication of this vector to the weighted eigenvector matrix. $O(n^2 \cdot d^2)$ for calculating the tensor product and summing up its components. Considering index structures could further improve the complexity. Adding the complexity of the basic clustering algorithm yields the overall complexity.

4) Differentiation from LUCK: LUCK can only detect one-dimensional linear correlations. LUCKe can not only detect arbitrary dimensional correlations, but also several correlations of different dimensionality within the same data set without any expert knowledge on this topic. For that, LUCKe uses the established method local PCA, where LUCK created a proprietary "orientation vector" $\overrightarrow{o_{p_i}}$ for each point p_i. Where the distance measure used in [3] only compares these $\overrightarrow{o_{p_i}}$ with the connection vector between the points, we found a measure incorporating the *complete* eigensystems. After the computation of the kNN we do not use any spatial distance function, s.t. even far away points lying on the same hyperplane can be detected as similar. Additionally, all distances and similarities lie in the value range $[0, 1]$, which improves the comparability and explainability of LUCKe. Also, the time-consuming and suboptimal search for k in LUCK is replaced by a simple and robust user input.

IV. Experiments and Results

We evaluate LUCKe in combination with DBSCAN, Agglomerative Clustering, and Spectral Clustering as described in III-D. To test LUCKe systematically, we performed extensive experiments on synthetic data, of which we show the most meaningful results in Sec. IV-A. Sec. IV-B contains the results for real-world data sets and Sec. IV-C summarizes our results. To simulate expert knowledge needed for good parameter settings, we tested for every algorithm (competitors as well as combinations with LUCKe) depending on the properties of the data set, the number of different hyperparameters, and the robustness of the algorithm w.r.t. each parameter, between 9 and 585 (on average 200) different parameter settings via grid search, applying those yielding the highest NMI for our algorithms as well as for our competitors. All experiments with LUCKe are performed with a 1.8 GHz CPU and 16 GB RAM. For CASH, a few results were not calculated for $dim > 5$ resp. $dim > 11$, because of increased memory requirements.

For LUCKe+DBSCAN we directly used the distance matrix of LUCKe. For LUCKe+Agglomerative we tested *single linkage*, *complete linkage* and *average linkage* as described in Sec. II-A. For LUCKe+Spectral Clustering, we tested both, fully-connected graphs as well as kNN graphs based on the similarity matrix of LUCKe. We applied the normalized Laplacian [27], and used the number of clusters as user input for the number of important eigenvectors. For the final clustering step, discretizing the eigenvectors surpassed using k-means in almost all cases. Note that we do not combine LUCKe with

highly elaborated improvements of the clustering algorithms, but the basic versions.

A. Synthetic Data Sets

LUCKe is evaluated w.r.t. the data set size n, the number of dimensions dim, amount of noise $noise$, and jitter $jitter$. Jitter describes the extent of deviation from points to a hyperplane; with a hyperplane we describe a d^*-dimensional subspace with $d^* \leq d$ in this paper. We created five different base cases with default settings $n = 500$, $dim = 3$, $noise = 0$ and $jitter = 0$, shown in Fig. 3. For each experiment in this subsection we keep all but one parameter the same to detect potential dependencies on the adjusted parameter. To prevent overoptimism in our experiments, we decided to use the five base cases defined beforehand, instead of majorly using such which could only be clustered correctly using LUCKe. For example, spatially distant clusters as in datasets PH and HDD are easy to detect for our competitors which often rely on density-connected clusters but offer no such advantage for LUCKe. Also non-continuous hyperplanes, where several spatially distant groups of points belong to the same correlation as, e.g., the data set described in Sec. III-B1, are not tested here even though they would be advantageous for LUCKe.

Data set XL consists of two crossing straight lines. Data set XH contains two crossed hyperplanes. Data set PH contains two parallel hyperplanes. Data set XHDD contains a $(dim - 2)$- dimensional hyperplane traversing a $(dim - 1)$-dimensional hyperplane. Data set HDD contains dim hyperplanes of different dimensionality, which partially overlap in some dimensions. All clusters are of similar density – the higher the dimensionality of hyperplanes, the more points they contain. Values of all features lie in $[0, 1]$ for all datasets.

Sec. IV-A1 presents the results of the experiments for LUCKe with the different data sets and settings, to analyse the behaviour of our novel distance resp. similarity function. In Sec. IV-A2, we compare the results of LUCKe with those of other correlation clustering algorithms introduced in Sec. II.

For brevity, we show only the most interesting results in the Appendix A, i.e., the behavior of all algorithms on data sets XH, PH, XHDD, and HDD for varying dim and $jitter$. Further results can be found under https://tinyurl.com/jxw5afzs. The results for data set XL are not included in this paper, as they did not change significantly varying any of the parameters n, dim, and $jitter$, and all algorithms but 4C yielded constantly almost perfect results (except for increasing $noise$). Varying n and $noise$ did mainly yield similar NMIs to the base case settings for all algorithms, thus they are also left out.

1) Properties of LUCKe:

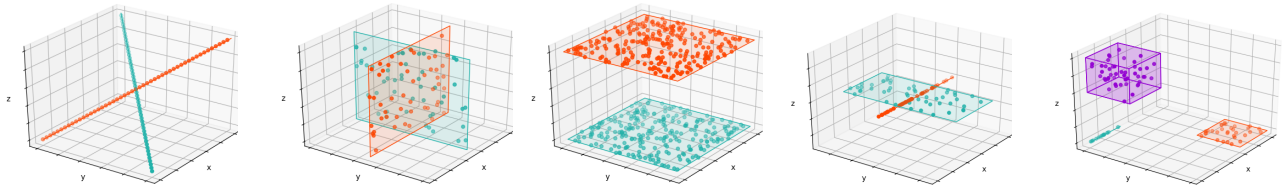
a) Efficiency: The average runtime to calculate the distance resp. similarity matrix on our base case data sets depending on the data set size n for diverse k is shown in Fig. 4. It fits our complexity calculation in Sec. III-E3.

b) Basic Setting: LUCKe is tested in combination with DBSCAN, Spectral Clustering and Agglomerative Clustering for the five data sets in the basic setting (see Fig. 3), results are shown in Fig. 5. For Data set XL and PH, all four algorithms yield very good results with an NMI between $0.95 - 0.98$ for Data set XL and an NMI of 1 for Data set PH. For Data set XH, LUCKe+Spectral and LUCKe+Agglomerative yield good results. Only a few points at the intersection of the hyperplanes are assigned to the wrong cluster. LUCKe+DBSCAN of course connects both crossing hyperplanes as they are density connected due to the points in the intersection, which can not be assigned uniquely to only one of the clusters. Results for Data sets XHDD and HDD are sensible to the number of dimensions, especially using DBSCAN: The distances between the points of a hyperplane become larger the higher the dimensionality of the hyperplane is. Detecting groups with differently high intra cluster distances means detecting groups with different densities, which, e.g., DBSCAN can not. Adding too much jitter to any data set with crossing clusters can make their detection impossible for all tested algorithms including the competitors.

c) Size of Data set: The five data sets are tested with data set sizes of $n = 1000$, $n = 2000$, $n = 3000$, $n = 4000$ and $n = 5000$ in addition to the default setting. The results regarding the basic setting do not change significantly with increasing n for any of the base cases.

d) Number of Dimensions: We tested all base case results with dimensionalities 4, 6, 9, 12, 20, and 35. All four algorithms detect crossing lines in Data set XL well even at $dim = 35$, while for Data set XHDD, none of the algorithms can produce a reasonable result as the number of dimensions increases, but neither do the competitors. Of all combinations with LUCKe and all competitors, parallel hyperplanes (PH) of very high dimensionalities with $dim > 12$ can only be detected by LUCKe+Spectral and ORCLUS: it is especially hard to detect those clusters distance-based, as distances between points of a dim-dimensional hyperplanes increase with the dimensionality, see Fig. 6.

e) Noise: Data sets with a noise proportion of 5%, 7.5%, 10%, 15% and 20% are applied in the tests. In general, LUCKe with k-Means, Spectal Clustering and the Agglomerative Clustering slightly deteriorates with increasing noise fraction, because the three clustering algorithms have no noise detection. Thus, all noise points are always assigned to a cluster, resulting in a lower NMI. Nevertheless, LUCKe with these three algorithms is able to detect the different correlated groups of points despite noise, if they detect them in the default setting of the data sets. DBSCAN is more robust against noise. However, the NMI also decreases here with increasing noise. This is mostly because noise points that are very close to points in a hyperplane are assigned to them, see, e.g., in Fig. 7 (left), where the first and the third dimension of the clustering result for LUCKe+DBSCAN for Data set PH with 15% noise is illustrated. Two planes (yellow and red) were found. For the upper (yellow) plane, some points were assigned to the plane that are not perfectly located on it. However, these points randomly generated as noise resemble data points with some degree of jitter. Therefore, it is actually desirable that they are added to the cluster. Noise points that are located between the two clusters are recognized as noise as well.



(a) Data set XL: two crossed straight lines

(b) Data set XH: two crossed hyperplanes

(c) Data set PH with two parallel hyperplanes

(d) Data set XHDD: two nested/crossed hyperplanes of different dimensionality

(e) Data set HDD: several hyperplanes of different dimensionality

Fig. 3. Five different synthetic data sets with color-coded similarly correlated data points

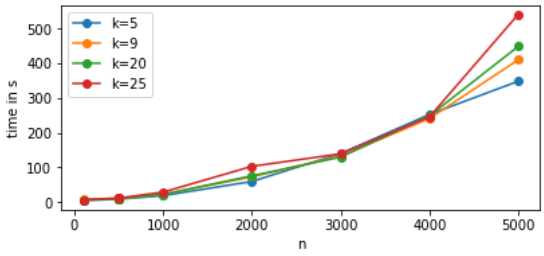


Fig. 4. Runtime depending on data set size n for different k

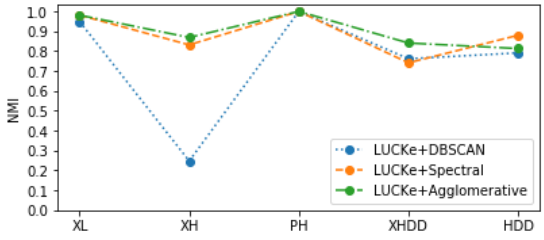


Fig. 5. Results for the basic setting for the five data sets and the four tested LUCKe variants

f) Jitter: For $jitter$ the values 0.05, 0.075, 0.1, 0.15 and 0.2 were tested. A value of $jitter = x$ means that the points scatter in a range of $[-x, x]$ around the hyperplane (all synthetic data sets are normed to a range of $[0, 1]$ in every dimension). The case $x = 0$ implies that the point lies perfectly on the hyperplane. As expected, the NMI of data sets becomes worse for more jitter. As of a value of $jitter = 0.1$, the points usually scatter strongly around the hyperplane, so that the original correlation in the data can not be detected anymore. However, with a small value for $jitter$, LUCKe is able to find meaningful clusters, see, e.g., in Fig. 7 (right), which shows the clustering result for LUCKe+Spectral on the first two dimensions of Data set XH with $jitter = 0.05$. The resulting NMI is only 0.53, even though almost all points are assigned to the correct cluster. The low NMI is due to points in the intersection, which cannot be assigned correctly to only one of the clusters as they are not unambiguously discernable. Here, a fuzzy approach could improve results.

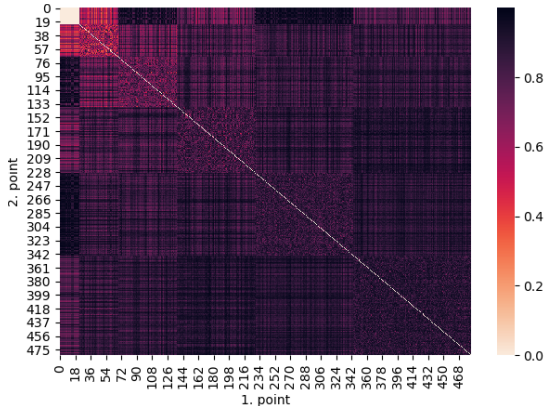


Fig. 6. LUCKe-distance matrix of points on hyperplanes of different (ascendingly ordered) dimensionalites in the 6-dimensional Data set HDD

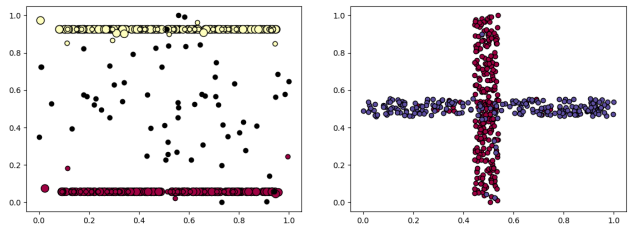


Fig. 7. Clustering results. Left: Data set PH with 15% noise. Right: Data set XH with $jitter = 0.05$.

2) Comparisons to Other Methods: In the following, LUCKe is compared with the correlation clustering algorithms described in Sec. II-B and II-C - ORCLUS, 4C, COPAC, ERiC, LMCLUS and CASH. All competitive algorithms are implemented in the Elki framework [28].

a) Data set XL: Comparing the algorithms with respect to Data set XL, LUCKe performs approximately as well as ORCLUS, COPAC and ERiC with each of its combinations. Only with an increasing amount of noise the values of LUCKe+Agglomerative and LUCKe+Spectral and deteriorate as they do not have any noise detection mechanism. However, ORCLUS cannot handle noise, too, for the same reason. For

4C, each tested parameter setting resulted in an NMI of 0.

b) Data set XH: For Data set XH, ORCLUS achieves the best results, closely followed by LUCKe+Agglomerative and LUCKe+Spectral. LMCLUS handles high noise levels best, but cannot deal with an increasing number of dimensions on this data set. As the data set contains crossing hyperplanes, they are not only density-connected but also with a smooth transition, which leads to a low NMI for LUCKe+DBSCAN and 4C, because they connect the clusters.

c) Data set PH: For Data set PH, a (nearly) perfect clustering result is obtained for different values of n for all algorithms. $jitter$ and dim have the most influence on the results. For increasing $jitter$, the NMI for all combinations of LUCKe drop, while ORCLUS, 4C, COPAC, and ERiC achieve an NMI of 1 even for $jitter = 0.15$. The clusters are separated clearly spatially and are thus easy to detect for our competitors, while simultaneously the increasing jitter leads to non-informative local PCAs for LUCKe. However, for increasing dimensionalities, LUCKe+Spectral and ORCLUS are by far the only algorithms that perform very well for the 35-dimensional data set.

d) Data set XHDD: Finding high-dimensional linear correlations which are overlapping or crossing is a difficult task. In the basic setting, none of our competitors reaches an NMI of higher than 0.67, where using LUCKe yields an NMI between 0.74 and 0.84. While LUCKe enables detecting both hyperplanes, other algorithms fail to distinguish them.

e) Data set HDD: Highly different dimensionalities of correlation clusters can result in differently dense clusters, which are especially hard to find for LUCKe+DBSCAN, see Fig. 6. As the clusters additionally are separated spatially, 4C and its successors reach higher NMIs than LUCKe. LMCLUS performs worse and CASH is comparable to LUCKe. Especially high degrees of noise and jitter can conceal the true clusters, where higher dim can emphasize the cluster structure, leading to better NMIs for some of the algorithms.

B. Real-world Data Sets

Of course, LUCKe is also applicable on real world data, as we exemplarily show in the following.

a) Image Segmentation Data: Fig. 8 shows a 2d-projection of the Image Segmentation data set [29], colored by label on the left and colored by cluster as obtained by LUCKe+Spectral (settings: complete graph, neighborhood size $k = 24$, seven clusters, NMI=0.55) on the right. The data set contains 30 randomly drawn "instances", i.e., fields of 3x3 pixels, of seven outdoor images, resulting in a data set with 210 instances and 19 different attributes, e.g., color values.

b) Hitters Data: The Hitters data set [30] contains aggregated information about the performance of baseball players, where the players' positions are used as labels. LUCKe+Agglomerative with $k = 6$ for the neighborhood size, *average linkage* and a distance threshold of 0.68 yields the best NMI with 0.49 for a clustering with eight clusters, three of which contain only one point, as shown in Fig. 9 (right). Fig. 9 (left) shows the "ground truth" with colors according to the positions of a player. Even though some positions are put together in one cluster on the right, we see that LUCKe enables finding the correlations. Especially, e.g., the green cluster on the bottom is interesting: it is split from the purple points on the lower right, as they belong to a higher-dimensional correlation than the green points on the lower left.

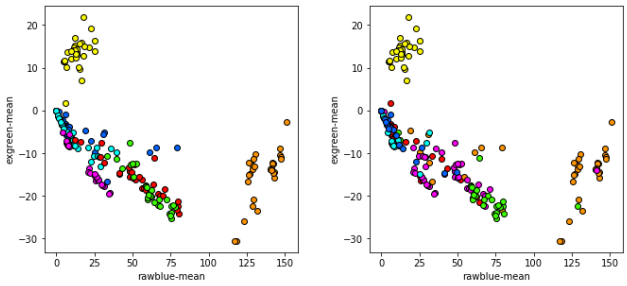


Fig. 8. Two descriptive dimensions of the Image Segmentation data set, colored by labels on the left, colored by clusters as found by LUCKe+Spectral on the right.

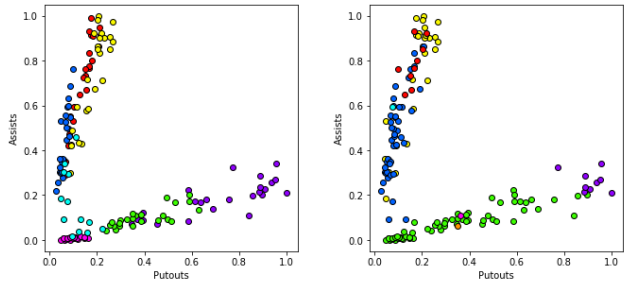


Fig. 9. Two descriptive dimensions of the Hitters data set, colored by labels on the left, colored by clusters as found by LUCKe+Agglomerative on the right.

C. Summary of Results

With a quadratic runtime w.r.t. the data set size LUCKe is scalable and the experiments showed that data set size does not influence the quality of the result significantly. For data sets with differently dimensional hyperplanes results seem to get worse for increasing dimensionality, because distances for data points in a high-dimensional hyperplane are higher than for data points in a hyperplane with lower dimensionality. This means, weakly correlated data points have a higher distance than strongly correlated, which is deliberate: like this, points which are not or only weakly correlated are not detected as correlation cluster. For strongly correlated clusters, LUCKe was able to detect even 35-dimensional correlation clusters. Thus, higher dimensionality of the data set in general does not imply worse results. For highly noisy data sets, it is of course recommendable to combine LUCKe with a basic clustering algorithm that is able to handle noise, like, e.g., DBSCAN, else the results get worse according to the performance of the basic clustering algorithm in combination with noise. On the

other hand, DBSCAN, which can not deal with differently dense clusters, is unfavorable to detect correlation clusters of highly different dimensionalities in the data set. For high levels of jitter as well as for only weakly correlated points, a large number of nearest neighbors k is recommendable to obtain a representative local PCA of the points. Intersecting hyperplanes (data set XH), were detected best by LUCKe and ORCLUS for all settings. Parallel hyperplanes (data set PH) with up to 35 dimensions were only found by ORCLUS and LUCKe+Spectral. For the complex data set XHDD, where a lower-dimensional hyperplane passes through a higher-dimensional hyperplane, LUCKe+Agglommerative achieved the overall best results for low dimensionalities.

Overall, LUCKe yielded even with only very basic clustering algorithms comparable results w.r.t. other correlation clustering algorithms and even surpassed them for explainable cases and settings.

V. Conclusion

We presented LUCKe, which gives us the opportunity to find even complex linear correlation clusters of arbitrary dimensionality using our favorite distance-based clustering algorithm. It builds the first highly advanced, generic bridge between classical clustering and correlation clustering. Where previous correlation clustering algorithms like 4C or COPAC already admittedly incorporate one classical clustering method or idea, LUCKe allows using a huge variety of classical clustering algorithms to find linear correlations. That opens a multitude of further applications: the complex problem of correlation clustering, where we neither know which arbitrary oriented and arbitrary dimensional subspaces are important, nor which points belong together, is now reduced to the problem of clustering, where only the second part of the problem needs to be solved. The research area of clustering is significantly further developed than the area of correlation clustering, thus correlation clustering can benefit largely from basic clustering, and LUCKe enables a wave of straightforward progression in this field.

References

[1] I. T. Jolliffe, "Principal components in regression analysis," in *Principal component analysis*. Springer, 1986, pp. 129–155.

[2] D. Mautz, W. Ye, C. Plant, and C. Böhm, "Discovering non-redundant k-means clusterings in optimal subspaces," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1973–1982.

[3] A. Beer, D. Kazempour, L. Stephan, and T. Seidl, "Luck-linear correlation clustering using cluster algorithms and a knn based distance function," in *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*, 2019, pp. 181–184.

[4] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96), Portland, OR*, 1996, pp. 226–231.

[5] C. Böhm, K. Kailing, P. Kröger, and A. Zimek, "Computing clusters of correlation connected objects," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, pp. 455–466.

[6] E. Achtert, C. Böhm, H.-P. Kriegel, P. Kröger, and A. Zimek, "Robust, complete, and efficient correlation clustering," in *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 2007, pp. 413–418.

[7] E. Achtert, C. Böhm, H.-P. Kriegel, P. Kröger, and A. Zimek, "On exploring complex relationships of correlation clusters," in *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*. IEEE, 2007, pp. 7–7.

[8] C. C. Aggarwal and C. K. Reddy, "Data clustering," *Algorithms and applications. Chapman&Hall/CRC Data mining and Knowledge Discovery series, London*, 2014.

[9] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[10] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[11] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[12] X. Y. Stella and J. Shi, "Multiclass spectral clustering," in *ICCV*. IEEE, 2003, pp. 313–319.

[13] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[14] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[15] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 169–194, 1998.

[16] S. Günnemann, I. Färber, K. Virochsiri, and T. Seidl, "Subspace correlation clustering: finding locally correlated dimensions in subspace projections of the data," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 352–360.

[17] E. Achtert, C. Böhm, P. Kröger, and A. Zimek, "Mining hierarchies of correlation clusters," in *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*. IEEE, 2006, pp. 119–128.

[18] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.

[19] C. C. Aggarwal and P. S. Yu, "Finding generalized projected clusters in high dimensional spaces," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 70–81.

[20] P. S. Bradley and O. L. Mangasarian, "K-plane clustering," *Journal of Global Optimization*, vol. 16, no. 1, pp. 23–32, 2000.

[21] J. C. R. Thomas, "A new clustering algorithm based on k-means using a line segment as prototype," in *Iberoamerican Congress on Pattern Recognition*. Springer, 2011, pp. 638–645.

[22] E. Arias-Castro, G. Lerman, and T. Zhang, "Spectral clustering based on local pca," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 253–309, 2017.

[23] R. Haralick and R. Harpaz, "Linear manifold clustering in high dimensional spaces by stochastic search," *Pattern recognition*, vol. 40, no. 10, pp. 2672–2684, 2007.

[24] E. Achtert, C. Böhm, J. David, P. Kröger, and A. Zimek, "Robust clustering in arbitrarily oriented subspaces," in *Proceedings of the 2008 SIAM International Conference on Data Mining*. SIAM, 2008, pp. 763–774.

[25] J. Yu, "Local and global principal component analysis for process monitoring," *Journal of Process Control*, vol. 22, no. 7, pp. 1358–1373, 2012.

[26] Z. Hu, K. Dong, W. Dai, and T. Tong, "A comparison of methods for estimating the determinant of high-dimensional covariance matrix," *The international journal of biostatistics*, vol. 13, no. 2, 2017.

[27] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[28] E. Achtert, H.-P. Kriegel, and A. Zimek, "Elki: a software system for evaluation of subspace clustering algorithms," in *International Conference on Scientific and Statistical Database Management*. Springer, 2008, pp. 580–585.

[29] C. L. Blake and C. J. Merz, "UCI machine learning repository," 1998, [Online] accessed: 26-January-2021. [Online]. Available: http://archive.ics.uci.edu/ml

[30] D. Meyer, A. Zeileis, and K. Hornik, *vcd: Visualizing Categorical Data*, 2020, r package version 1.4-8.

APPENDIX

A. Data set XH

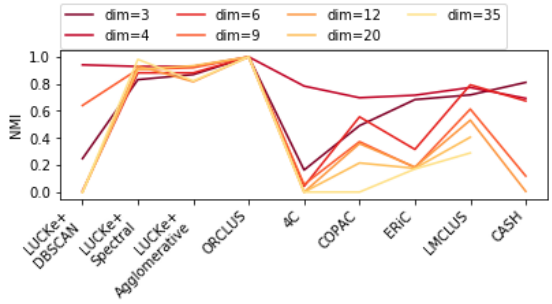


Fig. 10. NMI for different dim for Data set XH

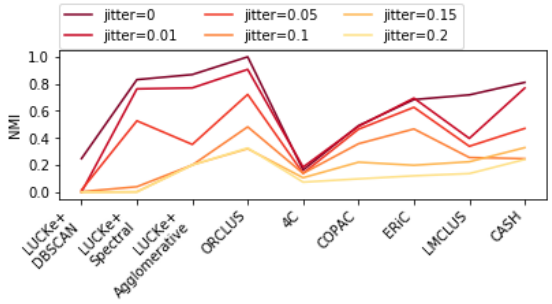


Fig. 11. NMI for different $jitter$ for Data set XH

B. Data set PH

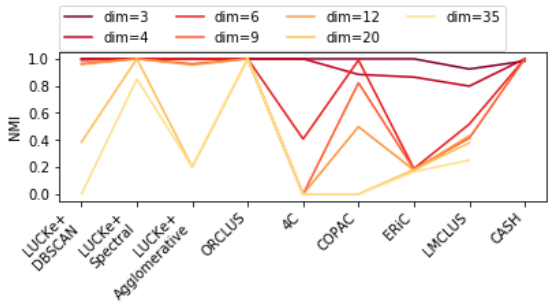


Fig. 12. NMI for different dim for Data set PH

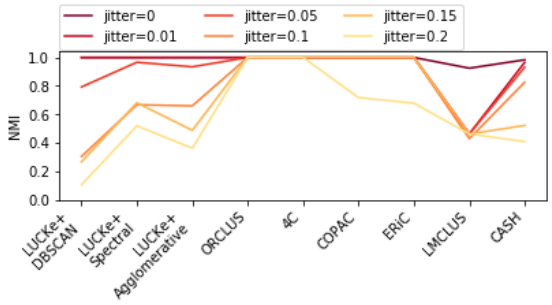


Fig. 13. NMI for different $jitter$ for Data set PH

C. Data set XHDD

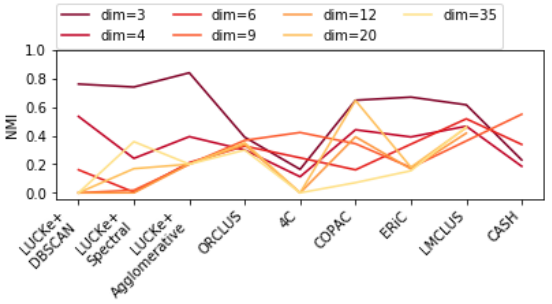


Fig. 14. NMI for different dim for Data set XHDD

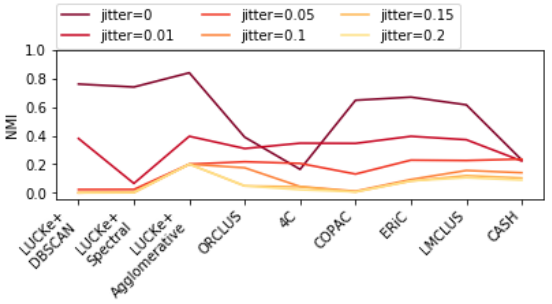


Fig. 15. NMI for different $jitter$ for Data set XHDD

D. Data set HDD

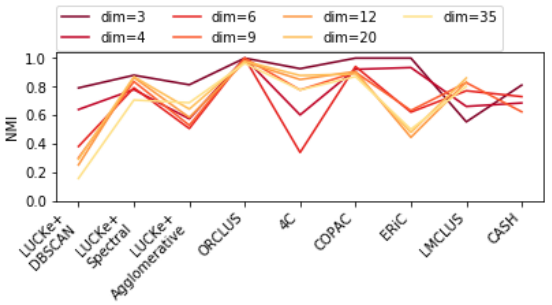


Fig. 16. NMI for different dim for Data set HDD

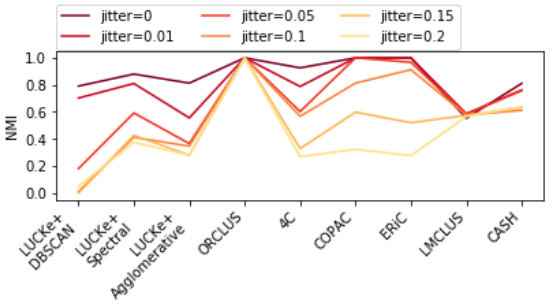


Fig. 17. NMI for different $jitter$ for Data set HDD

3.4 “Chain-detection for DBSCAN”

Publication: Janis Held, Anna Beer, and Thomas Seidl. “Chain-detection for DBSCAN”. in: *BTW 2019 - Datenbanksysteme für Business, Technologie und Web (Workshops)*. Gesellschaft für Informatik, Bonn, 2019, pp. 173–183. DOI: 10.18420/btw2019-ws-18

Statement of Originality: I noted the necessity of the method introduced in this paper, sketched the concept, and discussed it with Thomas Seidl. Janis Held refined the idea within the context of his Bachelor’s Thesis, implemented it, and performed the experiments.

Note: We received the best student paper award at BTW for this paper.

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.

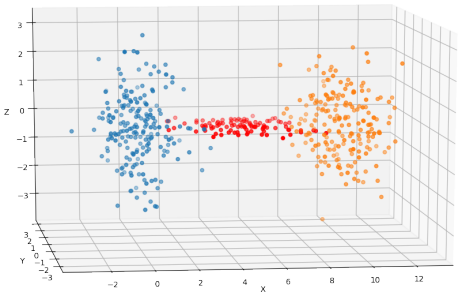


Fig. 1: The red points cause a density-connection between the intentional two clusters and thus form a chain.

[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.

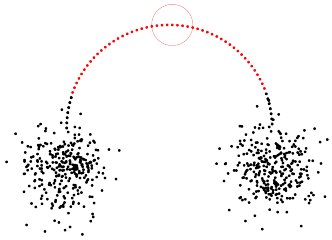


Fig. 4: The red dots may or may not be a chain, depending on the user. The red circle is one of the ϵ ranges.

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.

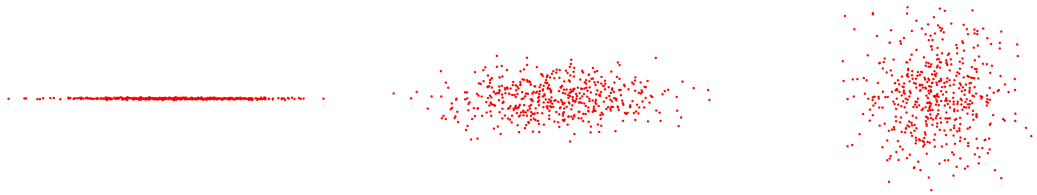
Like in Section 3.2, let $N = \{n_1, ..., n_m\}$ be the ϵ range of some point $p \in C$ and $\lambda_1, ..., \lambda_d$ the descending sorted normalized eigenvalues of the covariance matrix Θ corresponding to N. To calculate how good N lies within a $chainDim$ dimensional subspace, one calculates the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

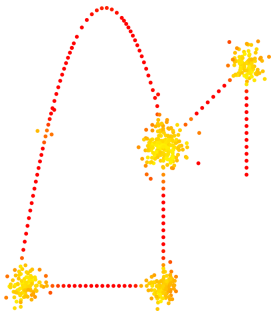


Fig. 6: Example data: Each point is colored by the normed error $\bar{e}$ derived from its ϵ range. Yellow means the error is close to 1 and red means it is close to 0.

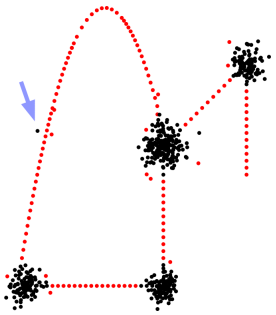


Fig. 7: Example data: With $allowedVariation = 0.2$ the red points are selected as shape-based chain-point candidates. The arrow highlights an outlier.

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

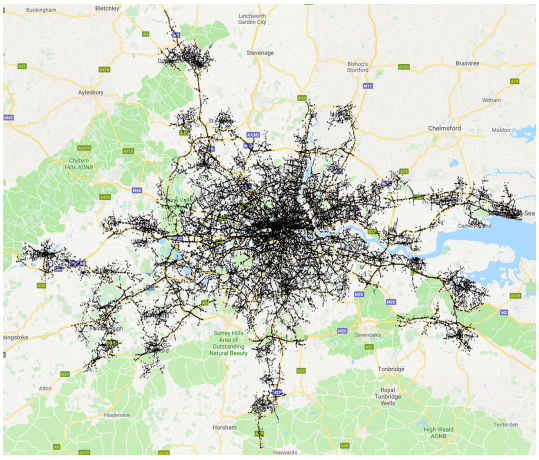


Fig. 8: The cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. The dots are stretched to fit the underlying map.

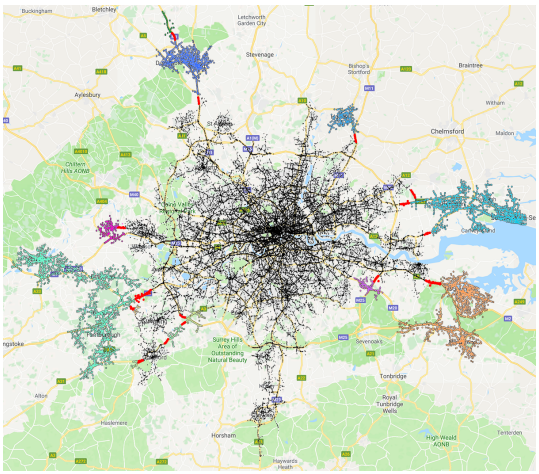


Fig. 9: Chain-detection applied on the cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. Chains are marked red.

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.

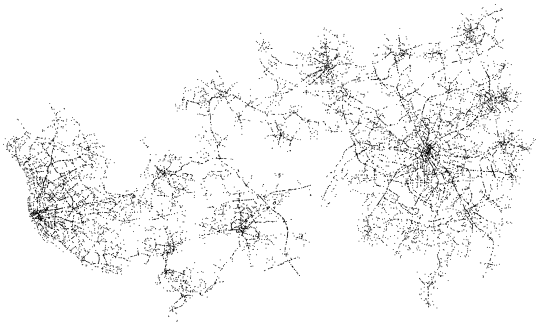


Fig. 10: The cluster of traffic accidents at Liverpool and Manchester.

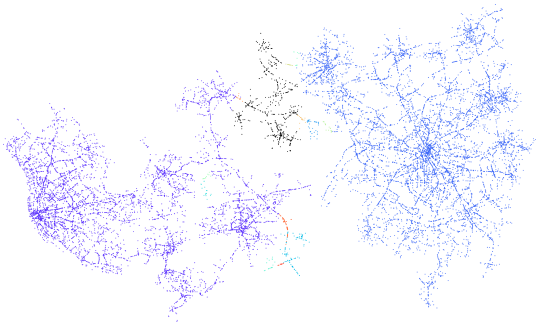


Fig. 11: Result of the chain-detection algorithm applied on the traffic accidents in Liverpool and Manchester.

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.

[He11] He, Y.; Tan, H.; Luo, W.; Mao, H.; Ma, D.; Feng, S.; Fan, J.: Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In: Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on. IEEE, pp. 473–480, 2011.

[JC16] Jolliffe, I. T.; Cadima, J.: Principal component analysis: a review and recent developments, 2016, URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4792409/, visited on: 01/11/2019.

[RSM07] Ruiz, C.; Spiliopoulou, M.; Menasalvas, E.: C-dbscan: Density-based clustering with constraints. In: International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing. Springer, pp. 216–223, 2007.

3.5 “Chain-detection Between Clusters”

Publication: Janis Held, Anna Beer, and Thomas Seidl. “Chain-detection Between Clusters”. In: *Datenbank-Spektrum* 19.3 (2019), pp. 219–230. DOI: 10.1007/s13222-019-00324-9

Statement of Originality: Janis Held adapted the algorithm and experiments we performed for the previous paper [47]. We adapted and extended the paper in collaboration.

Note: This paper is a generalization of the method presented in the previous Section 3.5 ([47]).

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

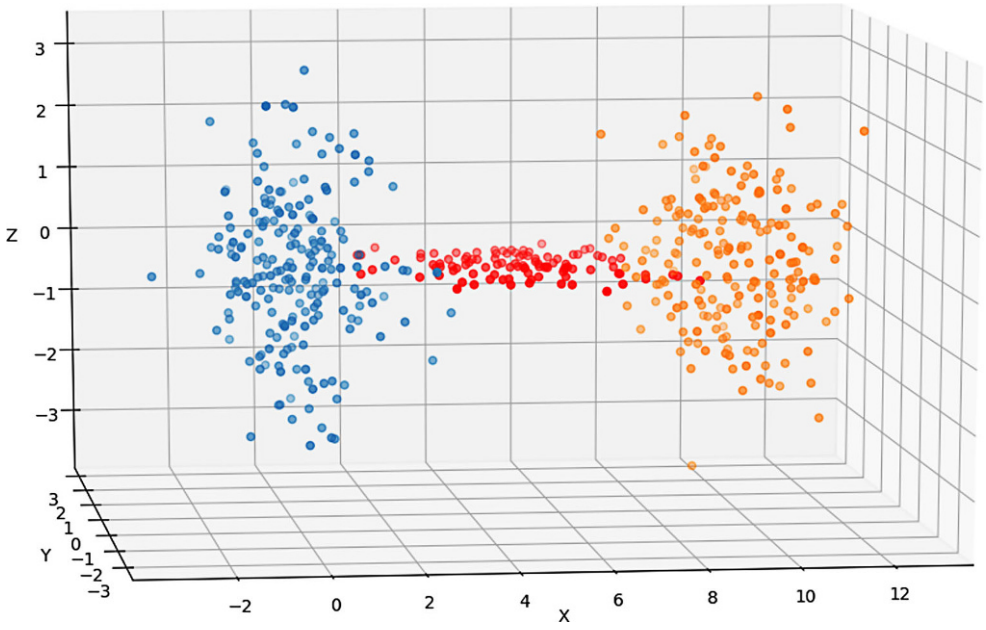


Fig. 1 The *red points* cause a density-connection between the intentional two clusters and thus form a chain

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

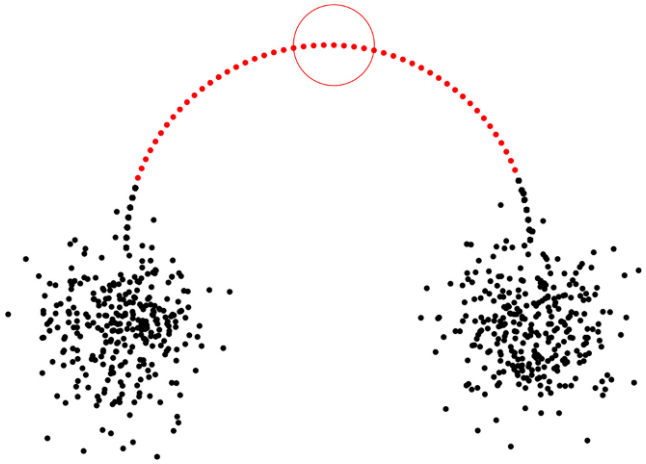


Fig. 4 The *red dots* may or may not be a chain, depending on the user. The *red circle* is one of the ε ranges

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.

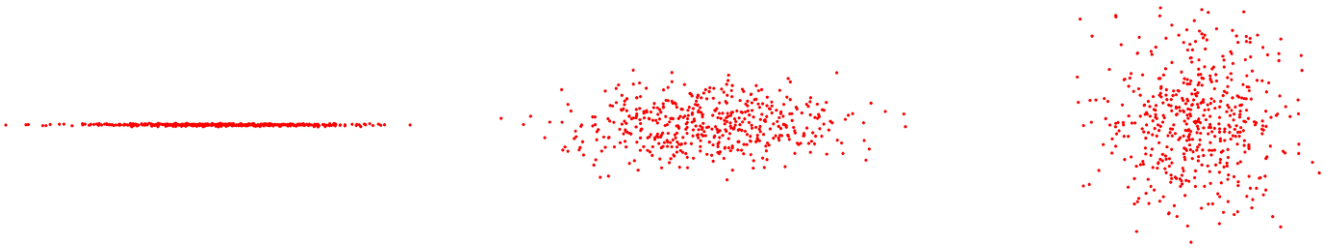


Fig. 5 Various degrees of fuzziness for normalized errors: $\bar{e} \approx 0.0002$, $\bar{e} \approx 0.1563$, and $\bar{e} \approx 0.9997$

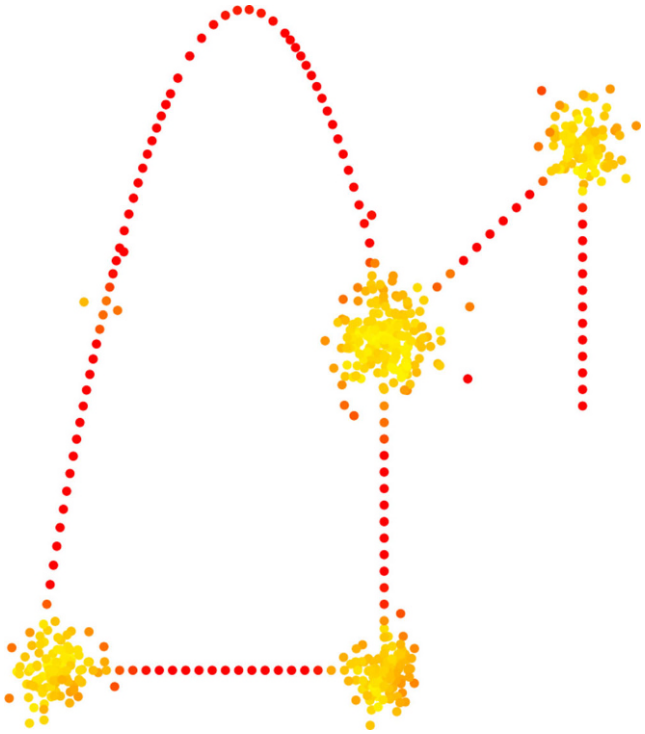


Fig. 6 Example data: Each *point* is colored by the normed error $\bar{e}$ derived from its ε range. *Yellow* means the error is close to 1 and *red* means it is close to 0

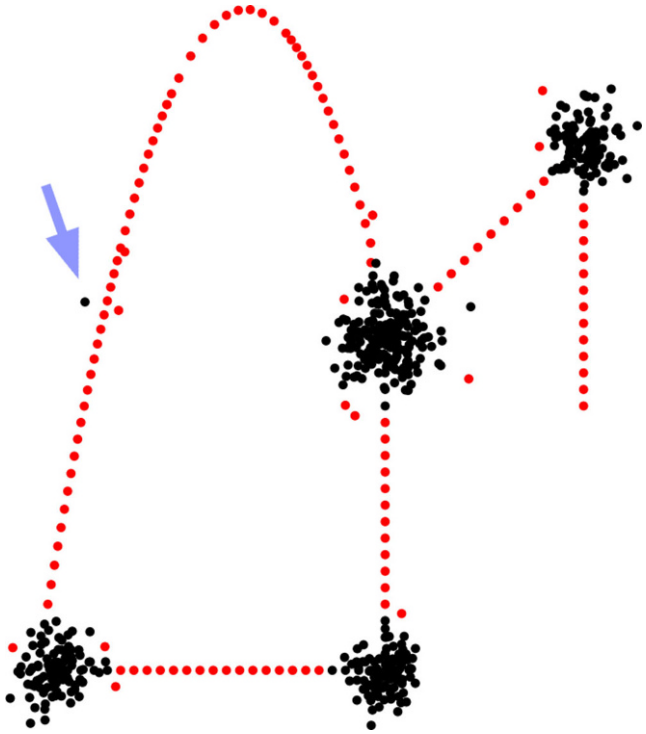


Fig. 7 Example data: With *allowedVariation* = 0.2 the *red points* are selected as shape-based chain-point candidates. The *arrow* highlights an outlier

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
```

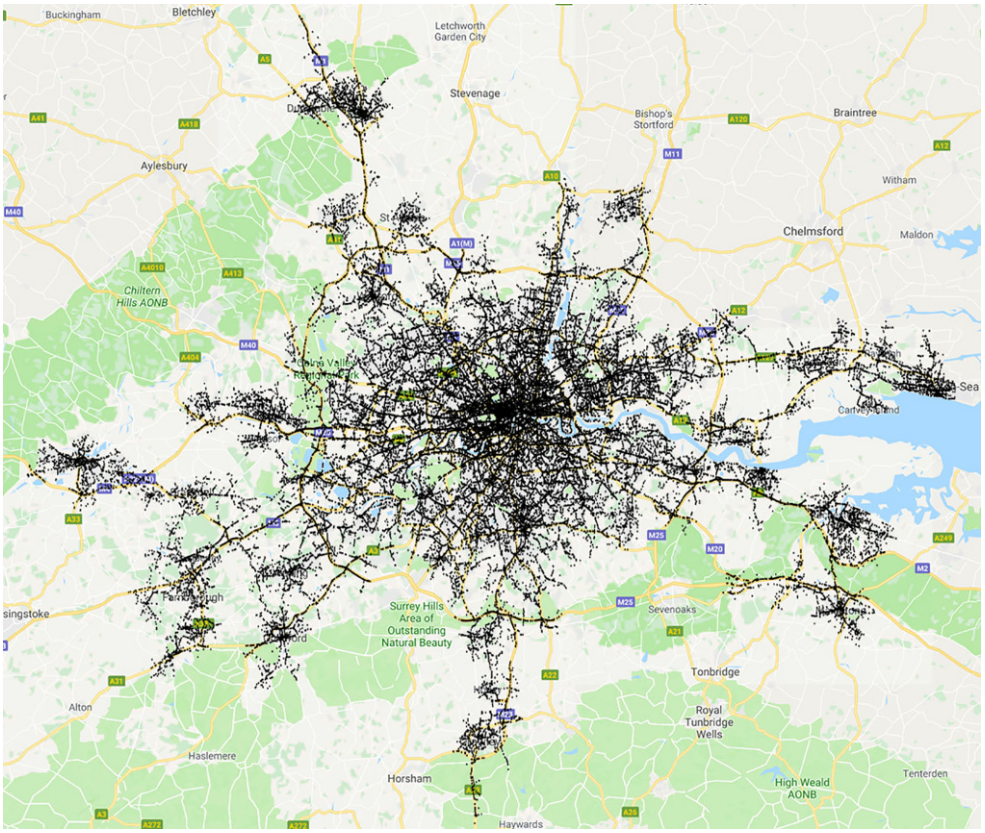


Fig. 8 The cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. The *dots* are stretched to fit the underlying map

$\bar{e}$ 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

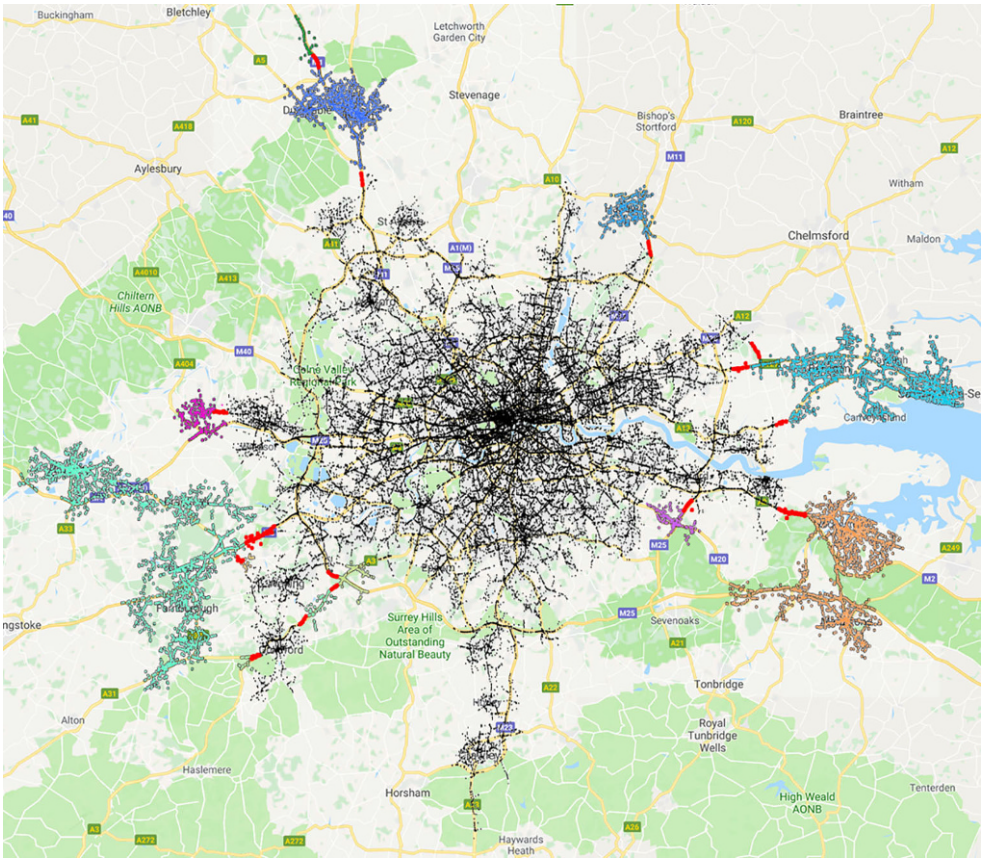


Fig. 9 Chain-detection applied on the cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. Chains are marked *red*

$\{q \in C | dist(p,q) \leq \varepsilon\}$. 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

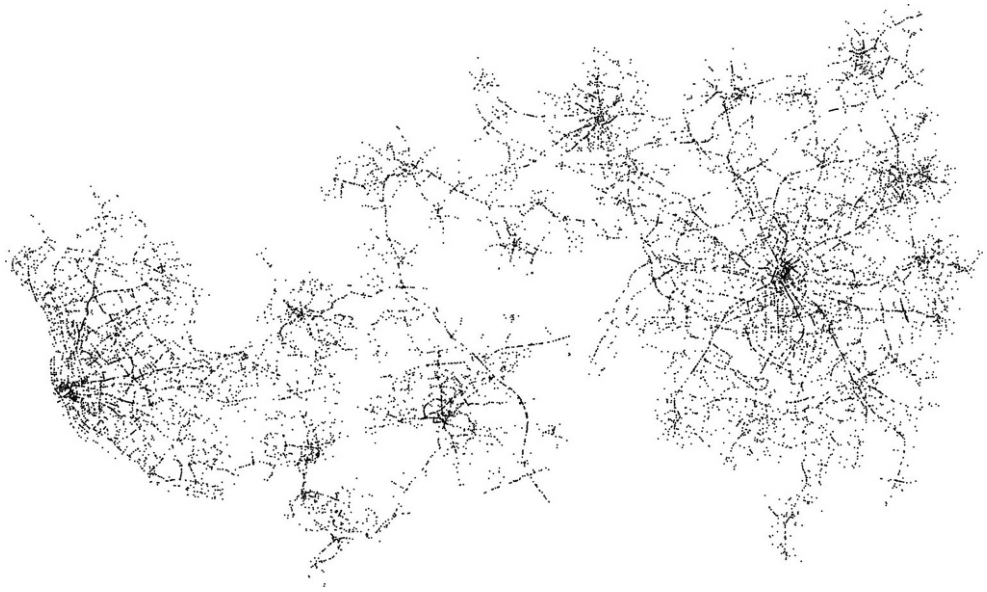


Fig. 10 The cluster of traffic accidents at Liverpool and Manchester

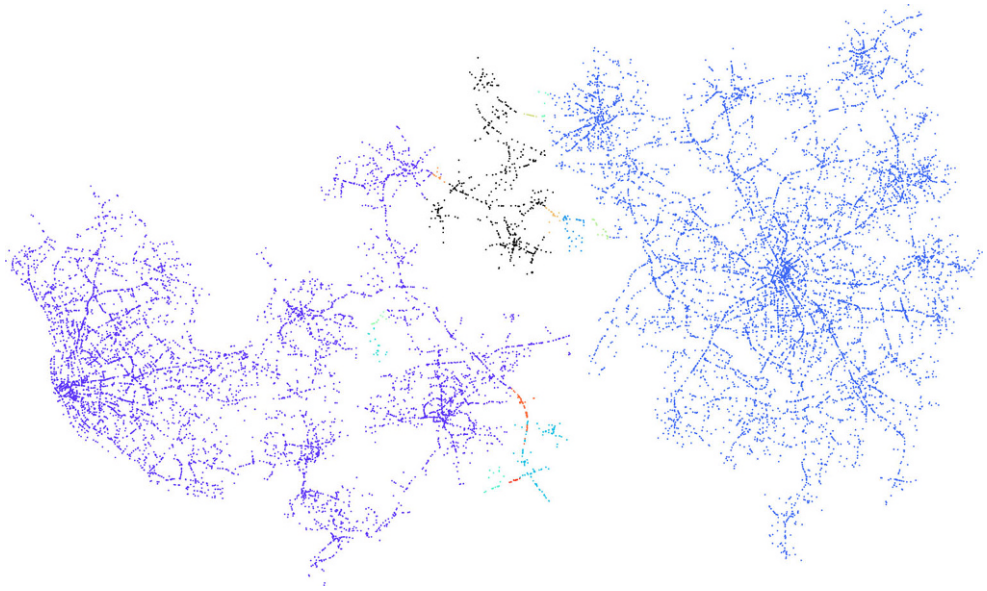


Fig. 11 Result of the chain-detection algorithm applied on the traffic accidents in Liverpool and Manchester

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

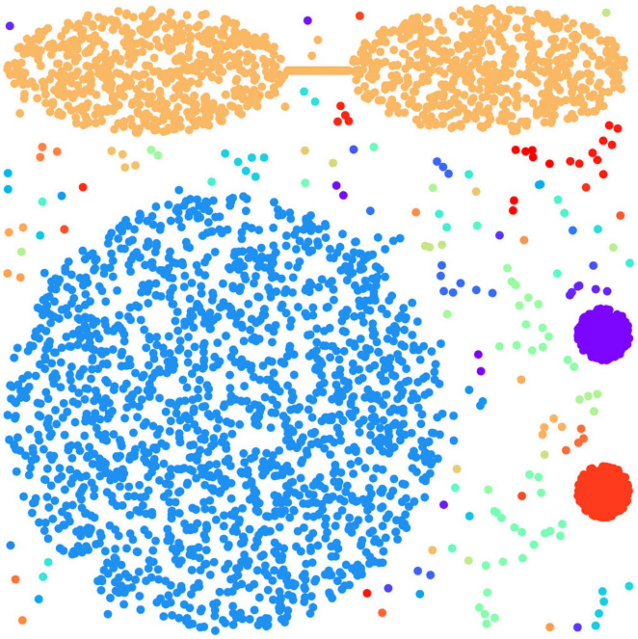


Fig. 12 Optimal single link clustering of cure-t2-4k with NMI score of ~ 0.8196

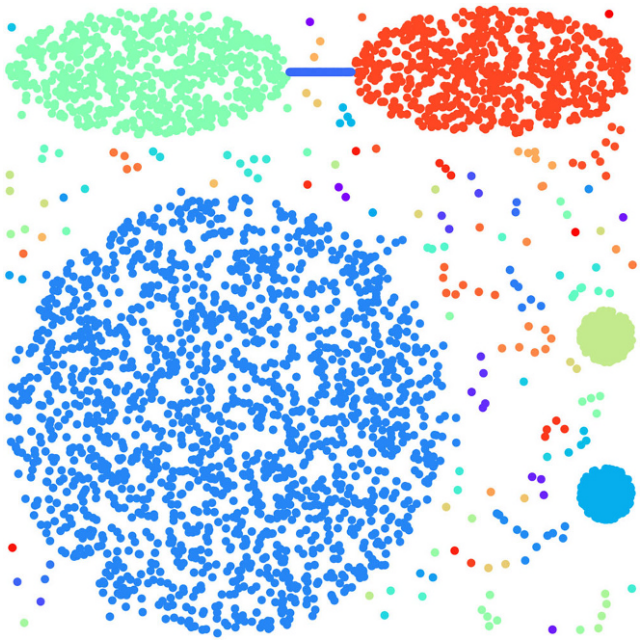


Fig. 13 Optimal single link clustering in combination with chain-detection of cure-t2-4k with NMI score of ~ 0.9382

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.

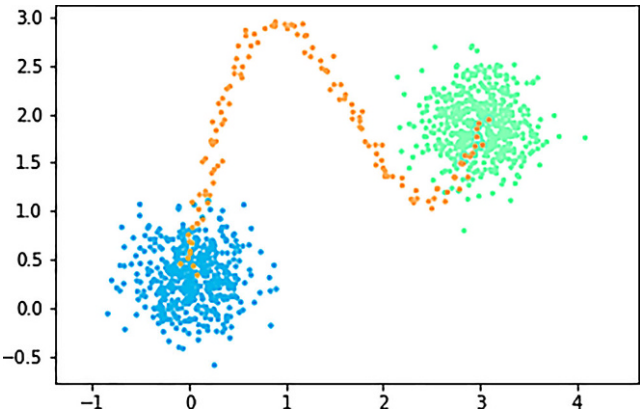


Fig. 14 Some random generated dataset with standard deviation of 0.05 for the chainpoint offset

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

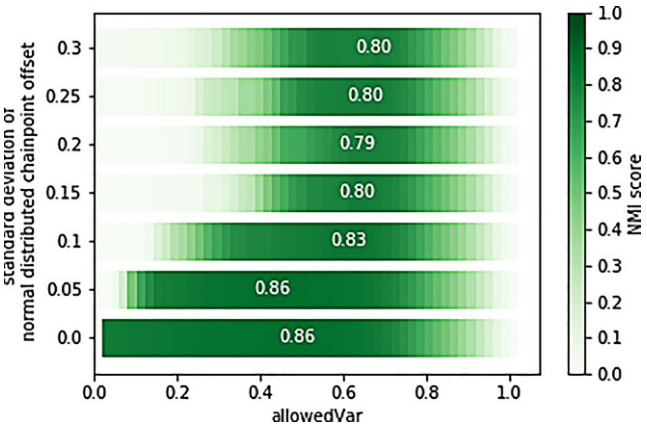


Fig. 15 NMI scores for DBSCAN + chain-detection with different chain densities and *allowedVariation* parameters. The *white number* is the highest NMI score found

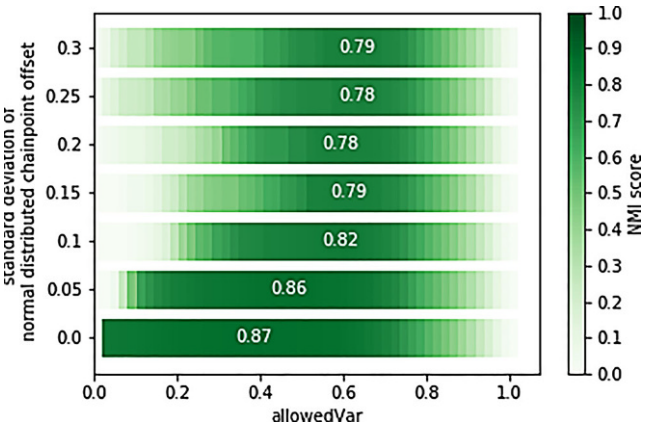


Fig. 16 NMI scores for single-link + chain-detection with different chain densities and *allowedVariation* parameters. The *white number* is the highest NMI score found

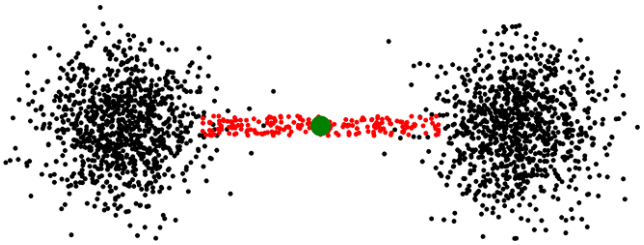


Fig. 17 The *red dots* will not be detected as a chain, because the ε range (marked as a *green circle*) is too small to detect the chain

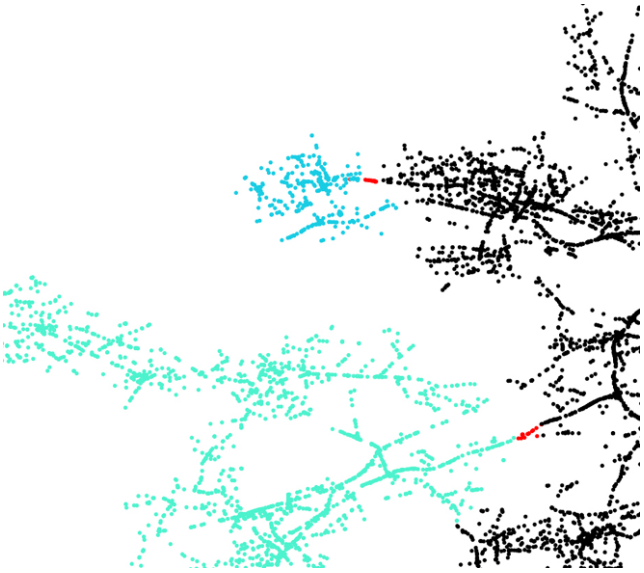


Fig. 18 Zoom in of Fig. 20c with $\varepsilon := 0.15$. Chains are marked *red*

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

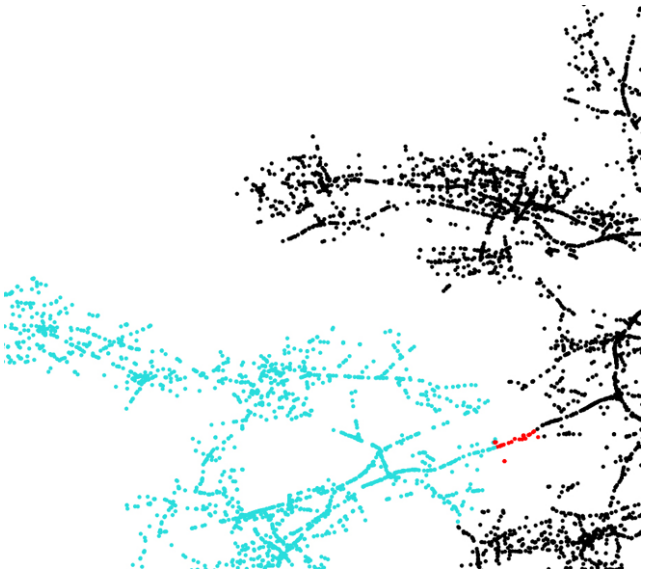


Fig. 19 Zoom in of Fig. 20d with $\varepsilon := 0.2$. Chains are marked *red*

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-

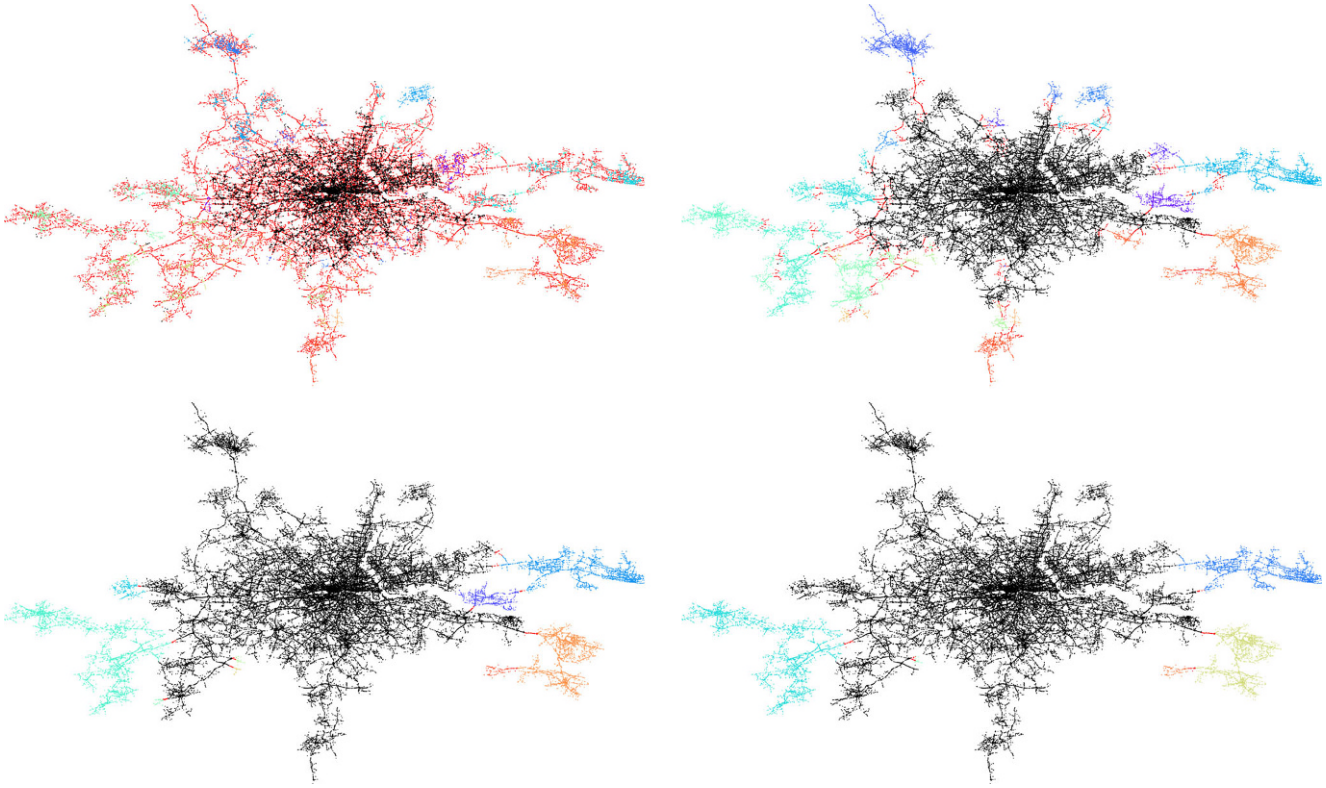


Fig. 20 Chaindetection in the London dataset for different values $\varepsilon = \{0.025, 0.05, 0.15, 0.2\}$

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

Chapter 4

Between Clustering and Outlier Detection

This chapter includes the following publications:

1. Anna Beer, Jennifer Lauterbach, and Thomas Seidl. "MORe++: k-Means Based Outlier Removal on High-Dimensional Data". In: *Proceedings of the International Conference on Similarity Search and Applications (SISAP)*. Springer, Cham. 2019, pp. 188–202. DOI: 10.1007/978-3-030-32047-8_17

2. Anna Beer, Dominik Seeholzer, Nadine-Sarah Schüler, and Thomas Seidl. "Angle-Based Clustering". In: *Proceedings of the International Conference on Similarity Search and Applications*. Cham: Springer International Publishing, 2020, pp. 312–320. DOI: 10.1007/978-3-030-60936-8_24

4.1 "MORe++: k-Means Based Outlier Removal on High-Dimensional Data"

Publication: Anna Beer, Jennifer Lauterbach, and Thomas Seidl. "MORe++: k-Means Based Outlier Removal on High-Dimensional Data". In: *Proceedings of the International Conference on Similarity Search and Applications (SISAP)*. Springer, Cham. 2019, pp. 188–202. DOI: 10.1007/978-3-030-32047-8_17

Statement of Originality: I developed the concept of the algorithm and guided Jennifer Lauterbachs Master's Thesis after consultation with Thomas Seidl. Jennifer Lauterbach implemented the concept, performed the experiments I designed, and helped me writing the paper.

Erratum: Instead of the running time of k-means-- the complexity of the problem it solves is given in the paper on page 3.

MORe++: k-Means Based Outlier Removal on High-Dimensional Data

Anna Beer( ), Jennifer Lauterbach, and Thomas Seidl

Ludwig-Maximilians-Universität München, Munich, Germany
{beer,seidl}@dbs.ifi.lmu.de, j.lauterbach@campus.lmu.de

Abstract. MORe++ is a k-**M**eans based **O**utlier **Re**moval method working on high dimensional data. It is simple, efficient and scalable. The core idea is to find local outliers by examining the points of different k-Means clusters separately. Like that, one-dimensional projections of the data become meaningful and allow to find one-dimensional outliers easily, which else would be hidden by points of other clusters. MORe++ does not need any additional input parameters than the number of clusters k used for k-Means, and delivers an intuitively accessible degree of outlierness. In extensive experiments it performed well compared to k-Means-- and ORC.

Keywords: Outlier detection · High-dimensional · Histogram-based · K-means

1 Introduction

As outlier detection in general delivers valuable results for fraud detection, medical problems, or finding errors in data, most techniques do not regard the plethora of attributes which is gathered for each data point in modern applications. An outlier, which is often defined as "an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism" [14], is more difficult to find in high-dimensional data than in low-dimensional, since the mechanisms generating data are difficult to identify in high-dimensional data due to the curse of dimensionality. Thus, most classic outlier detection algorithms are not applicable to high-dimensional data. Density based algorithms for example, are not meaningful in high-dimensional data, which usually is per se sparse. Also angular based outlier factors like, e.g., ABOD [18], are not interpretable anymore for high-dimensional data. Moreover, most of those algorithms do not scale with the number of dimensions.

Thus we introduce MORe++ (k-*M*eans-based *O*utlier *Re*moval using k-Means++), a fast method to score outliers in high-dimensional data. We achieve scalability w.r.t. the number of dimensions and retain explainability of the scores by regarding each dimension separately. In contrast to other methods we can even find clusters or outliers overlapping in some dimensions, since we do not

G. Amato et al. (Eds.): SISAP 2019, LNCS 11807, pp. 188–202, 2019.
https://doi.org/10.1007/978-3-030-32047-8_17

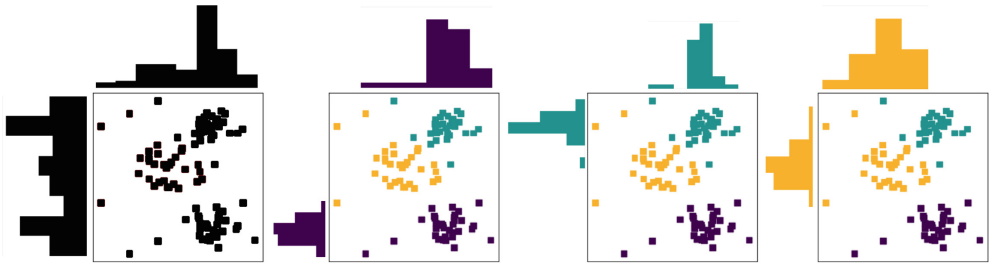


Fig. 1. Histogram of complete dataset in black, in contrast to histograms of points belonging to the same centroid according to k-means in green, yellow and purple. (Color figure online)

regard all points at once, but only those which are in one cluster according to k-Means. Using histograms, we accelerate our algorithm even further in regards to the number of points. Figure 1 shows how overlapping in a dimension prevents finding outliers if regarding the whole dataset at once, which is why we regard only a part of the datapoints at once.

Summarizing, our main contributions are:

1. We introduce a meaningful score for outliers in high-dimensional data
2. Our method is fast and scales linearly with the number of dimensions and points
3. It is based on k-Means and compatible to a lot of k-Means extensions
4. It is easy to implement
5. It is easily parallelizable and suitable for high-dimensional data, since it does not rely on distance measures operating on the full-dimensional space.

The remainder is structured as follows: in Sect. 2 we give an overview over other k-Means extensions and outlier detection methods using k-Means or a histogram-based approach. We also investigate diverse approaches of histogram segmentation. The complete algorithm is explained in detail in Sect. 3. In Sect. 4 we examine our algorithm regarding a plethora of aspects in overall 40 synthetic as well as real data experiments. Section 5 concludes this paper giving a short summary and prospect to promising future work.

2 Related Work

We first give the foundations looking at k-Means Clustering and existing recent extensions in Sect. 2.1. As there are already several methods combining k-Means and outlier removal, we give an overview over those in Sect. 2.2 and discuss the advantages of MORe++ in contrast to them. In Sect. 2.3 we look at histograms and outlier detection algorithms using them. We note that regarding only the projections of the complete dataset at once cannot lead to an effective outlier detection.

2.1 K-Means Clustering and Extensions

k-Means [19,20] is one of the most famous clustering algorithms and is still frequently used for diverse tasks. Given the number of clusters k, centers are randomly initialized in the original algorithm. All points are assigned to their closest center and the cluster centers are recomputed. Those two steps are repeated until no point changes its cluster membership any more and the algorithm converges against a local minimum of the mean distance from points to their cluster centers.

There exist several improvements of k-Means. For example, k-Means++ [2] optimizes the initial cluster centers by regarding the shortest distance to already chosen cluster centers. We will use this extension for our algorithm MORe++, as it usually improves the quality of clustering and reduces the variance of results. Another improvement is k-Median [5], which uses the median instead of the mean when calculating the new cluster centers to minimize the negative impact of outliers. Nevertheless, this comes at the cost of an increased runtime. On the other hand, kmeans|| [3] reduces runtime by parallelizing k-Means++. Instead of sampling single points for the initialization like k-Means++, $O(k)$ points are sampled $O(\log n)$ times. Also high-dimensional data can be clustered better with diverse variants of k-Means developed for subspace clustering, like NR-kmeans [22] or Sub-kmeans [21]. Where Sub-kmeans finds a "clustered" space containing all structural information and a noise space, NR-kMeans looks for an optimal arbitrarily oriented subspace for each partition. Those improvements, of which we apply only k-Means++ in this paper, are compatible to MORe++ and will be regarded in future work.

2.2 Outlier Detection and K-Means

There are several algorithms combining k-Means and outlier removal, of which we introduce the most common ones in the following. In Sect. 4 we will compare MORe++ to the first both introduced, k-means-- [7] and ORC [13].

k-means-- [7] combines outlier removal and k-Means by alternately removing outliers and performing k-Means iterations. In every step l points which are farthest away from their center are removed from the dataset for the next calculation of cluster centers, where l is given by the user. Even for $k = 1$, its running time is $O(n^{d^3})$, which is infeasible for high-dimensional data.

ORC (Outlier Removal Clustering) [13] assigns an outlyingness factor o_i to every point after a complete pass of k-Means. Points with o_i higher than a user given threshold T are removed from the set of points, and k-Means is performed again. o_i is, similar to k-means--, based on the distance to the nearest cluster center, and normalized by division by the highest distance between a point and its center. The algorithm is quite sensitive to the choice of T, and our experiments will show that ORC cannot handle high dimensional data well.

NEO-K-Means [28] considers outliers and overlapping clusters, for which it requires two parameters α and β. Using those parameters, it strives for a "trade-off between a clustering quality measure, overlap among the clusters, and

non-exhaustiveness (the number of outliers not assigned to any group)" [28] in every k-Means step. Reaching this trade-off requires iterations until convergence, there seems to be neither an upper bound for the running time, nor do the authors perform a complexity analysis. The main criterion is again the distance between points and their closest center, which becomes more and more useless with increasing number of dimensions.

KMOR [10] uses an additional cluster for all outliers, needing two parameters to control the number of outliers. A point is considered an outlier if the distance to *all* cluster centers is at least $\gamma \times d_{avg}$, which forbids finding local outliers.

Other methods combining k-Means and outlier detection are, e.g., ODC (Outlier Detection and Clustering) [1], where outliers are points having a distance to their cluster centers larger than p times the average distance. CBOD [17] and [16] are two-stage algorithms, where [16] additionally creates a minimum spanning tree on which they work on. [29] is a three stage algorithm first finding local outliers, then global outliers, and lastly combining clusters with similar densities and overlapping clusters.

All mentioned algorithms have in common that they rely on distance measures in the full-dimensional space, usually the Euclidean distance. As with increasing number of dimensions, all distances become similar due to the curse of dimensionality [4], the results get distorted for high-dimensional data. In contrast, MORe++ does not need any distance measure working on high-dimensional space. Additionally, due to the separate consideration of each dimension, it is already faster than these methods plus it is easy parallelizable.

2.3 Outlier Detection with Histograms

As using histograms is an established way to simplify data, several construction possibilities regarding the bin-width and bin-quantity exist: One of the most common possibilities, Sturges' rule [15], tends to oversmooth histograms and does not work well with large datasets and not normally distributed data. Other common rules are Scott's rule [26] and Freedman and Diaconis's rule [9], which are both better for larger samples. MORe++ uses Scott's rule which suggests h as the number of bins: $h = \frac{3.45\hat{\sigma}}{n^{1/3}}$, where $\hat{\sigma}$ is the sample standard deviation.

There are methods using histograms to find outliers: HBOS [12] constructs a histogram for each dimension and calculates an anomaly score for each data instance using the inverse estimated densities and supposing feature independence. [11] finds sparse regions in the dataset using histograms and a nearest neighbour approach. Based on those regions, local outlier candidates are identified, which can be removed from the set of outliers in a later optional reconsideration phase.

Looking at higher dimensional datasets and subspace clustering, objects may belong to different clusters in different subspaces, thus they could be outliers in some subspaces, but not in others. OutRank [27] addresses this issue by introducing a "degree of outlierness", the outlier rank. With that, points which are only in a subset of attributes anomalies, can also be detected as outliers [23]. In contrast, many outlier detection algorithms, like for example, LOCI (local

correlation integral) [24], look for outliers in the full dimensional space, or even micro-clusters. LOF [6], the local outlier factor, is another approach returning the degree of outlierness. It regards the isolation of a point with respect to the surrounding neighborhood, and was even extended for high-dimensional data [18,30].

3 MORe++

In the following we describe and analyze the outlier detection algorithm MORe++ in detail: Sect. 3.1 gives an overview, Sect. 3.2 explains how we find one-dimensional outliers in histograms, and Sect. 3.3 gives a complexity analysis.

3.1 Outline of MORe++

Based on the idea already shown in Fig. 1, MORe++ regards the points of every k-Means cluster separately. Like that, one-dimensional projections already enable the detection of outliers. Section 3.2 explains how to get one-dimensional outliers based on the according histogram. The higher the number of dimensions in which a point is considered an outlier, the higher is its outlier score. Thus, MORe++ finds a degree of outlierness. Algorithm 1 describes our approach in detail: on the basis of the clustering returned by k-Means++, we build a histogram for every dimension for every cluster. The method $calculate1dOutliers$ returns one-dimensional outliers for each dimension and each cluster given the according histogram, as explained in Sect. 3.2. The outlier score is the relation between the number of dimensions in which the point is considered a (one-dimensional) outlier and the total number of dimensions. Users can now either use this degree of outlierness, or give a threshold *ost* (outlier score threshold), so that points with an outlier score higher than *ost* are outliers in the full dimensional space. This approach delivers several advantages:

1. Our distance measure does not get skewed with increasing number of dimensions, as we regard every dimension separately
2. We can easily parallelize the calculation of outliers as we regard all clusters and also all dimensions independently. Thus, MORe++ is suitable for many points as well as for high- dimensional data.
3. Users do not have to know the number of outliers beforehand
4. A degree of outlierness gives more information than a hard classification
5. The threshold users *can* give is quite intuitive, as it is simply the minimal percentage of dimensions in which a point should be a one-dimensional outlier. As experiments will show, MORe++ is quite robust w.r.t. *ost* (we use the same value $ost = 0.2$ for 35 out of 40 experiments in total), thus a hard classification using a fixed *ost* is also a promising idea for future work
6. MORe++ is very fast with only $O(nd)$, where, e.g., k-means-- is exponential.

Algorithm 1. Pseudo-Code of MORe++

```
Data: Data X, number of clusters k
Result: Clustering labels, outlierScore for data points
foreach x ∈ X do
    numberOfOutlierDims[x] ← 0 ;
end
foreach c ∈ clusters do
    foreach d ∈ range(dimensions) do
        Build histogram;
        1dOutliers ← calculate1dOutliers(histogram);
        foreach 1dOutlier ∈ 1dOutliers do
            numberOfOutlierDims[1dOutlier] + +;
        end
    end
end
foreach x ∈ X do
    outlierScore ← numberOfOutlierDims[x]/dimensions ;
end
```

3.2 Detecting One-Dimensional Outliers in Histograms

To detect one-dimensional outliers in histograms, it is important that we only look at points assigned to one cluster by k-Means. Else, points of other clusters would cover outliers in the one-dimensional projections, as Fig. 1 already showed: see for example the outlier in the bottom middle, which is later in the first, purple cluster. Using histograms of the complete data, it is covered in both dimensions by the purple resp. the yellow cluster. Looking at the histogram of only the points assigned to the first (purple) cluster for dimension 1 (horizontal), it can be detected quite easily using the following approach:

If there are empty bins in the histogram, as shown in Fig. 2 on the left, then we partition the data along these empty bins. If there are no empty bins, we split the data where the height of the bins changes most from one bin to the next, relatively to the higher bin of both, as can be seen in Fig. 2 on the right. If there are several changes $s_0, ..., s_j$ which are (relatively) equally high, then we perform the split in the middle at $s_{\lfloor j/2 \rfloor}$. After the dataset is partitioned, all points which are not in the partition containing the majority of the points are marked as outliers for this dimension.

3.3 Complexity Analysis

For n points of dimensionality d the complexity of k-Means itself is $O(nkdi)$ with i the number of iterations until convergence and k the number of clusters. We build a histogram for every cluster and every dimension, which sums up to $O(kdb)$ for histograms with $b < n$ bins. Using Scott's rule for the construction of histograms (see Sect. 2.3), $b \in O(n^{-\frac{1}{3}})$. One-dimensional outliers are calculated

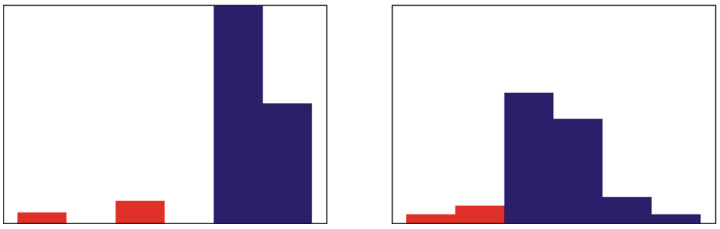


Fig. 2. Outliers (marked red) are detected using either empty bins or the highest relative difference between two adjacent bins. (Color figure online)

in $O(b+n) \subseteq O(n)$ and the calculation of all outlier scores can be done in $O(nd)$. Thus, MORe++ lies with $O(nd + kdn^{-\frac{1}{3}}) \subseteq O(nd)$ in a smaller complexity class than k-Means itself and is only linear in the number of dimensions as well as in the number of points. Furthermore, it is easily parallelizable.

In contrast, the running time of k-means– is with $O(n^{d3})$ much larger. ORC, which delivers clearly worse results than MORe++, needs to run j iterations of k-Means alternately with determining the outlyingness factor, which is in $O(nd)$, resulting in a total runtime of $O(j * (nkdi + nd)) \subset O(j * nkdi)$.

4 Experiments

We performed several experiments regarding the quality of outlier detection compared to ORC and k-Means-- (see Sect. 2), based on the ROC AUC (Area Under the Receiver Operating Characteristic Curve) value [8][1] and F1-measure; due to the lack of space and a high similarity of the results we only show the former here. All synthetic datasets were constructed using cluster centers drawn from a uniform distribution function and generating Gaussian distributed clusters around them. Outliers were added following a uniform distribution function.

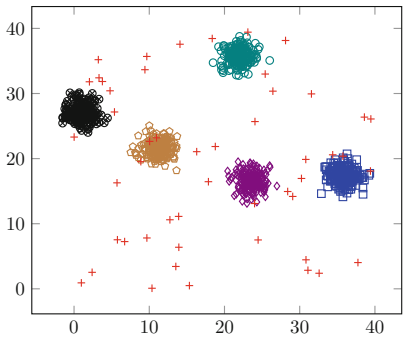


Fig. 3. 2d-projection of the base case experiment

In Sect. 4.1 we examine the influence of the following aspects onto the results of MORe++: size of dataset n, number of dimensions dim, percentage of outliers out, variance of clusters var, number of clusters k, and percentage of additional noise dimensions dim_n. For that, we created a base-case shown in Fig. 3 from which we kept all parameters but one at a time to investigate MORe++'s behaviour regarding that one aspect. In Sect. 4.2 we regard the behaviour of the algorithms on some real world datasets. They show, that even though MORe++ and k-Means-- deliver similar results

[1] Reminder: ROC AUC ranges from 0 to 1, where a perfect outlier prediction is 1. It regards the true positive rate vs. false positive rate. If ROC AUC is 0.5, the model has no class separation capacity.

in most of the previous test series, MORe++ clearly outperforms k-Means-- for real world data.

4.1 Influence of Various Aspects

To evaluate the behaviour of MORe++ regarding several aspects, we created a base case experiment as explained above with the following parameters: size of dataset $n = 1000$, number of dimensions $dim = 5$, percentage of outliers $out = 5\%$, variance of clusters $var = 1.2$, number of clusters $k = 5$, and percentage of additional noise dimensions $dim_n = 0$. Figure 3 shows two arbitrary dimensions of the base-case, where the ground truth outliers are marked by red crosses and clusters by colored shapes other than crosses. In each test series we changed exactly one parameter of that base case and compared the results to those of ORC and k-means--, where the base case is always marked with box brackets in the x-axis. To improve comparability, we also kept the (initially randomly chosen) cluster centers of the generated clusters the same, where possible.

As k-means-- is non-deterministic, we took the average of 100 executions. MORe++ and ORC are deterministic due to the use of cluster centers as in k-Means++ [2]. For each test series and each algorithm, we chose the parameter resulting in the best ROC AUC value after testing values from 0 to 1 in steps of 0.1 for the parameter *ost* in MORe++ and T in ORC, which resulted for both parameters and most of the test series in a value of 0.2, otherwise the best parameter settings are given in the according experiments. For k-Means-- the parameter l was taken from the ground truth (i.e. $l = 50$ for all experiments but the ones were the percentage of outliers or the size of the dataset was changed).

Experiments Regarding Number of Points. To examine MORe++'s behaviour with respect to the size of the dataset, we tested the base case with different values for the number of points $n = 500, 1000, 2500, 5000, 10000, 100000$. The results can be seen in Fig. 4 and show that MORe is better or comparable to ORC and k-Means-- in most cases. For 10000 points and only 5 dimensions, relatively many outliers are overlapping with the cluster itself, which explains the slight decline of results for a very high ratio between dimensions and points.

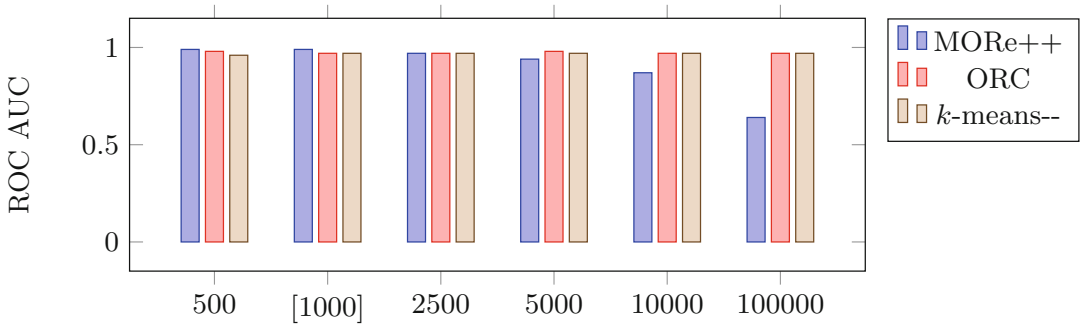


Fig. 4. ROC AUC Score for increasing number of points n

As MORe++ was developed for high-dimensional data, and especially for a high dimension to data ratio, this slight decline is predictable as well as manageable.

Experiments Regarding Number of Dimensions. We increased the number of dimensions up to 3000, which is a ratio of dimensions to data points of 3. For a lot of use cases, like data mining of textual data or image processing, the number of dimensions usually exceeds the number of data points. That often constitutes a problem for outlier detection algorithms as ORC: for a higher dimensionality than 30, the results of ORC[2], like it can be seen in Fig. 5 worsen a lot and from 1000 dimensions on, a constant ROC AUC of 0.6 is reached, which is only slightly better than guessing (which would be a ROC AUC of 0.5). k-Means-- on the other hand performs effectively as good as MORe++, and for both of them there is no decrease of quality subject to the dimensionality; they both perfom almost perfectly in high-dimensional space. For $dim \geq 5$, the ROC AUC for MORe++ is always at least 0.99, for k-Means-- the same holds for $dim \geq 30$. Note, that all clusters in this test series are in the full dimensional space, thus very similar results of MORe++ and k-Means-- were expectable.

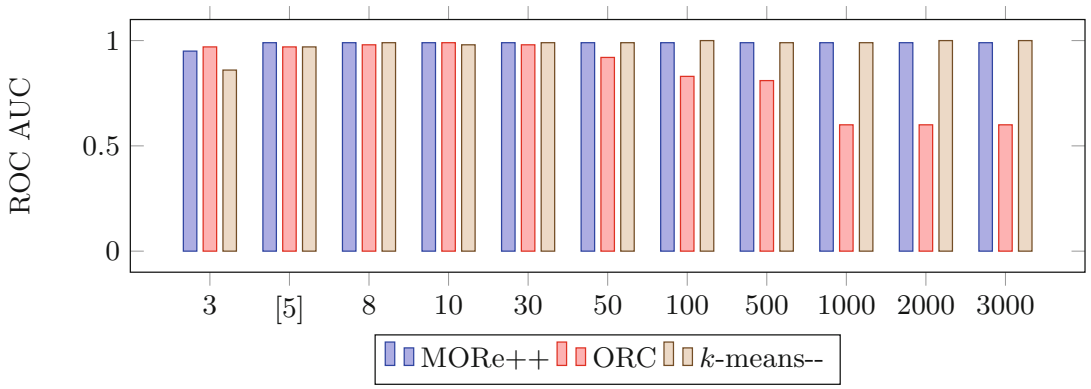


Fig. 5. ROC AUC Score for increasing number of dimensions dim

Experiments Regarding Percentage of Outliers. As Fig. 6 shows, MORe++ becomes less accurate for increasing number of outliers as well as ORC. A high percentage of outliers is difficult to handle well using our approach of finding one-dimensional outliers in histograms, as high amounts of outliers smoothen the one-dimensional histograms. But those high percentages of outliers constitute rather noise than some interesting, outlying points, which MORe++ aims to find, thus, this is more a question of where "outlierness" ends and "noise" starts.

[2] Best values for T: 0.8 for $dim = \{50, 100\}$, 0.9 for $dim = \{100, 500\}$, else $T = 0.2$.

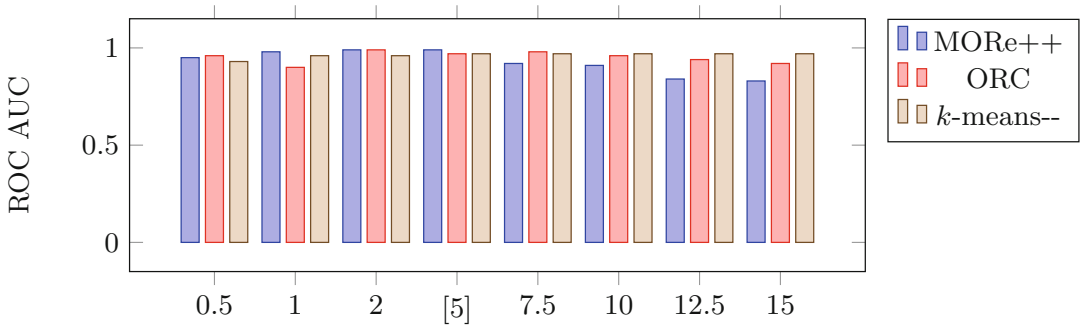


Fig. 6. ROC AUC Score for different outlier percentages

Experiments Regarding Variance of Clusters. For different variances $var = 0.7, 1.0, 1.2, 1.5, 2.0$ of the clusters MORe++ reached almost constant results, where ORC and k-Means-- get noteably worse with increasing variance, as Fig. 7 shows. That is, because with increasing variance the (full-dimensional) distances from points to their centers increase, too, thus they are more similar to the distances from outliers to cluster centers. As MORe++ does not rely on distinguishing high-dimensional distances of non-outliers and outliers to cluster centers, it is able to cope very well with diverse variances, in contrast to the comparative methods.

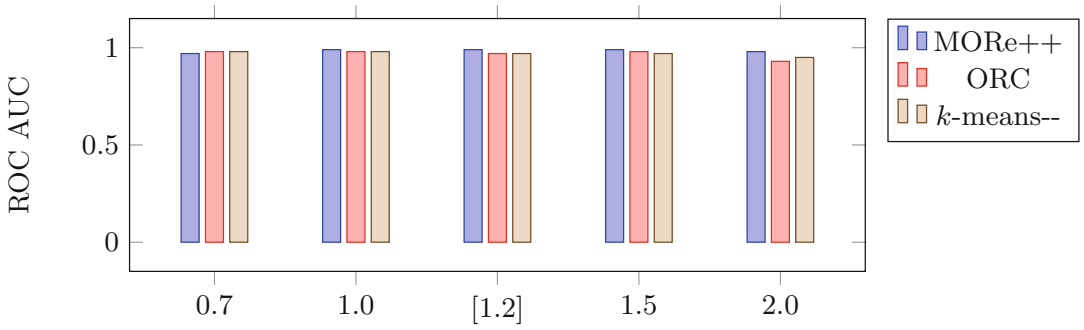


Fig. 7. ROC AUC Score for different cluster variances var

Experiments Regarding Number of Clusters. With increasing number of clusters, there are more overlapping clusters, thus k-Means and also all of the tested outlier detection algorithms[3] become worse, as can be seen in Fig. 8. But in contrast to k-means--, MORe++ gains an advantage, as the dataset is divided into more subsets (clusters found by k-Means) on which the outlier detection is performed separately. Thus, the outliers become more obvious in those smaller subsets, which counteracts the before mentioned negative effects. That results in a relative improvement to k-means--, although the quality of outlier detection

[3] Best values for T: 0.4 for $k = \{8, 10\}$, 0.6 for $k = \{25, 50\}$, else $T = 0.2$.

decreases noteably even for MORe++ for datasets with a higher number of clusters than $k \geq 25$.

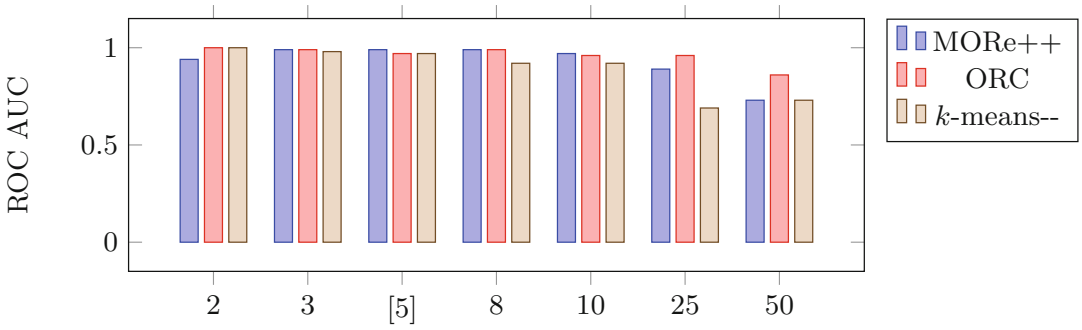


Fig. 8. ROC AUC Score for different numbers of clusters k

Experiments Regarding Noise Dimensions. Subspace clustering is based on the assumption, that with increasing number of dimensions more and more dimensions become "noise dimensions". According to that, we added noisy dimensions to our dataset, for which the results can be seen in Fig. 9[4]. With an increase of noise dimensions, distance measures in the full-dimensional space become more and more meaningless according to the curse of dimensionality, and the noise of some dimensions distorts the outlierness in other dimensions for algorithms using distances on the full-dimensional space. As MORe++ regards every dimension separately and counts the number of dimensions in which a point is an outlier, it is clearly more robust to additional noise dimension than its competitors.

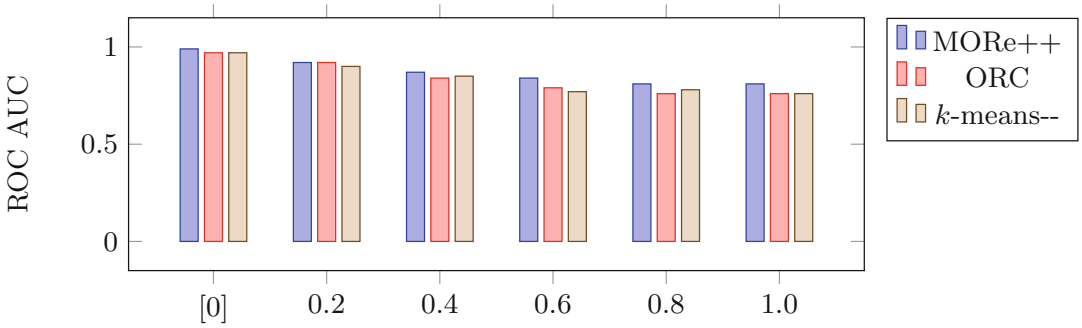


Fig. 9. ROC AUC Score for different number of noise dimensions

[4] Best ROC AUC values for ORC were achieved with $T = 0.5$ for $dim_n = 0.2$, $T = 0.7$ for $dim_n = 1.0$, and $T = 0.6$ else. For MORe++ $ost = 0.3$ delivered best results for $dim_n = \{0.8, 1.0\}$, else $ost = 0.2$.

Evaluation of Systematic Experiments. The biggest difference of results could be seen for high-dimensional data, where ORC was clearly outperformed by MORe++ and k-Means--. As many results seem to be similar or only slightly better than k-Means--, we want to emphasize the difference in runtime, where MORe++ only needs (including running the k-Means) $O(nkdi)$ and k-Means-- needs $O(n^{d^3})$. Further, MORe++ persuaded especially in the following points:

- For datasets of high dimensionality
- For datasets with clusters of higher variances
- For datasets with noise dimensions

4.2 Real World Datasets

We tested some real world datsets with different properties: sizes ranged between 148 and almost 95000, dimensionalities between 3 and 166, percentage of outliers between 0.4 and 7 and number of clusters between 1 and 5. Table 1 gives an overview over size, number (and percentage) of outliers, and number of clusters of the real world datasets which we used and which can be found in the ODDS library [25]. Table 2 gives the parameters chosen for all algorithms, where we used the parameter resulting in the best ROC AUC, resp. the ground truth value as number of outliers l for k-means--. For MORe++ and ORC we tested values between 0 and 1 in steps of 0.1. As in the previous section, we took the average of 100 executions of k-means-- due to its non-determinism. Figure 10 shows the results of the experiments: for Lympho, Shuttle, and Smtp MORe++ clearly outperforms both other algorithms. Glass is an interesting experiment, as ORC is the best performing algorithm here, followed by MORe++. For Musk, MORe++ achieved the best results, closely followed by k-means-- and ORC. So, even though MORe++ performed quite similar to k-means-- in most cases in the previous section, it seems to be more suitable for real world scenarios plus it is by far faster. ORC performed well on the Glass dataset, but cannot deal with very high number of dimensions as shown in Sect. 4.1. Thus, for outlier detection in high-dimensional datasets, MORe++ is preferable.

Table 1. Overview of real world datasets

Dataset	n	dim	outlier	k
Glass	214	9	9 (4.2%)	5
Lympho	148	18	6 (4.1%)	2
Musk	3062	166	97 (3.2%)	3
Shuttle	49097	9	3511 (7%)	1
Smtp	95156	3	2211 (0.4%)	1

Table 2. Chosen parameters for real world datasets

	MORe++ *ost*	ORC *T*	k-means– *l*
Glass	0.1	0.3	9
Lympho	0.4	0.7	6
Musk	0.3	0.7	97
Shuttle	0.3	0.3	3511
Smtp	0.4	0.5	2211

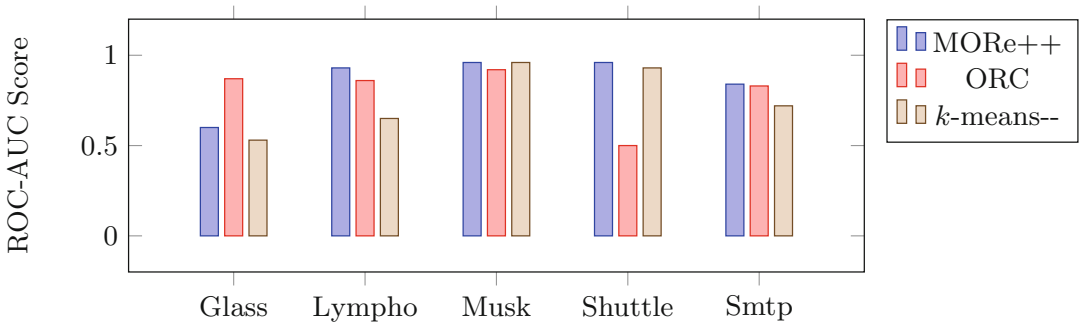


Fig. 10. ROC AUC for real world datasets

5 Conclusion

In conclusion we developed the outlier detection algorithm MORe++, which is based on histograms of one-dimensional projections of separately regarded k-Means clusters. By projecting onto single dimensions, we can circumvent some aspects of the curse of dimensionality: neither do we need a distance measure working in high-dimensional space nor is our runtime exponential in the number of dimensions. Users do not have to know the number of outliers beforehand and local outliers can easily be detected. The algorithm is easily parallelizable and easy to implement. A plethora of variations and improvements of k-Means could be used to further improve our already good results, and also using another algorithm than k-Means as foundation for the partitioning of the data could be tried.

Acknowledgement. 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. Ahmed, M., Mahmood, A.N.: A novel approach for outlier detection and clustering improvement. In: 2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA), pp. 577–582. IEEE (2013)
2. Arthur, D., Vassilvitskii, S.: k-means++: the advantages of careful seeding. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1027–1035. Society for Industrial and Applied Mathematics (2007)
3. Bahmani, B., Moseley, B., Vattani, A., Kumar, R., Vassilvitskii, S.: Scalable k-means++. Proc. VLDB Endow. **5**(7), 622–633 (2012)
4. Bellman, R.E.: Adaptive Control Processes: A Guided Tour, vol. 2045. Princeton University Press, Princeton (2015)
5. Bradley, P.S., Mangasarian, O.L., Street, W.N.: Clustering via concave minimization. In: Advances in Neural Information Processing Systems, pp. 368–374 (1997)
6. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: ACM Sigmod Record, vol. 29, pp. 93–104. ACM (2000)

7. Chawla, S., Gionis, A.: k-means-: a unified approach to clustering and outlier detection. In: Proceedings of the 2013 SIAM International Conference on Data Mining, pp. 189–197. SIAM (2013)
8. Fawcett, T.: An introduction to ROC analysis. Pattern Recognit. Lett. **27**(8), 861–874 (2006)
9. Freedman, D., Diaconis, P.: On the histogram as a density estimator: L 2 theory. Probab. Theory Relat. Fields **57**(4), 453–476 (1981)
10. Gan, G., Ng, M.K.P.: K-means clustering with outlier removal. Pattern Recognit. Lett. **90**, 8–14 (2017)
11. Gebski, M., Wong, R.K.: An efficient histogram method for outlier detection. In: Kotagiri, R., Krishna, P.R., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 176–187. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71703-4_17
12. Goldstein, M., Dengel, A.: Histogram-based outlier score (HBOS): a fast unsupervised anomaly detection algorithm. In: KI-2012: Poster and Demo Track, pp. 59–63 (2012)
13. Hautamäki, V., Cherednichenko, S., Kärkkäinen, I., Kinnunen, T., Fränti, P.: Improving K-means by outlier removal. In: Kalviainen, H., Parkkinen, J., Kaarna, A. (eds.) SCIA 2005. LNCS, vol. 3540, pp. 978–987. Springer, Heidelberg (2005). https://doi.org/10.1007/11499145_99
14. Hawkins, D.M.: Identification of Outliers, vol. 11. Springer, Dordrecht (1980). https://doi.org/10.1007/978-94-015-3994-4
15. Hyndman, R.J.: The problem with Sturges rule for constructing histograms (1995)
16. Jiang, M.F., Tseng, S.S., Su, C.M.: Two-phase clustering process for outliers detection. Pattern Recognit. Lett. **22**(6–7), 691–700 (2001)
17. Jiang, S., An, Q.: Clustering-based outlier detection method. In: 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery, vol. 2, pp. 429–433. IEEE (2008)
18. Kriegel, H.P., Zimek, A., et al.: Angle-based outlier detection in high-dimensional data. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 444–452. ACM (2008)
19. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)
20. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Oakland, CA, USA, vol. 1, pp. 281–297 (1967)
21. Mautz, D., Ye, W., Plant, C., Böhm, C.: Towards an optimal subspace for k-means. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 365–373. ACM (2017)
22. Mautz, D., Ye, W., Plant, C., Böhm, C.: Discovering non-redundant k-means clusterings in optimal subspaces. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1973–1982. ACM (2018)
23. Müller, E., Assent, I., Iglesias, P., Mülle, Y., Böhm, K.: Outlier ranking via subspace analysis in multiple views of the data. In: 2012 IEEE 12th International Conference on Data Mining, pp. 529–538. IEEE (2012)
24. Papadimitriou, S., Kitagawa, H., Gibbons, P.B., Faloutsos, C.: Loci: fast outlier detection using the local correlation integral. In: Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405), pp. 315–326. IEEE (2003)
25. Rayana, S.: ODDS library (2016). http://odds.cs.stonybrook.edu

26. Scott, D.W.: On optimal and data-based histograms. Biometrika **66**(3), 605–610 (1979)
27. Seidl, T., Müller, E., Assent, I., Steinhausen, U.: Outlier detection and ranking based on subspace clustering. In: Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2009)
28. Whang, J.J., Dhillon, I.S., Gleich, D.F.: Non-exhaustive, overlapping k-means. In: Proceedings of the 2015 SIAM International Conference on Data Mining, pp. 936–944. SIAM (2015)
29. Zhou, Y., Yu, H., Cai, X.: A novel k-means algorithm for clustering and outlier detection. In: 2009 Second International Conference on Future Information Technology and Management Engineering, pp. 476–480. IEEE (2009)
30. Zimek, A., Schubert, E., Kriegel, H.P.: A survey on unsupervised outlier detection in high-dimensional numerical data. Stat. Anal. Data Min.: ASA Data Sci. J. **5**(5), 363–387 (2012)

4.2 "Angle-Based Clustering"

Publication: Anna Beer, Dominik Seeholzer, Nadine-Sarah Schüler, and Thomas Seidl. "Angle-Based Clustering". In: *Proceedings of the International Conference on Similarity Search and Applications*. Cham: Springer International Publishing, 2020, pp. 312–320. DOI: 10.1007/978-3-030-60936-8_24

Statement of Originality: I developed the concept of the algorithm and guided Dominik Seeholzer's Bachelor's Thesis after consultation with Thomas Seidl. In context of the thesis, Dominik Seeholzer implemented the algorithm and several similar versions and performed the experiments that I designed. Nadine Sarah Schüler helped writing the paper.

Angle-Based Clustering

Anna Beer( ), Dominik Seeholzer, Nadine-Sarah Schüler, and Thomas Seidl

Ludwig-Maximilians-Universität München, Munich, Germany
{beer,schueler,seidl}@dbs.ifi.lmu.de,
d.seeholzer@campus.lmu.de

Abstract. The amount of data increases steadily, and yet most clustering algorithms perform complex computations for every single data point. Furthermore, Euclidean distance which is used for most of the clustering algorithms is often not the best choice for datasets with arbitrarily shaped clusters or such with high dimensionality. Based on ABOD, we introduce ABC, the first angle-based clustering method. The algorithm first identifies a small part of the data as border points of clusters based on the angle between their neighbors. Those few border points can, with some adjustments, be clustered with well-known clustering algorithms like hierarchical clustering with single linkage or DBSCAN. Residual points can quickly and easily be assigned to the cluster of their nearest border point, so the overall runtime is heavily reduced while the results improve or remain similar.

1 Introduction

If there are clusters in a dataset, most of the points lie rather in the middle of a cluster than at its border, and if the clusters of the border points are known, the assignment of inner points is easy and fast using a simple 1NN classification. To identify border points we suggest an angle based approach inspired by *Angle-Based Outlier Detection* (ABOD) [4], which is robust even for higher dimensionalities.

Our new clustering method ABC (Angle-Based Clustering), consists of three steps: First, by assessing the angles between difference vectors of points to their kNN, we can reliably identify points located at the boundaries of clusters. Secondly, we apply existing clustering techniques on those border points only, which allows us to reduce the number of points to be clustered severely. Finally, inner points are assigned to the same cluster as their nearest border point. As clustering has a higher complexity than the angle-based border point extraction as well as inner point assignment, the total runtime is dramatically reduced by clustering only a small fraction of all data points.

Our main contributions are as follows:

- Based on angles between a point and its kNN we detect the border points bounding clusters
- We apply adapted versions of DBSCAN and Hierarchical Single-Linkage Clustering on the border points

S. Satoh et al. (Eds.): SISAP 2020, LNCS 12440, pp. 312–320, 2020.
https://doi.org/10.1007/978-3-030-60936-8_24

– In experiments we show not only the speedup of algorithms using only the border points, but also the improvement of results regarding quality

2 Related Work

ABOD [4] was the first algorithm to use angles for outlier detection by regarding the variance of angles between the difference vectors of a point to all pairs of other points. Several works extended it regarding, e.g., acceleration [8], streams [13], and or stability [5]. ABSAD [14] uses angles between points and axis-parallel lines for an angle-based subspace anomaly detection method.

We could only find one work which uses angles in the field of clustering: SCUBI [11] combines classical clustering with detecting boundary information using angles to create a highly scalable clustering scheme. In contrast to our approach, they use the angles only for an approximation to an intrinsically density-based boundary extraction. Furthermore, we consider the previously calculated angles also for the clustering step by improving the distance function.

There are diverse approaches to identify border points: density based [11,12], hull based [7], and graph based [6] . Nevertheless, they lead to problems for higher dimensionalities, either regarding meaningfulness, or complexity.

3 Mathematical Background

Angles Between Data Points. Angles in a finite-dimensional real Euclidean vector space $\mathbb{V}^{\mathbb{R}}(\simeq \mathbb{R}^d, d \in \mathbb{N}, d \geq 2)$ are defined between any pair of vectors $A, B \in \mathbb{V}^{\mathbb{R}}$ with:

$$cos\Theta(A,B) = \frac{(A,B)_R}{|A|\,|B|}, \tag{1}$$

where $(A,B)_R = \sum_{k=1}^{d} A_k B_k$ is the scalar product between the two vectors and $|A| = \sqrt{(A,A)_R}$ [9]. For the resulting (real) angle $\Theta(A,B)$ the following holds true: $0 \leq \Theta \leq \pi$.

Directional Angle and Enclosing Angle. Figure 1 (left) shows the minimal angle for a point X between two difference vectors to its neighboring points which "encloses" all other neighboring points (green shape). We call it the *enclosing angle* Θ_{enc} of a point. One way to calculate the *enclosing angle* in two dimensions requires to calculate the directional angle between two vectors. In a 2d vector space with vectors $\overrightarrow{XY} = (u_1, u_2)$, $\overrightarrow{XZ} = (v_1, v_2) \in \mathbb{V}_2$, the counter-clockwise directional angle from $\overrightarrow{XY}$ to $\overrightarrow{XZ}$ is $\Theta_{YZ}(X) = atan2(u_2, u_1) - atan2(v_2, v_1)$. If the resulting Θ_{dir} is negative, we add 2π to receive only positive values between 0 and 2π. Figure 1 shows an example directional angle Θ_{dir}. Note, that if the directional angle is less than π, it will be equal to the cosine angle.

To obtain the enclosing angle of a point X, we calculate the directional angle between difference vectors to all pairs of neighbors and differentiate two cases:

First, if $\exists Y \in kNN(X) : \forall Z \in kNN(X) : \Theta_{YZ} \geq \pi$ as illustrated in Fig. 1 (middle), the enclosing angle can be calculated as $2\pi - min(\{\Theta_{YZ} | Y, Z \in kNN(X)\})$. Otherwise, the enclosing angle can be calculated as $2\pi - max(\{min(\{\Theta_{YZ} | Z \in kNN(X)\}) | Y \in kNN(X)\})$, as shown in Fig. 1 (right). We can use the concept behind *enclosing angle* to characterize the relative position of neighboring points. Points in the center of a cluster tend to have much larger enclosing angles.

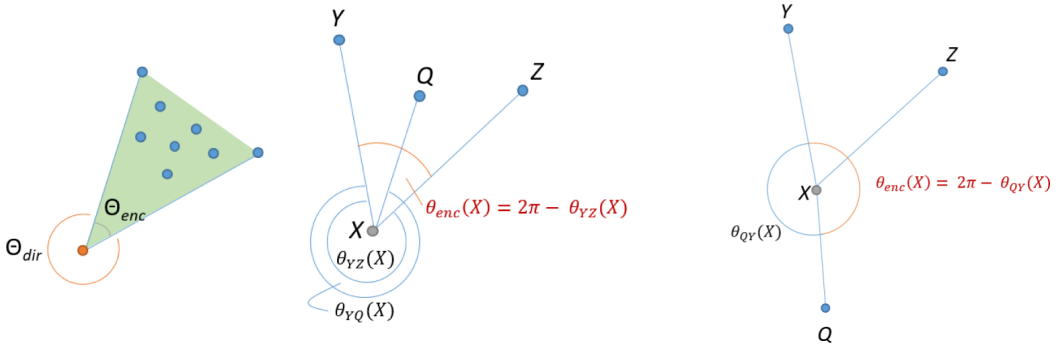


Fig. 1. Left: Enclosing Angle θ_{enc} and counter-clockwise Directional Angle θ_{dir}. Middle and Right: Example calculation of the enclosing angle θ_{enc}. (Color figure online)

4 ABC: Angle-Based Clustering Approach

ABC consists of three steps: First we calculate an angle-based border degree, see Sect. 4.1. The top β points with the highest border degree are the border points. Secondly, we cluster the border points using either an adapted DBSCAN or Hierarchical-Single Linkage Clustering, see Sect. 4.2. Finally, inner non-border points are assigned to cluster of their nearest border point. With a k-d tree this can be done in $O(n \log n)$.

4.1 Border Point Detection Based on Enclosing Angles

Because the nearest neighbors are all located in a similar direction for border points, their enclosing angle (see Sect. 3) tends to be much smaller compared to inner points. As we work with higher dimensionalities we use the following approximation: The *enclosing angle based border degree* is calculated as the maximum of all angles between the vector formed by query point to the kNN-mean and the vector from query point to one of the neighbors. Figure 2 (left) shows a simplified 2d example. The approximated enclosing angle θ_{enc} for border points tends to be much smaller than for inner points. The green shape encompasses the enclosed points.

The complete enclosing angle based border point extraction process proceeds as follows: For each point the kNN, the average distance to them, and the enclosing angles are calculated. For the *direction* of a border point, we use the vector

from the query point to the kNN-mean. Border points are then sorted by border degree and the $\beta \cdot n$ points with the highest border degree are returned as the *Boundary*. Figure 2 (middle and right) shows an example on a two dimensional dataset, where darker points imply a higher border degree.

Parameter Analysis. Small values for k can lead to inner points being falsely identified as border points, high values can lead to inter-cluster border points not being recognized as such, i.e., we only find the global boundary of all clusters. For datasets with many close clusters a small k should be preferred, while far separated clusters yield better results with a larger k.

The parameter β determines the separation threshold between border and inner points. Too high values yield more border points leading to a longer execution time of the subsequent clustering step. Too small values will fail to correctly identify enough cluster boundaries. In general, we have found values for β between 5–20% to yield optimal results.

4.2 ABC-DBSCAN/ABC-Hierarchical-SL

To cluster the boundary points we can use an adaption of DBSCAN [1] in which we regard also the *direction* of each border point to its neighbors. As border points that lie close to each other but have opposing directions are unlikely to belong to the same cluster, we use the following new the distance function instead of the Euclidean:

Definition 1. *Direction-Angle modified Distance Function*
Given two border points $A, B \in \mathcal{D}$ and their respective direction vectors $\boldsymbol{a}, \boldsymbol{b}$ as well as the Euclidean distance $d(A,B)_{eucl}$ between the points and the angle $\Theta(A,B)$ between their direction vectors. Then, given a direction-angle modifier σ_{mod} , the ***direction-angle modified distance*** *$d(A,B)_{mod}$ is calculated as:*

$$d(A,B)_{mod} = d(A,B)_{eucl} * (1 + (\frac{\sigma_{mod} - 1}{\pi}) * cos\Theta(A,B)) \tag{2}$$

A larger angle between the direction vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ results in a larger modified distance, where σ_{mod} controls the maximum. A higher σ_{mod} leads to more influence of direction-angle similarity compared to the Euclidean distance. When $\sigma_{mod} = 1$, then $d(A,B)_{mod} = d(A,B)_{eucl}$. A value of $\sigma_{mod} < 1$ increases the distance between points with different angle. Note, that this distance function does not represent a metric, since the triangle inequality does not always hold.

Another well suited approach to cluster border points is hierarchical agglomerative clustering using single linkage (Hierarchical-SL) [3]. Again with a complexity of $O(n^2)$, potential time savings using Angle-Based border point clustering are high. Also here we use the modified distance as described in Definition 1.

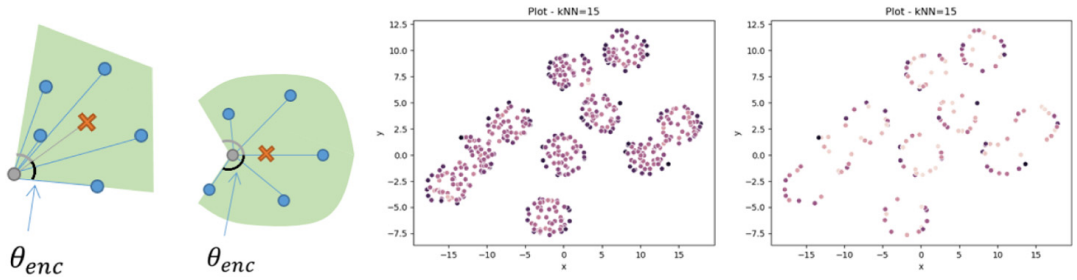


Fig. 2. Left: Approximated Enclosing Angles for border point and inner point. The red cross marks the mean of the blue kNN of the regarded gray point. Right: Border degree and selected border points ($k = 15$, $\beta = 0.2$). (Color figure online)

Complexity Analysis. Calculating the border degree requires an kNN query with complexity $O(n \log n)$ using a k-d tree [10]. The border degree calculation itself has complexity $O(n * k)$, as an angle between each nearest neighbor of each point and the mean of all its kNN is calculated. The sorting and selection of border points is $O(n \log n)$. In total, we get $O(n \log n + nk)$. As k is typically very small ($k \leq \log n$) the overall complexity is then $O(n \log n)$.

5 Experiments and Results

The following Sect. 5.1 covers results of experiments analyzing the runtime of algorithms. The quality on different kinds of datasets, both synthetic and real, are compared in Sect. 5.2 based on the Adjusted Rand Index (ARI).

5.1 Runtime

As ABC only requires to cluster a small fraction of all data points it is highly scalable and well suited for big datasets. Figure 3 (left) summarizes the experiments on how long each of the main three steps (border degree calculation, border point clustering and inner point assignment) take for an increasing number of points. As clustering is the most time consuming task with growing number of observations, reducing the amount of points having to be clustered significantly saves time.

As seen in Fig. 3 (right), ABC-DBSCAN outperforms the naive implementation of DBSCAN with time complexity $O(n^2)$. Even with the use of optimized index structures, the complexity of DBSCAN cannot be reduced below $O(n^{4/3})$ for higher dimensional data [2]. Thus, for large enough datasets, the ABC version with $O(n \log n)$ outperforms even optimized variants of DBSCAN.

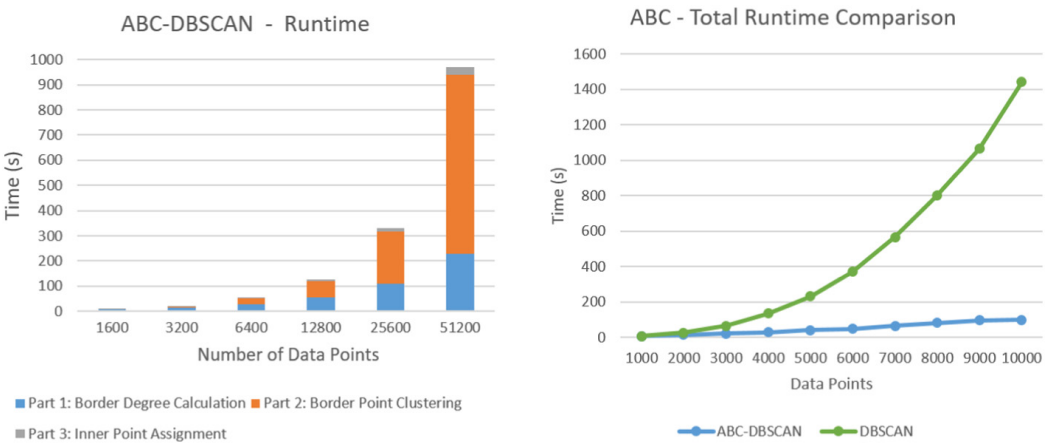


Fig. 3. Left: ABC-DBSCAN components runtime with $d = 5$, $\beta = 0.2$, $k = 10$. Right: Total runtime of DBSCAN and ABC-DBSCAN.

5.2 Quality

Datasets. First, we compare the quality of results on synthetic Gaussian data while modifying either cluster count, dimension count or standard deviation (the last one was left out due to the lack of space, even though ABC constantly outperformed the competitors slightly). The default dataset consists of $n = 1000$ data points, $c = 5$ clusters, $d = 5$ dimensions and a standard deviation $\sigma = 0.1$. Then, we test the algorithms on synthetic complex shaped data sets with and without noise. Finally, we investigate how they perform on real data sets.

Algorithms. We compare **ABC-DBSCAN** to the classic DBSCAN. Additionally, we compare it to **ABC-SCUBI-DBSCAN**, for which we adapt the idea of [11] and exclude a point from the DBSCAN ε-range if its angle is greater than $\pi/2$ (instead of our combined distance measure), but still use our border-degree measurement. Then, we compare the **ABC-Hierarchical-SL** approach to the classic Hierarchical-SL algorithm.

For ABC-DBSCAN and DBSCAN the same range of parameters is tested and the best result is kept. ABC-Hierarchical-SL and Hierarchical-SL get the correct amount of clusters given as the maximum cluster parameter. For the border point calculation, we used parameters $\beta = 0.3$ and $k = 15$. For the direction-angle modifier for ABC-DBSCAN and ABC-Hierarchical, we tested values $\sigma_{mod} \in \{0.1, 0.2, 0.3, 0.5, 1, 2, 5\}$ for different weightings of the angle compared to distance and kept the best result.

5.3 Synthetic Gaussian Distributed Data

Based on the dataset described above we varied the number of clusters c from 1 to 500, as shown in Fig. 4 (top). ABC-Hierarchical-SL outperforms the classical Hierarchical-SL, especially for higher c, where the latter only performs poorly. For the DBSCAN versions, the overall performance decreases with increasing c, but the ABC versions yield constantly better results than the original DBSCAN.

For varying dimensionalities $d \in [2, 1000]$. ABC-Hierarchical-SL as well as Hierarchical-SL converge towards an ARI of 1. The ABC version works slightly better even for small d. All DBSCAN based algorithms suffer from the *"curse of dimensionality"*, dropping to an ARI of 0 for high $d \geq 70$. ABC-DBSCAN still performs well for a much higher d than the classic DBSCAN.

5.4 Benchmark Datasets

To evaluate more complex cluster shapes, we also tested our algorithms with the *Complex9* dataset and its noisy version *Cluto-t7*. Both contain nine different types of clusters including blobs, moons and anisotropically distributed shapes. As depicted in Fig. 4 (bottom), ABC-Hierarchical-SL achieves near perfect results and outperforms the original, since the single link effect connecting two different Hierarchical-SL clusters is prevented by using our adapted distance measure. ABC-DBSCAN and ABC-SCUBI-DBSCAN are slightly outperformed by the original DBSCAN. In such cases, ABC could still be chosen with a trade-off between a huge improvement of the runtime and a rather small decrease of the quality. Results for the noisy dataset *Cluto-t7* show similar behavior, except for a significant improvement from ABC-Hierarchical-SL over the original.

Finally, we applied all algorithms on the real datasets *Iris*, *Seed*, and *Ecoli* from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml). In summary, the ABC versions performed at least comparatively well, in many cases even better than the original, as shown exemplarily in Fig. 4.

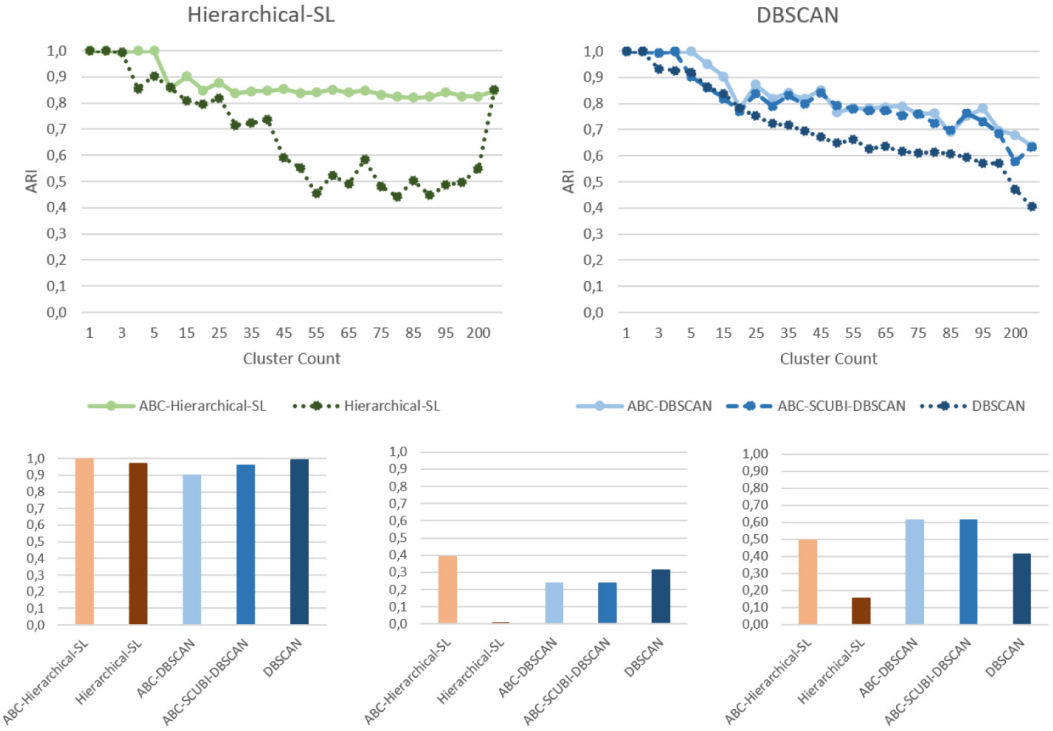


Fig. 4. Top: ARI of synthetic Gaussian distributed data for increasing number of clusters Bottom: ARI of Complex9 (left), Noisy Cluto-t7 (middle) and Ecoli (right)

6 Conclusion

We developed ABC, an angle-based clustering method, which is based on common clustering algorithms like DBSCAN and hierarchical Single-Link clustering, but many times faster as only the few cluster border points, have to be clustered by the respective algorithm. The points lying in the middle of a cluster can easily be assigned to the cluster of their nearest border point. We developed a method to detect those border points based on the angle enclosing their nearest neighbors, which is significantly smaller for points bordering a cluster than for those lying in the inner part. Experiments show that the results are similar or slightly better than those of the original algorithms on synthetic as well as on real world data.

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. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, vol. 96, pp. 226–231 (1996)
2. Gan, J., Tao, Y.: DBSCAN revisited: mis-claim, un-fixability, and approximation. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 519–530. ACM (2015)
3. Johnson, S.C.: Hierarchical clustering schemes. Psychometrika **32**, 241–254 (1967). https://doi.org/10.1007/BF02289588
4. Kriegel, H.P., Zimek, A., et al.: Angle-based outlier detection in high-dimensional data. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 444–452. ACM (2008)
5. Li, X., Lv, J.C., Cheng, D.: Angle-based outlier detection algorithm with more stable relationships. In: Handa, H., Ishibuchi, H., Ong, Y.-S., Tan, K.C. (eds.) Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Volume 1. PALO, vol. 1, pp. 433–446. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-13359-1_34
6. Liu, D., Nosovskiy, G.V., Sourina, O.: Effective clustering and boundary detection algorithm based on Delaunay triangulation. Pattern Recogn. Lett. **29**(9), 1261–1273 (2008)
7. Moreira, A., Santos, M.Y.: Concave hull: a k-nearest neighbours approach for the computation of the region occupied by a set of points (2007)
8. Pham, N., Pagh, R.: A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 877–885. ACM (2012)
9. Scharnhorst, K.: Angles in complex vector spaces. Acta Applicandae Mathematica **69**(1), 95–103 (2001). https://doi.org/10.1023/A:1012692601098
10. Sproull, R.F.: Refinements to nearest-neighbor searching ink-dimensional trees. Algorithmica **6**(1–6), 579–589 (1991). https://doi.org/10.1007/BF01759061

11. Tong, Q., Li, X., Yuan, B.: A highly scalable clustering scheme using boundary information. Pattern Recogn. Lett. **89**, 1–7 (2017)
12. Xia, C., Hsu, W., Lee, M.L., Ooi, B.C.: Border: efficient computation of boundary points. IEEE Trans. Knowl. Data Eng. **18**(3), 289–303 (2006)
13. Ye, H., Kitagawa, H., Xiao, J.: Continuous angle-based outlier detection on high-dimensional data streams. In: Proceedings of the 19th International Database Engineering & Applications Symposium, pp. 162–167. ACM (2015)
14. Zhang, L., Lin, J., Karim, R.: An angle-based subspace anomaly detection approach to high-dimensional data: with an application to industrial fault detection. Reliab. Eng. Syst. Safety **142**, 482–497 (2015)

Chapter 5

Between Clustering and Ordering

This chapter includes the following publications:

1. Anna Beer, Valentin Hartmann, and Thomas Seidl. "Orderings of Data - More Than a Tripping Hazard: Visionary". In: *Proceedings of the 32nd International Conference on Scientific and Statistical Database Management (SSDBM)*. Association for Computing Machinery, 2020, 17:1–17:4. DOI: 10.1145/3400903.3400911

2. Anna Beer and Thomas Seidl. "Graph Ordering and Clustering: A Circular Approach". In: *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*. SSDBM '19. Association for Computing Machinery, 2019, 185—188. DOI: 10.1145/3335783.3335802

5.1 "Orderings of Data - More Than a Tripping Hazard: Visionary"

Publication: Anna Beer, Valentin Hartmann, and Thomas Seidl. "Orderings of Data - More Than a Tripping Hazard: Visionary". In: *Proceedings of the 32nd International Conference on Scientific and Statistical Database Management (SSDBM)*. Association for Computing Machinery, 2020, 17:1–17:4. DOI: 10.1145/3400903.3400911

Statement of Originality: The idea originated from a discussion between me and Valentin Hartmann. Research and writing was in collaboration with Valentin Hartmann under my lead. Thomas Seidl gave some inspiration of potentially missing topics.

Orderings of Data - More Than a Tripping Hazard

Visionary

Anna Beer
LMU Munich
Munich, Germany
beer@dbs.ifi.lmu.de

Valentin Hartmann
EPFL
Lausanne, Switzerland
valentin.hartmann@epfl.ch

Thomas Seidl
LMU Munich
Munich, Germany
seidl@dbs.ifi.lmu.de

ABSTRACT

As data processing techniques get more and more sophisticated every day, many of us researchers often get lost in the details and subtleties of the algorithms we are developing and far too easily seem to forget to look also at the very first steps of every algorithm: the input of the data. Since there are plenty of library functions for this task, we indeed do not have to think about this part of the pipeline anymore. But maybe we should. All data is stored and loaded into a program in *some* order. In this vision paper we study how ignoring this order can (1) lead to performance issues and (2) make research results unreproducible. We furthermore examine desirable properties of a data ordering and why current approaches are often not suited to tackle the two mentioned problems.

CCS CONCEPTS

• **Computing methodologies** → *Ontology engineering*; • **General and reference** → *Measurement*;

KEYWORDS

Visionary, Ordering, Data Input, Uniqueness, Reproducibility

1 INTRODUCTION

Data is everywhere. We live in the age of data and most readers of this vision paper work with data every day. Data comes in all shapes and sizes, and humans try to gain knowledge from it. We apply classification, clustering, outlier detection, and many others algorithms to it and develop new tools every day. The questions we try to answer and the approaches we come up with are as diverse as the data itself. But all algorithms we have developed so far have one thing in common that rarely gets the attention it should. Discussion about algorithms is so centered around results and computational performance that the very beginning of every algorithm is quite often simply forgotten: The fundamental step of storing the data and presenting it to the algorithm, which is often not made by the researcher using the data, but by the one who collected the data, who wrote the preprocessing function or the function to load the data into main memory.

SSDBM 2020, July 7–9, 2020, Vienna, Austria

ACM ISBN 978-1-4503-8814-6/20/07.
https://doi.org/10.1145/3400903.3400911

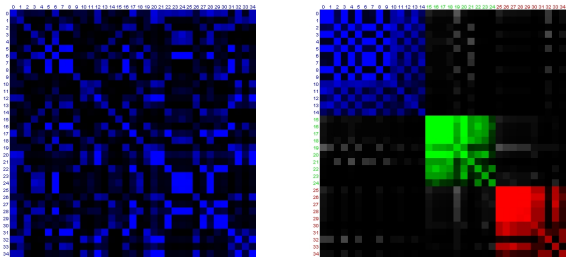


Figure 1: Adjacency matrices of a synthetic graph with differently ordered nodes.

However, this step is crucial not only for the performance, but also for the result of an algorithm, as Figure 3 illustrates. That phenomenon already occurs with simple algorithms and algorithm primitives like, e.g., k-NN-based algorithms. Here it can happen that several points have the same distance to a query point P and that the set of the k nearest neighbors depends on the tie-break rule. In this case the scikit implementation [4] still returns exactly k points, and bases the decision which points to discard on the ordering of the data points. Similar problems can occur in many other popular algorithms, too. In the field of clustering alone it encompasses algorithms like DBSCAN [13], k-Means [22], or agglomerative hierarchical clustering [17].

Randomly instead of deterministically deciding how to organize the data will also most likely not result in an ordering with optimal performance for the task at hand, since, based on this decision, datasets can have vastly different structural properties; see Figures 1 and 2. Thus, we want to encourage research on orderings of data, in more detail, we want to encourage finding solutions to several different aspects connected to orderings of data:

(1) A unique ordering for tabular data, which is especially interesting for high-dimensional data, as well as a unique ordering for the nodes of a graph
(2) A measure for the orderliness of data. That can either depend on how well suited an ordering is for the subsequent task, or how "close" it is to a unique ordering.

Section 2 discusses these issues in more detail. In Section 3 we develop approaches how to overcome them. We extract the need of an internal quality measure for orders and define the essential conditions of such a measure. In Section 4 we give an overview over the first steps that have already been made in this direction. Section 5 puts data orderings in context to several other research areas. Section 6 concludes this vision paper with a summary of the questions we rate as important and promising for further investigation.

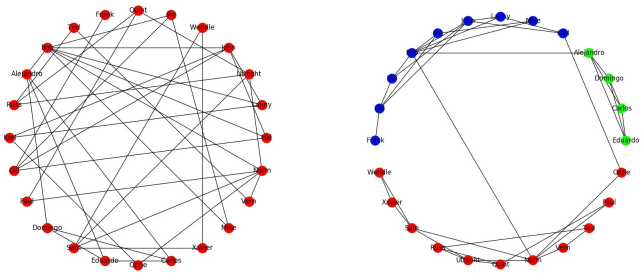


Figure 2: Graph of the social network "Strike" [11, 23]: Randomly ordered in a circle on the left, ordered by their similarity according to [8] with colors matching the ground truth on the right.

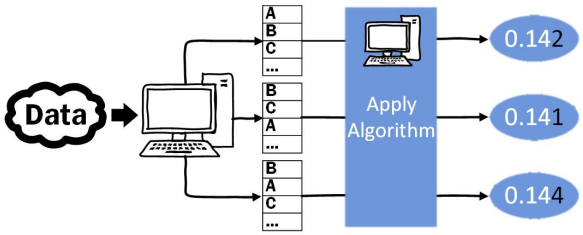


Figure 3: Different orderings of the same data can lead to different results when the same algorithm is applied to it.

2 THE PROBLEM

Not considering the ordering of the data objects one feeds to an algorithm has the major drawback of potentially making the output as well as the runtime of the algorithm non-deterministic, as Figure 3 illustrates. In the first part of this section we focus on the reproducibility in regards of research results and in the second part we address the variance of runtimes, as consistent runtimes are of high importance in various domains as, e.g., autonomous driving and other real-time applications.

Reproducibility. In the past years the problem of reproducibility of research results in computer science has received increasing attention with top-tier conferences awarding prices for reproducibility [3] or hosting reproducibility workshops [6], and universities trying to reproduce results at a large scale [5]. One of the reasons for these efforts is that especially in machine learning, theory is lacking behind the state-of-the-art, so validity and performance of many methods is only proven via experiments.

Randomness in algorithms can have a huge influence not only on the running time, but also on the output of an algorithm [14, Fig. 5] (consider, e.g., the case of training a non-convex machine learning model with several local minima). Addressing this problem by fixing a random seed alone is not sufficient; one also needs to fix the data ordering.

Performance. Physically organizing the data in a way that best fits the access patterns is a problem with a long history in database research [19, 27, 29]. The common idea is to exploit the locality of reference present in many applications [12]. In the case of data ordering, the spatial locality of reference is of particular interest. A simple example is breadth first search, where the nodes of a graph that are connected to each other are accessed shortly after each other, as well as nodes having the same depth with respect to the root node. To achieve optimal performance, such elements that are frequently accessed together or right after each other should also be close to each other on the physical storage. More concretely, it should be more efficient to access them sequentially — potentially reading a small amount of unwanted data in addition — than via random access. Sequential access is not only far more efficient for hard disks, but also for SSDs [9], eMMCs [20] and DRAM [16, Page 62]. Especially for column-store DBS, ordering of data plays a central role for the performance, like, e.g., when using holistic indexing [26].

Mathematical Formulation. Let D be the set of objects of a certain class, e.g., graphs on n unlabeled vertices. Let R be the set of representation of these objects of a certain type, e.g., the set of all $n \times n$ adjacency matrices. Then there exists an injective mapping from R to D (each adjacency matrix corresponds to exactly one graph). The inverse mapping, however, in many cases is not well-defined, since there can be many different representations of the same mathematical object (consider an adjacency matrix that only differs from another one in the order of the vertices).

3 HOW TO APPROACH THE PROBLEM

To remove the non-determinism inherent in all algorithms that work with non-unique representations of data, one has to define an injective mapping from the set of objects D to the set of representations R. Since the representations that algorithms work with are typically bit strings that list the elements that D consists of in a certain order, this means defining a unique ordering.

There are "better" or "worse" orderings. Depending on the application, different properties may be desirable. Two examples are:

- Points or nodes that are similar to each other should occur close together. Orderings of this type can, for example, efficiently support classic clustering.
- Points or nodes with similar roles, like, e.g., points in the middle of a cluster or hubs in a graph, should occur close together. Orders like these could significantly improve speed and support pruning.

To work towards fulfilling such a property with an ordering, one needs to quantify it. So on the one hand we need unique representations, and on the other hand we need quality measures for orderings.

Even though many internal as well as external quality measurements for clusters have been developed (already in 1981, [24] performed a study of *thirty* internal measures for cluster analysis), so far there are vanishingly few for orderings. And yet quality measurements for ordering are not only closely related to those for clusterings, but could also support clustering. A good clustering implies a partial order on nodes, but the nodes inside a cluster stay orderless. Vice versa, an ordering of nodes can easily be used to cluster them, since similar nodes are already close to each other.

Thus, nodes which are ordered "well" regarding such a quality measurement for ordering build a better foundation for clustering than those which are ordered "poorly", as, e.g., shown in [8].

Requirements for an Internal Measure of Orderliness. Inspired by [28], we formulate requirements :

(1) Scale invariance. There shall be no bias against larger or smaller amounts of data.
(2) The measure should be absolute, i.e., it should be possible to compare the orderliness of different datasets.
(3) Invariance with respect to language (e.g., regarding the attribute description of a dataset).

When using an ordering as way of improving the performance of a downstream task, one also has to take into account the time the ordering itself takes, see Sec. 5. The importance of this, however, rapidly decreases when the dataset is not used only once, but either queried many times or distributed in its ordered form, so that multiple runs of downstream algorithms can profit from the ordering. In these cases, orderings may be automatically obtained by analyzing the access patterns of the downstream algorithms on the dataset at hand, and letting a machine learning model derive a good data organization strategy automatically. The advantage of such dataset-specific approaches has recently been demonstrated for index structures [21].

4 EXISTING SOLUTIONS

There are already some approaches to data ordering, which we will present here. They all have the drawback that they either do not maximize a quality measure at all or that this quality measure is very general and not aimed at a specific goal or adapted to a specific algorithm. We also note that there are algorithms which order data in a preprocessing step, thus a certain ordering is obviously important for some use-cases. On the other hand an ordering suitable for many different algorithms is desirable.

Order of Input. The most simple method is to use the order in which the data is given. Of course this order is typically not optimal performance-wise, but at least it is unique — if one ensures that always the same data file is used. But often there are many different versions of the same data set available, especially of popular ones. Take for example the graph Zachary's Karate Club [31], which can be obtained from [1], [2] and [7], which all use a different representation. Also, for the same file different representations in main memory can result depending on how it is read.

4.1 Ordering of Graphs

Order by degree. A straight-forward method many frameworks use is to order the nodes by their degree. Unfortunately, this is a non-deterministic order for most graphs, since it is very common that two or more nodes in a graph have the same degree.

Depth-first and Breadth-first. For depth-first and breadth-first graph traversal, a root node has to be determined, which already makes the resulting ordering non-deterministic. The order in which each node's neighbors are later visited is also non-deterministic.

Cuthill-McKee. The Cuthill-McKee algorithm [10] orders the nodes in a way such that the corresponding adjacency matrix is a band matrix with small bandwidth. For a given ordering π that assigns integers 1, ..., n to the n nodes v_i of a graph, the bandwidth is defined as $B = \max\{|\pi(v_i) - \pi(v_j)| : v_i v_j \in E\}$, i.e., the maximum distance of a non-zero entry to the diagonal in the adjacency matrix. However, the algorithm uses the node degree as one of the ordering criteria, which is not unique. Thus a tie-break rule must be used, which can either rely on randomness or on the order in which the data is presented to the algorithm.

Gscore. In [30] the Gscore, a measure for the closeness of nodes with many common predecessors, is introduced and an algorithm given to order the graph in such a way that the score is low. This is shown to decrease the number of cache misses of several graph algorithms. However, minimizing the Gscore in NP-hard, so one can only hope for coming close to the minimum. In addition, as the orderings introduced earlier, this ordering is agnostic of the algorithm processing the data further and will thus often not be able to reach the maximum possible performance improvement an ordering could give for a specific algorithm.

Edge Length Minimization. One of the most recent orderings is described in [8], where all nodes of a graph are arranged in a circle and the optimal ordering is the one with the shortest average length of edges. Nevertheless, it is neither deterministic nor absolute.

4.2 Ordering of Tabular Data

Even though ordering of non-structured data seems to raise fewer questions at first, we will see that there are multiple aspects worth targeting when ordering multidimensional data.

Alphanumerical Sorting. Probably the easiest approach is to order the data alphanumerically, beginning with the first column, in case of ties proceeding to the second column and so on. For this we need to decide on an ordering of the columns. They can of course also be sorted alphanumerically by the attribute name, but that means that the ordering of the columns, and thus the ordering of the whole data, depends on the language in which the attributes are named. The same can occur when sorting within the attributes if the data is, e.g., categorical.

Locality Preserving Orderings. There are several orderings that try to preserve local structures in the data (e.g., Z-order [25] or Hilbert curve [15]), which can improve access times. When working in higher dimensionalities, this produces an ordering in which points that are close in Euclidean distance stay closer to each other in the ordering than they would using, e.g., alphanumerical sorting. But similar to alphanumerical sorting, the resulting orderings depend on the ordering of the dimensions, and the performance improvements are not optimal, since the methods are general and not specialized to specific algorithms.

Multidimensional scaling. Multidimensional scaling algorithms map higher dimensional data to a lower dimensional space while trying to preserve the distances between the points. The most famous result in this domain is the Johnson-Lindenstrauss Lemma [18]. When the target dimension is one, those algorithms return an ordering. However, the fewer dimensions one uses, the worse the distance preservation guarantees get, and in the extreme case of one dimension basically no guarantees can be given.

5 RELATION TO OTHER RESEARCH AREAS

We want to point out some connections of ordering to other research areas, since an exchange between those fields could be very fruitful:

- Anytime algorithms. These are algorithms that can be stopped at any execution step before having finished and still return a feasible — though generally not optimal — solution. For anytime algorithms the order in which data is processed is crucial for the results obtained in early and intermediate stages. On one hand, a deeper understanding of the influence of different properties of the orderings on the workings of algorithms might be obtained by looking at the performance of anytime algorithms. On the other hand, anytime algorithms could better predict the quality of intermediate results if properties of the data ordering were known beforehand.
- Stream processing. In data streams the order of the elements is fixed and algorithms have to adapt to this. The solutions in this field might help to determine the robustness of algorithms to different orderings of their inputs.
- Databases. Alphanumerical sorting is frequently applied in databases, in addition to other techniques such as index structures, to allow for faster execution of certain queries. Here the main challenge is to determine for which columns the overhead of sorting or the creation of an index structure is justified. This is an interesting, somewhat orthogonal direction of research, whose results could very well be combined with new methods for the sorting itself.

6 CONCLUSION

In this vision paper we brought attention to the problem of data orderings. We illustrated its importance by noting that every data user is also a data reader, and reading requires an implicit or explicit ordering of data. We highlighted the often underestimated impact the ordering can have: it can hinder attempts to reproduce research results, but also offer an opportunity to improve the performance of algorithms. Based on these observations, we deduced the necessity of a quality measure for orderings. Such a measure can not only be used as an objective function when ordering data, but also help to make a decision as to whether it is necessary to reorder the data before further usage or not. We described different goals that one can aim at with an ordering, as well as the challenges when trying to achieve such a goal. We furthermore gave an overview over existing approaches and their drawbacks. We want to encourage research regarding the ordering of data, in particular in the challenging area of graphs. We also want to encourage research on the impact different data orderings have on the behaviour of algorithms.

ACKNOWLEDGMENTS

This work has been partially 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] 2001. *UCINET IV DatasetsUCINET IV Datasets.* Retrieved Apr. 12, 2019 from http://vlado.fmf.uni-lj.si/pub/networks/data/Ucinet/zachary.dat
[2] 2013. *Network data.* Retrieved Apr. 12, 2019 from http://www-personal.umich.edu/~mejn/netdata/
[3] 2017. *SIGMOD Most Reproducible Paper Award.* Retrieved Apr. 12, 2019 from https://sigmod.org/sigmod-awards/sigmod-most-reproducible-paper-award/
[4] 2018. *1.6. Nearest Neighbors - scikit-learn 0.20.1 documentation.* Retrieved Apr. 12, 2019 from https://scikit-learn.org/stable/modules/neighbors.html#unsupervised-nearest-neighbors
[5] 2018. *ICLR Reproducibility Challenge.* Retrieved Apr. 12, 2019 from https://www.cs.mcgill.ca/~jpineau/ICLR2019-ReproducibilityChallenge.html
[6] 2018. *Reproducibility in ML Workshop, ICML'18.* Retrieved Apr. 12, 2019 from https://sites.google.com/view/icml-reproducibility-workshop/home
[7] 2018. *ucidata-zachary | Miscellaneous Networks.* Retrieved Apr. 12, 2019 from http://networkrepository.com/ucidata_zachary.php
[8] Anna Beer and Thomas Seidl. 2019. Graph Ordering and Clustering: A Circular Approach. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management.* ACM, 185–188.
[9] Feng Chen, David A. Koufaty, and Xiaodong Zhang. 2009. Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems* (Seattle, WA, USA) *(SIGMETRICS '09).* ACM, New York, NY, USA, 181–192. https://doi.org/10.1145/1555349.1555371
[10] Elizabeth Cuthill and James McKee. 1969. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference.* ACM, 157–172.
[11] Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. 2011. *Exploratory social network analysis with Pajek.* Vol. 27. Cambridge University Press.
[12] Peter J Denning. 2006. The locality principle. In *Communication Networks And Computer Systems: A Tribute to Professor Erol Gelenbe.* World Scientific, 43–67.
[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96. 226–231.
[14] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2017. Deep Reinforcement Learning That Matters. *arXiv preprint arXiv:1709.06560* (2017).
[15] David Hilbert. 1891. Ueber die reellen Züge algebraischer Curven. *Math. Ann.* 38, 1 (1891), 115–138.
[16] Intel. 2012. *Intel 64 and IA-32 Architectures Optimization Reference Manual.* https://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf
[17] Stephen C Johnson. 1967. Hierarchical clustering schemes. *Psychometrika* 32, 3 (1967), 241–254.
[18] William B Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics* 26, 189-206 (1984), 1.
[19] J. P. Kearns and S. DeFazio. 1983. Locality of Reference in Hierarchical Database Systems. *IEEE Transactions on Software Engineering* SE-9, 2 (March 1983), 128–134. https://doi.org/10.1109/TSE.1983.236457
[20] Je-Min Kim and Jin-Soo Kim. 2012. *AndroBench: Benchmarking the Storage Performance of Android-Based Mobile Devices.* Springer Berlin Heidelberg, Berlin, Heidelberg, 667–674. https://doi.org/10.1007/978-3-642-27552-4_89
[21] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18).* ACM, New York, NY, USA, 489–504. https://doi.org/10.1145/3183713.3196909
[22] S. Lloyd. 1982. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theor.* 28, 2 (1982), 129–137. https://doi.org/10.1109/TIT.1982.1056489
[23] Judd H Michael. 1997. Labor dispute reconciliation in a forest products manufacturing facility. *Forest products journal* 47, 11/12 (1997), 41.
[24] Glenn W Milligan. 1981. A Monte Carlo study of thirty internal criterion measures for cluster analysis. *Psychometrika* 46, 2 (1981), 187–199.
[25] Guy M Morton. 1966. A computer oriented geodetic data base and a new technique in file sequencing. (1966).
[26] Eleni Petraki, Stratos Idreos, and Stefan Manegold. 2015. Holistic indexing in main-memory column-stores. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data.* ACM, 1153–1166.
[27] W. Rödiger, T. Mühlbauer, P. Unterbrunner, A. Reiser, A. Kemper, and T. Neumann. 2014. Locality-sensitive operators for parallel main-memory database clusters. In *2014 IEEE 30th International Conference on Data Engineering.* 592–603. https://doi.org/10.1109/ICDE.2014.6816684
[28] Twan Van Laarhoven and Elena Marchiori. 2014. Axioms for graph clustering quality functions. *The Journal of Machine Learning Research* 15, 1 (2014), 193–215.
[29] Paul Watson. 2005. Databases in Grid Applications: Locality and Distribution. In *Database: Enterprise, Skills and Innovation*, Mike Jackson, David Nelson, and Sue Stirk (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–16.
[30] Hao Wei, Jeffrey Xu Yu, Can Lu, and Xuemin Lin. 2016. Speedup Graph Processing by Graph Ordering. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16).* ACM, New York, NY, USA, 1813–1828. https://doi.org/10.1145/2882903.2915220
[31] Wayne W Zachary. 1977. An information flow model for conflict and fission in small groups. *Journal of anthropological research* 33, 4 (1977), 452–473.

5.2 "Graph Ordering and Clustering: A Circular Approach"

Publication: Anna Beer and Thomas Seidl. "Graph Ordering and Clustering: A Circular Approach". In: *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*. SSDBM '19. Association for Computing Machinery, 2019, 185—188. DOI: 10.1145/3335783.3335802

Statement of Originality: I developed the concept, implemented everything, performed all experiments and wrote the paper. I discussed the idea with Thomas Seidl, who pointed out some concerns that helped me to further improve the paper.

Graph Ordering and Clustering — A Circular Approach

Anna Beer
LMU Munich
Munich, Germany
beer@dbs.ifi.lmu.de

Thomas Seidl
LMU Munich
Munich, Germany
seidl@dbs.ifi.lmu.de

ABSTRACT

As the ordering of data, particularly of graphs, can influence the result of diverse Data Mining tasks performed on it heavily, we introduce the Circle Index, the first internal quality measurement for orderings of graphs. It is based on a circular arrangement of nodes, but takes in contrast to similar arrangements from the field of, e.g., visual analytics, the edge lengths in this arrangement into account. The minimization of the Circle Index leads to an arrangement which not only offers a simple way to cluster the data using a constrained MinCut in only linear time, but is also visually convincing. We developed the clustering algorithm CirClu, which implements this minimization and MinCut, and compared it with several established clustering algorithms achieving very good results. Simultaneously we compared the Circle Index with several internal quality measures for clusterings. We observed a strong coherence between the Circle Index and the matching of achieved clusterings to the respective ground truths in diverse real world datasets.

CCS CONCEPTS

• **General and reference** → **Evaluation**; • **Theory of computation** → **Unsupervised learning and clustering**; • **Human-centered computing** → Graph drawings;

KEYWORDS

Graph Ordering, Clustering, Graphs, Quality measure

ACM Reference Format:
Anna Beer and Thomas Seidl. 2019. Graph Ordering and Clustering — A Circular Approach. In *31st International Conference on Scientific and Statistical Database Management (SSDBM '19), July 23–25, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3335783.3335802

1 INTRODUCTION

Clustering, the art of partitioning data s.t. similar elements belong to the same group, is an established problem for Data Scientists. In contrast, orderings of data are still a neglected subject, even though they do not only serve as useful interim step for clustering algorithms, but also deliver useful information about data, which a pure partitioning cannot provide. The order of nodes in a graph can have severe influence on the clustering algorithm performed on it, and a good ordering enables solving the constrained MinCut problem in linear time, avoiding the exponential complexity of the unconstrained MinCut problem.

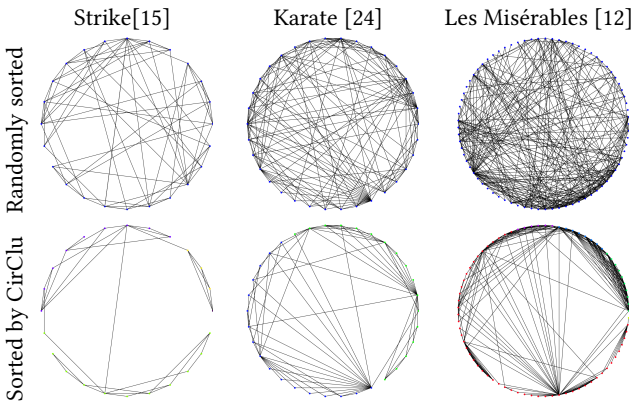


Figure 1: Three of our experiments, sorted randomly vs. sorted by CirClu aiming for a low Circle Index

We introduce a circular embedding of graphs onto the one-dimensional unit-sphere, which minimizes the edge lengths and leads to the Circle Index, our novel internal quality measure for orderings of nodes. Even though circular arrangements of graphs are quite common in visual analytics, none of them regards the edge lengths or uses the circular arrangement for further clustering or mathematical analysis of the respective network. In Fig. 1 we show the main effect of our optimization criterion: the confusing representations of the graphs above are randomly ordered. On the bottom they are ordered and colored according to our simple iterative clustering algorithm, CirClu (Circular Clustering), which we developed to underline the usefulness of our proposed ordering: it minimizes the Circle Index and performs a MinCut on the result, where there are only n possible cuts. Even though the clustering algorithm and also the Circle Index itself are work in progress and preliminary yet, they already deliver surprisingly good results.

We give an overview over related work w.r.t. graph orderings as well as internal quality measures and clustering in Sec. 2. In Sec. 3 we give the mathematical background for our work. In Sec. 4 we define and investigate the Circle Index, which is an internal robust quality measure for orderings of graphs. In Sec. 5 we introduce the clustering algorithm CirClu, which shows the expressiveness of the Circle Index based on several real world experiments presented in Sec. 6. Sec. 7 illustrates a multitude of future work enabled by the circular ordering. Our main contributions are as follows:

- We examine the importance of orderings of graphs
- We introduce an internal quality measure for orderings of graph, the Circle Index. It is, simplified, based on the average length of an edge if all nodes of a graph are arranged on a circle.

SSDBM '19, July 23–25, 2019, Santa Cruz, CA, USA

ACM ISBN 978-1-4503-6216-0/19/07…$15.00
https://doi.org/10.1145/3335783.3335802

- We suggest CirClu, a clustering algorithm minimizing this measure iteratively and performing a MinCut on the result, which is then of only linear complexity.

2 RELATED WORK

Even though circular arrangement is already used in several works, they are not comparable to ours: [17] does not work on graphs, [8] uses several small circles, [3] and [14] focus on the aesthetic of the layout with the goal to present graphs neatly or with as few intersecting edges as possible. The ways to achieve such a layout are also manifold: [8] uses circular dilation, [3] chooses a combination of greedy and empirical, and [2] uses a force-directed and fuzzy multilevel approach.

The Circle Index is novel and fills a great gap: there are no quality measures for orderings of graphs yet. Existing quality measures are usually for clusterings and regard only the partitioning, but not the distribution of the nodes inside a cluster. Apart from that there are altogether only a few measures for clusterings without ground-truth — which is not available for most real world datasets. [1] compare those common quality metrics and discover that they all have some weaknesses, most of which cannot be true for the Circle Index by definition: They give better results for smaller numbers of clusters (Modularity, Conductance, Coverage) or larger ones (Silhouette Coefficient, Performance). They cannot deal with high numbers of singleton clusters (Silhouette Coefficient, Modularity) or with large networks (Performance). They do not regard internal cluster/edge density (Coverage, Conductance) or are expensive to calculate (Silhouette Coefficient). All of those metrics are based on the number of edges between different clusters and the number of edges inside the clusters, so those biases are founded already in their definitions. By contrast, Circle Index regards both types of edges (between and inside clusters) indirectly using the circular arrangement of nodes.

Comparative Methods. To demonstrate the expressiveness of the Circle Index, we compare CirClu to different clustering algorithms of diverse areas in Sec. 6, where we use the adjacency matrix as input if necessary. We compare with centroid based k-Means [13], two hierarchical agglomerative clustering methods [22] and Agglomod [16], Eigenvector based Spectral Clustering [18, 20, 21], message passing Affinity Propagation [10], and modularity minimizing Community Louvain Algorithm [4].

Internal Quality Measures of Clusterings. We compare the Circle Index which we adapt for clusterings in Sec. 6 with Modularity, Coverage, and Performance. Modularity measures the strength of division into clusters and gives information about the community structure of networks. Coverage is the relation of the summarized weight of intra-cluster edges to the weight of all edges. Performance is the relation between internal edges in a cluster and edges that do not exist between cluster's nodes to other nodes in the graph.

Graph Orderings. There are related algorithms, which order graphs, but, to the best of our knowledge, they are all looking for linear orderings. [7] is based on the spectrum of the Laplacian, using the Fiedler vector [9], the eigenvector corresponding to the second lowest eigenvalue, to either sort a graph or bisect it. Cuthill-McKee [5] orders nodes s.t. the adjacency matrix is a band matrix with small bandwidth. Those sortings may seem similar to CirClu, but since nodes are ordered linearly, there is a first and a last node which is not naturally for all graphs but trees. Thus, especially cyclic and strongly connected graphs can not be ordered well, as Sec. 6 shows. In a topological ordering [11] for every directed edge (u, v) in a graph, u comes before v. That is only possible if the graph has no directed cycles, i.e., the graph is a tree, and it does not lead to a clustering which finds the characteristically highly interconnected groups. Newer algorithms like [23] look for graph orderings to speedup CPU computing to enhance efficiency of graph algorithms, but do not group similar or highly interconnected nodes together, but such which are frequently accessed together.

3 PRELIMINARIES

Circular Mean. The mean M of a point set N is the point with the lowest possible sum of distances to all points in N, colored red in Fig. 2 for $N = P, Q, R$. The circular mean CM of a point set where all points lie on a circle is the point on the circle line which is closest to the mean of those points, colored purple in the figure.

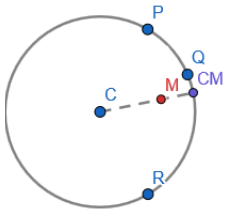


Figure 2: The mean M and the circular mean CM of points P, Q and R.

Edge Length. If we arrange all nodes of a graph of size n uniformly in a circle of arbitrary constant radius r we can use basic geometry to calculate the length of an edge, which corresponds to the distance between the according nodes. For the i-th node n_i and the j-th node n_j in the circle we can calculate their distance $d(n_i, n_j)$ as follows:

$$d(n_i, n_j) = 2r * sin\left(\pi \frac{|i-j|}{n}\right) \tag{1}$$

Average Edge Length. Given an ordered graph $G = (V, E)$, where "ordered" means that there exists an injective mapping $f : V \to \{0, 1, ..., |V|\}$, where the nodes are uniformly arranged in a unit-sphere following the order given by f, we can compute the Average Edge Length: Let $e.n_1$ refer to the source node of edge e and $e.n_2$ refer to its target node. Using Eq. 1, we get for the Average Edge Length $d(G)$ of a graph:

$$d(G) = \frac{1}{|E|} \sum_{e \in E} d(e.n_1, e.n_2) = \frac{2}{|E|} \sum_{e \in E} sin\left(\pi \frac{|f(e.n_1) - f(e.n_2)|}{|V|}\right) \tag{2}$$

Lower Bound for Average Edge Length. A lower bound b_l for the Average Edge Length can be approximated efficiently by regarding all nodes independently and assuming they all lie in the middle of their neighbors, which are located as near as possible to the respective node. This lower bound can only be reached for some special graphs, but captures the structure of the graph enough for our purpose. Let $N(n)$ be the neighbors of node n and div returns the integer quotient of an Euclidean division, then:

$$b_l(G) = \frac{1}{|E|} \sum_{n \in V} \sum_{i=1}^{|N(n)|} sin\left(\pi \frac{i \; div \; 2}{|V|}\right) \tag{3}$$

The 2 in the numerator of Eq. 2 is neutralized as every edge is counted twice. $i\ div\ 2$ origins in the assumed optimal alignment: the minimal edge length from a node to its neighbor is the length to the very next slot on the circle if the node has maximal two neighbors (one left, one right). The third and forth neighbor would then each be 2 slots away and so on, resulting in $i\ div\ 2$.

Minimum Cut. To obtain clusters based on the ordered graph, we partition it similar to the Minimum Cut (MinCut) problem. Where the original MinCut has exponentially many possibilities to set cuts, we have only $|V|$, starting in the center of the circle. The number of edges cut can easily be counted and should be low to produce a good separation of two clusters. To avoid getting clusters of only one or a few rather outlying points we do not minimize the number of edges cut per se, but the ratio R of cut edges to possible cut edges:

$$R = \frac{|E_C \cap (U \times W)|}{|U| \cdot |W|}. \tag{4}$$

U and W are two disjoint sets in which the cut divides the nodes of a cluster C and E_C are all edges lying in the cluster which is split. Edges between nodes which are both not in the cut cluster C are not counted.

4 CIRCLE INDEX

To obtain comparable values for the quality of orderings of a graph we take the average edge length as well as its lower bound into account. Otherwise we would receive lower (meaning better quality) values for larger or sparser graphs. Thus we define the Circle-Index $CI(G)$ of a graph as relation between the average edge length $d(G)$ of the graph and the lower bound $b_l(G)$ for it:

$$CI(G) = \frac{d(G)}{b_l(G)} \tag{5}$$

The Circle Index closes two gaps in the field of quality measures for graphs: While the ordering of nodes in a graph is not only an important preprocessing step in many clustering algorithms, but also highly beneficial for soft and fuzzy clustering, there are no quality measures for it, yet. Note that every graph clustering algorithm needs to use *some* ordering of the nodes for the input of the graph alone. Second, it allows to evaluate orderings without knowing the labeling, which is very often not available. It particularly overcomes the limitations of existing measures for evaluation of clusterings discussed in Sec. 2.

5 CIRCLU

CirClu is a simple iterative algorithm minimizing the Circle Index CI and performing a MinCut afterwards. As points of a graph $G = (V, E)$ lie uniformly distributed in a circle, there are $|V|$ possible slots a node can occupy, which are handled as a linked list. One by one, each node is moved to the slot nearest to the circular mean (see Sec. 3) of its neighbors, which is a greedy minimization of this point's edge lengths. Nodes between the new slot of the moved node and the old slot move up into the direction where less nodes have to move (to close the gap emerged at the old slot and make place at the new slot).

When the CI does not decrease anymore, we reached a local minimum and can now easily compute where to set cuts similar to the MinCut problem to partition the graph: we only have $|V|$ possibilities to set the cut, where there would be $2^{|V|}$ different possibilities to cut arbitrarily through the graph. A cut is performed from the center of the circle to the circle line and the intersected edges are counted for every possible cut. The cuts are ordered by ascending R as defined in Eq. 4 and set one by one until the desired number of clusters is reached. The first partitioning emerges with the second cut, afterwards every cut generates a new cluster by dividing an old.

Weighted graphs. To handle weighted graphs we can modify equations 2 and 3 by multiplying the length of an edge e with its weight $w(e)$. With that nodes with higher weighted edges are moved closer together than those with lower weighted ones.

Runtime Efficiency. In every iteration step the calculation of the optimal position for a node takes $O(|E|)$ since every node has at most $|E|$ neighbors. This calculation is done for every node, so the runtime until the graph is completely ordered is $O(i * |E| * |V|)$, for i the number of iterations until convergence. In the second step there are $|V|$ possibilities to make a cut between two clusters. For every possibility the ratio R from Equation 4 is calculated which needs $O(|E|)$ each time, so altogether $O(|E| * |V|)$. With those two steps we obtain a runtime of $O(i * |E| * |V|)$ for the complete algorithm.

6 EXPERIMENTAL EVALUATION

We compare the Circle Index (CI) with internal quality measures for clusters, Modularity, Coverage, Performance, and Conductance. To evaluate the results w.r.t. real world data, we regarded only non-synthetic data with a given ground truth. To be independent of flaws in diverse external quality measures aligning clustering results with labels, we use the average of Normalized Mutual Information, Adjusted Rand Index, V-Measure, and Adjusted Mutual Information, so that their biases compensate for each other, and abbreviate this average with EM (for external measures). Simultaneously, we compare k-Means (KM) [13], Agglomerative Clustering (AC) [22], Spectral Clustering (SC) [18, 20, 21], Affinity Propagation (AP) [10], the Community Louvain Algorithm (CL) [4] and Agglomod (A) [16] with CirClu in regards of both, the internal as well as the external measurements. To obtain the Circle Index for non-ordered, but clustered data, the nodes within each cluster are sorted by their degree. We used public python implementations on a 32 GB RAM, 3.4GHz machine for all experiments.[1]

We conducted experiments on the social networks "Zacharys Karate Club" [24] and "Strike" [6, 15], on the trading network "Worldtrade" [6, 19], and the co-occurrence network "Les Misérables" [12]. Results are shown in Fig. 3, where normCI is the normalized CI. The CI was normalized separately for every dataset to values between 0 and 1, corresponding to the worst resp. best result, by a linear normalization and subsequent subtraction from 1[2], to allow a simple visual analysis. We note, that in all cases, the best CI implies the best results w.r.t. the ground truth, i.e., the highest EM. As there are no appropriate ordering algorithms yet, and we could only compare with clustering algorithms, the inversion does not hold:

[1] Our code is online available under: https://github.com/p4nna/CirClu

[2] In detail, for all CIs, we subtracted the minimal occurring CI, divided this value by the range of occurring values for CI, and subtracted the result from 1.

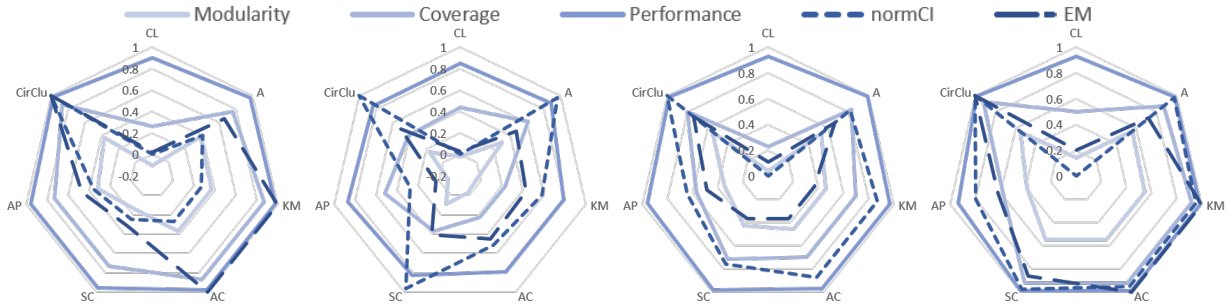


Figure 3: Results for datasets (left to right): Zacharys Karate Club, Worldtrade, Les Misérables, and Strike for algorithms Community Louvain (CL), Agglomod (A), k-Means (KM), Agglomerative Clustering (AC), Spectral Clustering (SC), Affinity Propagation (AP), and CirClu. EM is the average of the External Measures, and normCI the normalized Circle Index

a good clustering does not imply a good ordering as there are too many possible permutations of nodes inside each cluster. CirClu achieved one of the best results for all those datasets, even though they heavily discern in their form: Zacharys Karate Club and Strike have only two clusters, Worldtrade is highly interconnected with rather blurry clusters than clique-like ones, and Les Misérables has several hubs, the protagonists. We also note, that the trend of normCI and EM are, even though stretched for some algorithms, more similar to each other than to the other internal measurements, resulting in similar shapes on the radar charts in Fig. 3.

7 CONCLUSION

We introduced the Circle Index, an internal measure for the quality of an ordering of a graph, where there is no comparable measure yet. Sec. 6 showed, that the Circle Index is also applicable for clusterings without an underlying order inside the clusters. In comparison to established internal measures like Modularity, Coverage, or Performance it is often a better prediction for the quality of a clustering w.r.t. the ground truth. Our new clustering algorithm CirClu orders the nodes of a graph by minimizing the Circle Index and then partitions the graph by finding the minimum cuts inside that ordering. It works well for networks with almost all types of clusters.The embedding onto the unit sphere is natural and does not force the graph to have a first node and a last node (which makes only sense for loop free graphs which are rather rare). Every graph clustering algorithm needs to be given the graph in *any* order, so every clustering algorithm has to deal with it implicitly, even though most of them do not mention this at all. Our new concepts are extensible and build a broad basis to build on. We did not cover directed graphs yet, and different normalizations of the Circle Index should be discussed. Also we did not investigate the usage for Big Data yet, which we plan for future work, since we wanted to focus on the meaningfulness of found clusters. Besides it would be interesting to know how a higher dimensional sphere would affect the algorithm.

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] Hélio Almeida, Dorgival Guedes, Wagner Meira, and Mohammed J Zaki. 2011. Is there a best quality metric for graph clusters?. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 44–59.

[2] Mohammadreza Ashouri, Ali Golshani, Dara Moazzmi, and Mandana Ghasemi. 2016. Graphs Drawing through Fuzzy Clustering. *arXiv preprint arXiv:1603.07011* (2016).

[3] Michael Baur and Ulrik Brandes. 2004. Crossing reduction in circular layouts. *WG* 3353 (2004), 332–343.

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[5] Elizabeth Cuthill and James McKee. 1969. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*. ACM, 157–172.

[6] Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. 2011. *Exploratory social network analysis with Pajek*. Vol. 27. Cambridge University Press.

[7] Chris Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst Simon. 2001. Spectral min-max cut for graph partitioning and data clustering. (2001).

[8] Uğur Doğrusöz, Brendan Madden, and Patrick Madden. 1996. Circular layout in the graph layout toolkit. In *International Symposium on Graph Drawing*. Springer, 92–100.

[9] Miroslav Fiedler. 1973. Algebraic connectivity of graphs. *Czechoslovak mathematical journal* 23, 2 (1973), 298–305.

[10] Brendan J Frey and Delbert Dueck. 2007. Clustering by passing messages between data points. *science* 315, 5814 (2007), 972–976.

[11] Arthur B Kahn. 1962. Topological sorting of large networks. *Commun. ACM* 5, 11 (1962), 558–562.

[12] Donald Ervin Knuth. 1993. *The Stanford GraphBase: a platform for combinatorial computing*. Vol. 37. Addison-Wesley Reading.

[13] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[14] Erkki Mäkinen. 1988. On circular layouts. *International Journal of Computer Mathematics* 24, 1 (1988), 29–37.

[15] Judd H Michael. 1997. Labor dispute reconciliation in a forest products manufacturing facility. *Forest products journal* 47, 11/12 (1997), 41.

[16] Mark EJ Newman. 2004. Fast algorithm for detecting community structure in networks. *Physical review E* 69, 6 (2004), 066133.

[17] Gonzalo E Paredes and Luis S Vargas. 2012. Circle-Clustering: A new heuristic partitioning method for the clustering problem. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE, 1–8.

[18] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000), 888–905.

[19] David A Smith and Douglas R White. 1992. Structure and dynamics of the global economy: network analysis of international trade 1965–1980. *Social forces* 70, 4 (1992), 857–893.

[20] X Yu Stella and Jianbo Shi. 2003. Multiclass spectral clustering. In *Proceedings Ninth IEEE International Conference on Computer Vision*. IEEE, 313.

[21] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17, 4 (2007), 395–416.

[22] Joe H Ward Jr. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association* 58, 301 (1963), 236–244.

[23] Hao Wei, Jeffrey Xu Yu, Can Lu, and Xuemin Lin. 2016. Speedup graph processing by graph ordering. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1813–1828.

[24] Wayne W Zachary. 1977. An information flow model for conflict and fission in small groups. *Journal of anthropological research* 33, 4 (1977), 452–473.

Bibliography

[1] Elke Achtert, Christian Böhm, Jörn David, Peer Kröger, and Arthur Zimek. "Global correlation clustering based on the hough transform". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 1.3 (2008), pp. 111–127.

[2] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. "Robust, complete, and efficient correlation clustering". In: *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM. 2007, pp. 413–418.

[3] Elke Achtert, Christian Böhm, Peer Kröger, and Arthur Zimek. "Mining hierarchies of correlation clusters". In: *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*. IEEE. 2006, pp. 119–128.

[4] Charu C. Aggarwal and Philip S. Yu. "Finding generalized projected clusters in high dimensional spaces". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 70–81.

[5] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. "Automatic subspace clustering of high dimensional data for data mining applications". In: *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. 1998, pp. 94–105.

[6] Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. "A comparison of extrinsic clustering evaluation metrics based on formal constraints". In: *Information retrieval* 12.4 (2009), pp. 461–486.

[7] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. "OPTICS: Ordering points to identify the clustering structure". In: *ACM Sigmod record* 28.2 (1999), pp. 49–60.

[8] David Arthur and Sergei Vassilvitskii. "K-Means++: The Advantages of Careful Seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2007, 1027—1035.

[9] Ira Assent. "Clustering high dimensional data". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.4 (2012), pp. 340–350.

[10] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. "Correlation clustering". In: *Machine learning* 56.1 (2004), pp. 89–113.

[11] Sugato Basu, Arindam Banerjee, and Raymond Mooney. "Semi-supervised Clustering by Seeding". In: *In Proceedings of 19th International Conference on Machine Learning (ICML-2002)*. 2002, pp. 27–34.

[12] Christian Baumgartner, Claudia Plant, Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. "Subspace selection for clustering high-dimensional data". In: *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE. 2004, pp. 11–18.

[13] Michael Baur and Ulrik Brandes. "Crossing reduction in circular layouts". In: *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer. 2004, pp. 332–343.

[14] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. "The R*-tree: An efficient and robust access method for points and rectangles". In: *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*. 1990, pp. 322–331.

[15] Anna Beer, Ekaterina Allerborn, Valentin Hartmann, and Thomas Seidl. "KISS - A fast kNN-based Importance Score for Subspaces". In: *Proceedings of the 24th International Conference on Extending Database Technology (EDBT)*. 2021, pp. 391–396. DOI: 10.5441/002/edbt.2021.40.

[16] Anna Beer, Valentin Hartmann, and Thomas Seidl. "Orderings of Data - More Than a Tripping Hazard: Visionary". In: *Proceedings of the 32nd International Conference on Scientific and Statistical Database Management (SSDBM)*. Association for Computing Machinery, 2020, 17:1–17:4. DOI: 10.1145/3400903.3400911.

[17] Anna Beer, Daniyal Kazempour, Julian Busch, Alexander Tekles, and Thomas Seidl. "Grace - Limiting the Number of Grid Cells for Clustering High-Dimensional Data". In: *Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen" (LWDA)*. 2020, pp. 11–22. URL: http://ceur-ws.org/Vol-2738/LWDA2020_paper_11.pdf.

[18] Anna Beer, Daniyal Kazempour, Lisa Stephan, and Thomas Seidl. "LUCK- Linear Correlation Clustering Using Cluster Algorithms and a KNN Based Distance Function". In: *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*. SSDBM '19. New York, NY, USA: Association for Computing Machinery, 2019, 181—184. DOI: 10.1145/3335783.3335801.

[19] Anna Beer, Jennifer Lauterbach, and Thomas Seidl. "MORe++: k-Means Based Outlier Removal on High-Dimensional Data". In: *Proceedings of the International Conference on Similarity Search and Applications (SISAP)*. Springer, Cham. 2019, pp. 188–202. DOI: 10.1007/978-3-030-32047-8_17.

[20] Anna Beer, Nadine Sarah Schüler, and Thomas Seidl. "A Generator for Subspace Clusters". In: *Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen" (LWDA)*. Vol. 2454. CEUR Workshop Proceedings. 2019, pp. 69–73. URL: http://ceur-ws.org/Vol-2454/paper_29.pdf.

[21] Anna Beer, Dominik Seeholzer, Nadine-Sarah Schüler, and Thomas Seidl. "Angle-Based Clustering". In: *Proceedings of the International Conference on Similarity Search and Applications*. Cham: Springer International Publishing, 2020, pp. 312–320. DOI: 10.1007/978-3-030-60936-8_24.

[22] Anna Beer and Thomas Seidl. "Graph Ordering and Clustering: A Circular Approach". In: *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*. SSDBM '19. Association for Computing Machinery, 2019, 185—188. DOI: 10.1145/3335783.3335802.

[23] Anna Beer, Lisa Stephan, and Thomas Seidl. "LUCKe — Connecting Clustering and Correlation Clustering". In: *2021 International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2021, pp. 431–440. DOI: 10.1109/ICDMW53433.2021.00059.

[24] Richard E. Bellman. *Adaptive control processes*. Princeton university press, 1961.

[25] Jon Louis Bentley. "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9 (1975), pp. 509–517.

[26] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. "When is "nearest neighbor" meaningful?" In: *International conference on database theory*. Springer. 1999, pp. 217–235.

[27] Christian Böhm, Karin Kailing, Peer Kröger, and Arthur Zimek. "Computing clusters of correlation connected objects". In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 2004, pp. 455–466.

[28] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. "LOF: identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104.

[29] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[30] Sanjay Chawla and Aristides Gionis. "k-means–: A unified approach to clustering and outlier detection". In: *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM. 2013, pp. 189–197.

[31] Chun-Hung Cheng, Ada Waichee Fu, and Yi Zhang. "Entropy-based subspace clustering for mining numerical data". In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. 1999, pp. 84–93.

[32] Elizabeth Cuthill and James McKee. "Reducing the bandwidth of sparse symmetric matrices". In: *Proceedings of the 1969 24th national conference*. ACM. 1969, pp. 157–172.

[33] Rajesh N. Davé and Raghuram Krishnapuram. "Robust clustering methods: a unified view". In: *IEEE Transactions on Fuzzy Systems* 5.2 (1997), pp. 270–293.

[34] Zhaohong Deng, Kup-Sze Choi, Yizhang Jiang, Jun Wang, and Shitong Wang. "A survey on soft subspace clustering". In: *Information sciences* 348 (2016), pp. 84–106.

[35] Levent Ertöz, Michael Steinbach, and Vipin Kumar. "Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data". In: *Proceedings of the 2003 SIAM international conference on data mining*. SIAM. 2003, pp. 47–58.

[36] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996, 226—231.

[37] Martin Ester and Jörg Sander. *Knowledge discovery in databases: Techniken und Anwendungen*. Springer-Verlag, 2013.

[38] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From Data Mining to Knowledge Discovery in Databases". In: *AI Magazine* 17.3 (1996), pp. 37–54.

[39] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "The KDD process for extracting useful knowledge from volumes of data". In: *Communications of the ACM* 39.11 (1996), pp. 27–34.

[40] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. "Knowledge discovery in databases: An overview". In: *AI magazine* 13.3 (1992), pp. 57–70.

[41] Guojun Gan and Michael Kwok-Po Ng. "K-means clustering with outlier removal". In: *Pattern Recognition Letters* 90 (2017), pp. 8–14.

[42] J. C. Gower. "A Comparison of Some Methods of Cluster Analysis". In: *Biometrics* 23.4 (1967), pp. 623–637.

[43] Antonin Guttman. "R-trees: A dynamic index structure for spatial searching". In: *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 1984, pp. 47–57.

[44] Ville Hautamaki, Ismo Karkkainen, and Pasi Franti. "Outlier detection using k-nearest neighbour graph". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 3. IEEE. 2004, pp. 430–433.

[45] Douglas M. Hawkins. *Identification of outliers*. Vol. 11. Springer, 1980.

[46] Janis Held, Anna Beer, and Thomas Seidl. "Chain-detection Between Clusters". In: *Datenbank-Spektrum* 19.3 (2019), pp. 219–230. DOI: 10.1007/s13222-019-00324-9.

[47] Janis Held, Anna Beer, and Thomas Seidl. "Chain-detection for DBSCAN". In: *BTW 2019 - Datenbanksysteme für Business, Technologie und Web (Workshops)*. Gesellschaft für Informatik, Bonn, 2019, pp. 173–183. DOI: 10.18420/btw2019-ws-18.

[48] David Hilbert. "Ueber die reellen Züge algebraischer Curven". In: *Mathematische Annalen* 38.1 (1891), pp. 115–138.

[49] Alexander Hinneburg and Daniel A. Keim. "Optimal Grid-Clustering : Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering". In: *Proceedings of the 25 th International Conference on Very Large Databases*. 1999, pp. 506–517.

[50] Victoria Hodge and Jim Austin. "A survey of outlier detection methodologies". In: *Artificial intelligence review* 22.2 (2004), pp. 85–126.

[51] Ian T. Jolliffe. "Principal Components in Regression Analysis". In: *Principal Component Analysis*. Springer New York, 1986, pp. 129–155.

[52] Karin Kailing, Hans-Peter Kriegel, Peer Kröger, and Stefanie Wanka. "Ranking interesting subspaces for clustering high dimensional data". In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer. 2003, pp. 241–252.

[53] Daniyal Kazempour, Anna Beer, Friederike Herzog, Daniel Kaltenthaler, Johannes-Y. Lohrer, and Thomas Seidl. "FATBIRD: A Tool for Flight and Trajectories Analyses of Birds". In: *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE. 2018, pp. 75–82. DOI: 10.1109/eScience.2018.00023.

[54] Daniyal Kazempour, Anna Beer, Peer Kröger, and Thomas Seidl. "I fold you so! An internal evaluation measure for arbitrary oriented subspace clustering". In: *2020 International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2020, pp. 316–323. DOI: 10.1109/ICDMW51313.2020.00051.

[55] Daniyal Kazempour, Anna Beer, Johannes-Y. Lohrer, Daniel Kaltenthaler, and Thomas Seidl. "PARADISO: an interactive approach of parameter selection for the mean shift algorithm". In: *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. 2018, pp. 1–4. DOI: 10.1145/3221269.3221299.

[56] Daniyal Kazempour, Kilian Emmerig, Peer Kröger, and Thomas Seidl. "Detecting Global Periodic Correlated Clusters in Event Series based on Parameter Space Transform". In: *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*. 2019, pp. 222–225.

[57] Daniyal Kazempour, Markus Mauder, Peer Kröger, and Thomas Seidl. "Detecting global hyperparaboloid correlated clusters based on hough transform". In: *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. 2017, pp. 1–6.

[58] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. "Density-based clustering". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.3 (2011), pp. 231–240.

[59] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 3.1 (2009), pp. 1–58.

[60] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. "Subspace clustering". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.4 (2012), pp. 351–364.

[61] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. "Angle-based outlier detection in high-dimensional data". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 444–452.

[62] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. "Understanding of internal clustering validation measures". In: *2010 IEEE international conference on data mining*. IEEE. 2010, pp. 911–916.

[63] Stuart P. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–136.

[64] James MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. 1967, pp. 281–297.

[65] Erkki Mäkinen. "On circular layouts". In: *International Journal of Computer Mathematics* 24.1 (1988), pp. 29–37.

[66] Fionn Murtagh and Pedro Contreras. "Algorithms for hierarchical clustering: an overview". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.1 (2012), pp. 86–97.

[67] Harsha Nagesh, Sanjay Goil, and Alok Choudhary. "Adaptive grids for clustering massive data sets". In: *Proceedings of the 2001 SIAM International Conference on Data Mining*. SIAM. 2001, pp. 1–17.

[68] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. "On spectral clustering: Analysis and an algorithm". In: *Advances in neural information processing systems*. 2002, pp. 849–856.

[69] Sandra Obermeier, Anna Beer, Florian Wahl, and Thomas Seidl. "Cluster Flow - an Advanced Concept for Ensemble-Enabling, Interactive Clustering". In: *BTW*. Vol. P-311. LNI. Gesellschaft für Informatik, Bonn, 2021, pp. 175–194. DOI: 10.18420/btw2021-09.

[70] Jack A. Orenstein and Tim H. Merrett. "A Class of Data Structures for Associative Searching". In: *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*. 1984, pp. 181–190.

[71] Lance Parsons, Ehtesham Haque, and Huan Liu. "Subspace clustering for high dimensional data: a review". In: *ACM SIGKDD explorations newsletter* 6.1 (2004), pp. 90–105.

[72] Tao Pei, Ajay Jasra, David J. Hand, A.-Xing Zhu, and Chenghu Zhou. "DECODE: a new method for discovering clusters of different densities in spatial data". In: *Data Mining and Knowledge Discovery* 18.3 (2009), pp. 337–369.

[73] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. "A review of novelty detection". In: *Signal Processing* 99 (2014), pp. 215–249.

[74] Eréndira Rendón, Itzel Abundez, Alejandra Arizmendi, and Elvia M. Quiroz. "Internal versus external cluster validation indexes". In: *International Journal of computers and communications* 5.1 (2011), pp. 27–34.

[75] Karlton Sequeira and Mohammed Zaki. "SCHISM: a new approach to interesting subspace mining". In: *International Journal of Business Intelligence and Data Mining* 1.2 (2005), pp. 137–160.

[76] Lucas Vendramin, Ricardo J. G. B. Campello, and Eduardo R. Hruschka. "Relative clustering validity criteria: A comparative overview". In: *Statistical analysis and data mining: the ASA data science journal* 3.4 (2010), pp. 209–235.

[77] Nguyen Xuan Vinh, Julien Epps, and James Bailey. "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance". In: *The Journal of Machine Learning Research* 11 (2010), pp. 2837–2854.

[78] Ulrike Von Luxburg, Robert C. Williamson, and Isabelle Guyon. "Clustering: Science or art?" In: *Proceedings of ICML workshop on unsupervised and transfer learning*. JMLR Workshop and Conference Proceedings. 2012, pp. 65–79.

[79] Rui Xu and Don Wunsch. *Clustering*. Vol. 10. John Wiley & Sons, 2008.

[80] M.-S. Yang. "A survey of fuzzy clustering". In: *Mathematical and Computer modelling* 18.11 (1993), pp. 1–16.

[81] Arthur Zimek and Peter Filzmoser. "There and back again: Outlier detection between statistical reasoning and data mining algorithms". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.6 (2018).

[82] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. "A survey on unsupervised outlier detection in high-dimensional numerical data". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 5.5 (2012), pp. 363–387.