2021 International Conference on Data Mining Workshops (ICDMW) | 978-1-6654-2427-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDMW53433.2021.00059

# 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**

## 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

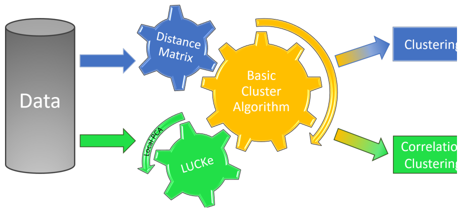


Fig. 1. Concept. Using LUCKe instead of an, e.g. Euclidean distance matrix when applying any basic clustering algorithm yields correlation clusters instead of traditional ones

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 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 $\varepsilon$-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 $\varepsilon$- 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 $\lambda$, and a threshold $\delta$ 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* $\lambda_P$ 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 $\varepsilon$, 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: $\Delta$ specifies a certain degree of allowed deviation when checking the approximate linear dependency. $\delta$ 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 $\tau$. 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 $Matrix$
```

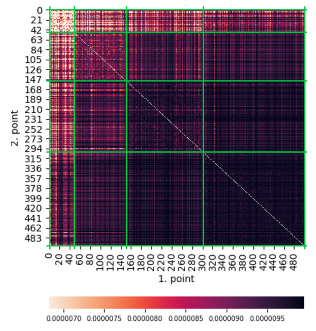
## A. Computation of the Eigensystem

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.

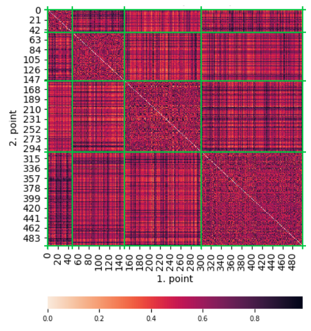
## B. Adaptation of the Eigensystem

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 $\Omega$ 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 $\Omega$ 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.

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

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:

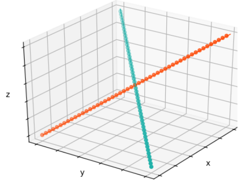
*a) Efficiency:* The average runtime to calculate the distance resp. similarity matrix on our base case data sets depending on the data set size $n$ for diverse $k$ is shown in Fig. 4. It fits our complexity calculation in Sec. III-E3.

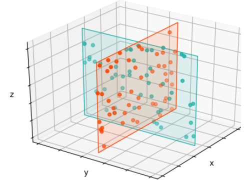
*b) Basic Setting:* LUCKe is tested in combination with DBSCAN, Spectral Clustering and Agglomerative Clustering for the five data sets in the basic setting (see Fig. 3), results are shown in Fig. 5. For Data set XL and PH, all four algorithms yield very good results with an NMI between $0.95 - 0.98$ 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.

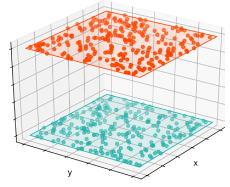
*c) Size of Data set:* The five data sets are tested with data set sizes of $n = 1000$, $n = 2000$, $n = 3000$, $n = 4000$ and $n = 5000$ in addition to the default setting. The results regarding the basic setting do not change significantly with increasing $n$ for any of the base cases.

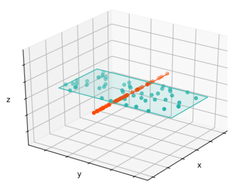
*d) Number of Dimensions:* We tested all base case results with dimensionalities 4, 6, 9, 12, 20, and 35. All four algorithms detect crossing lines in Data set XL well even at $dim = 35$, while for Data set XHDD, none of the algorithms can produce a reasonable result as the number of dimensions increases, 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

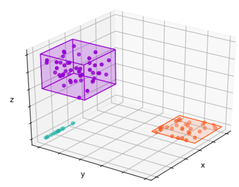

(e) Data set HDD: several hyperplanes of different dimensionality

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

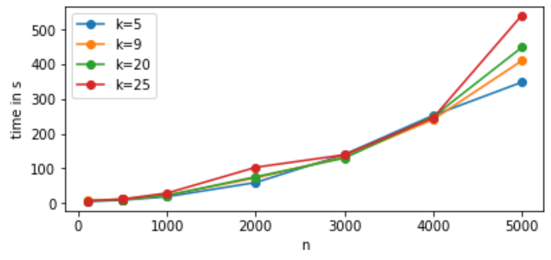


Fig. 4. Runtime depending on data set size $n$ for different $k$

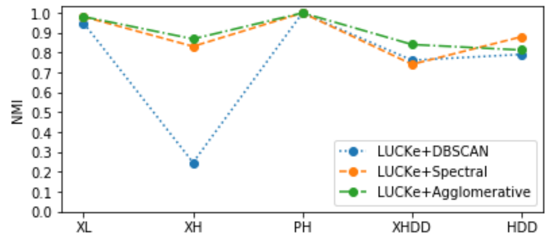


Fig. 5. Results for the basic setting for the five data sets and the four tested LUCKe variants

*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.

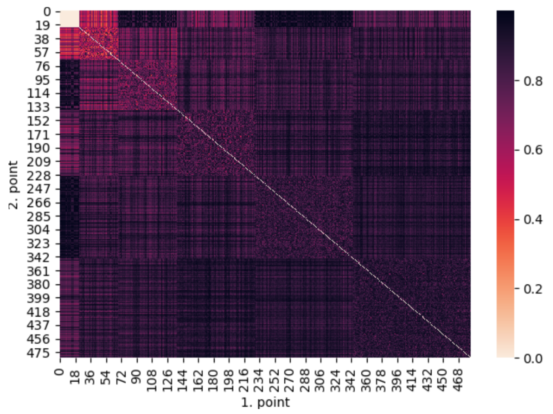


Fig. 6. LUCKe-distance matrix of points on hyperplanes of different (ascendingly ordered) dimensionalites in the 6-dimensional Data set HDD

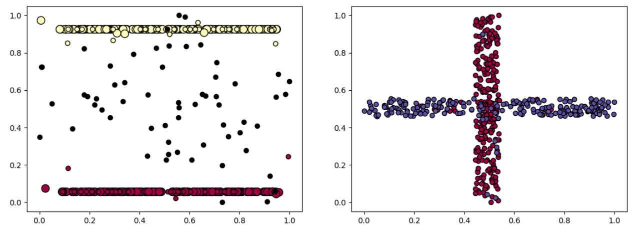


Fig. 7. Clustering results. Left: Data set PH with 15% noise. Right: Data set XH with $jitter = 0.05$.

*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

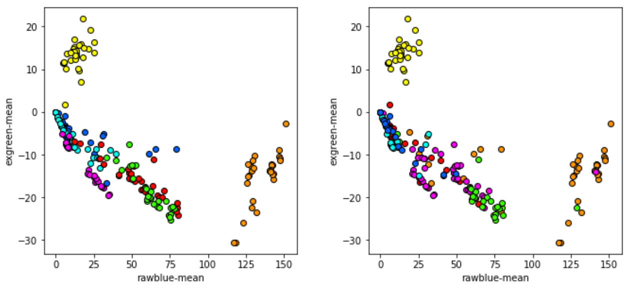


Fig. 8. Two descriptive dimensions of the Image Segmentation data set, colored by labels on the left, colored by clusters as found by LUCKe+Spectral on the right.

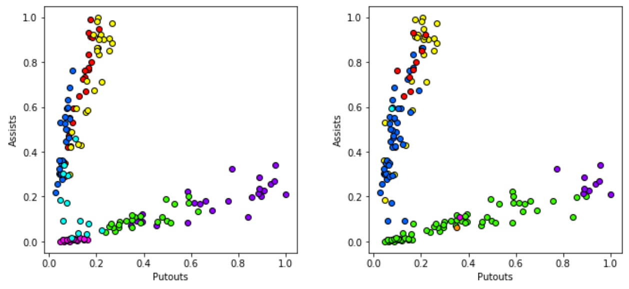


Fig. 9. Two descriptive dimensions of the Hitters data set, colored by labels on the left, colored by clusters as found by LUCKe+Agglomerative on the right.

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.

## 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 straight-forward 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*

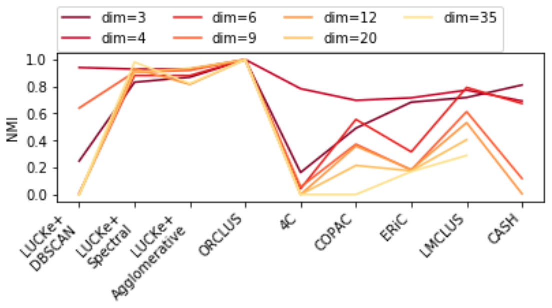


Fig. 10. NMI for different $dim$ for Data set XH

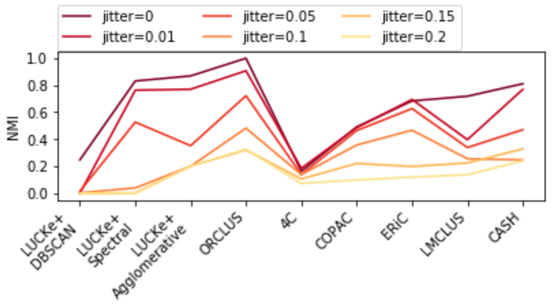


Fig. 11. NMI for different $jitter$ for Data set XH

### B. *Data set PH*

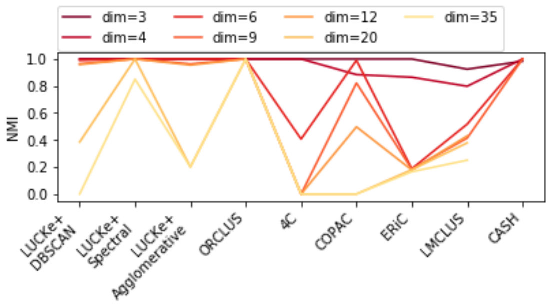


Fig. 12. NMI for different $dim$ for Data set PH

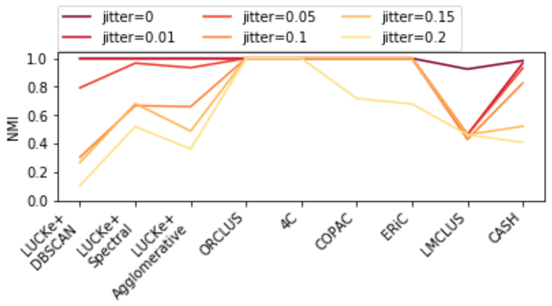


Fig. 13. NMI for different $jitter$ for Data set PH

### C. *Data set XHDD*

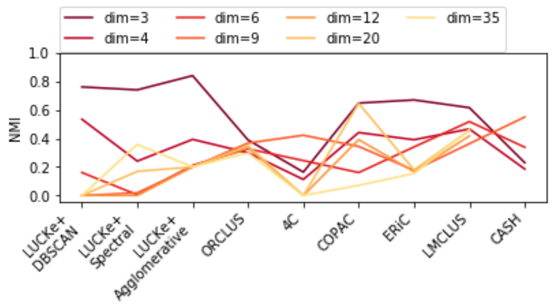


Fig. 14. NMI for different $dim$ for Data set XHDD

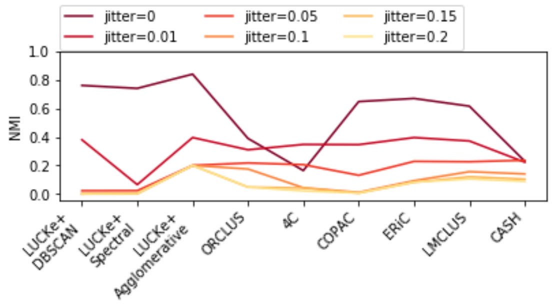


Fig. 15. NMI for different $jitter$ for Data set XHDD

### D. *Data set HDD*

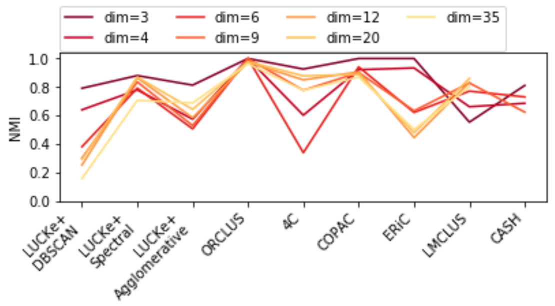


Fig. 16. NMI for different $dim$ for Data set HDD

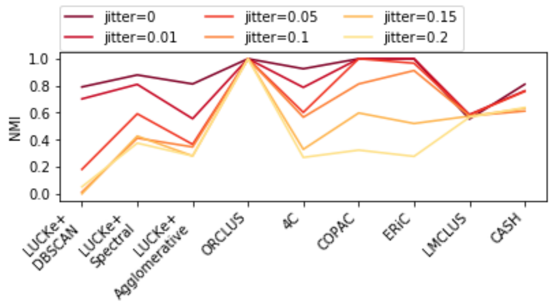


Fig. 17. NMI for different $jitter$ for Data set HDD