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.

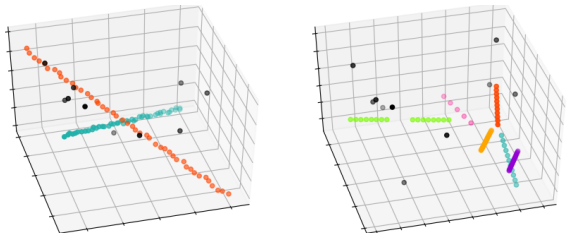

Figure 1: Base case experiments

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.

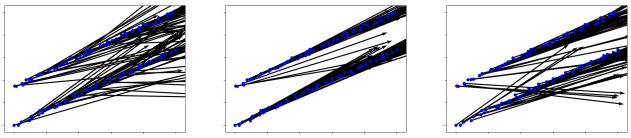

Figure 2: Orientation vectors for $k = 5$, $k = 15$, and $k = 25$

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

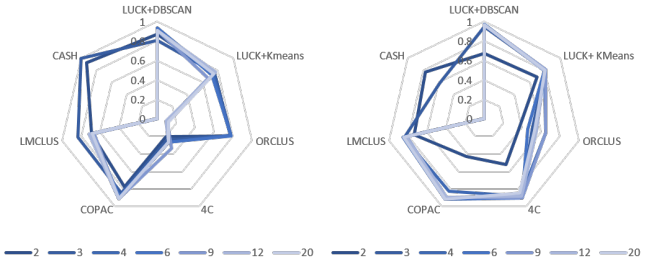

Figure 3: ARI for growing number of dimensions

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.

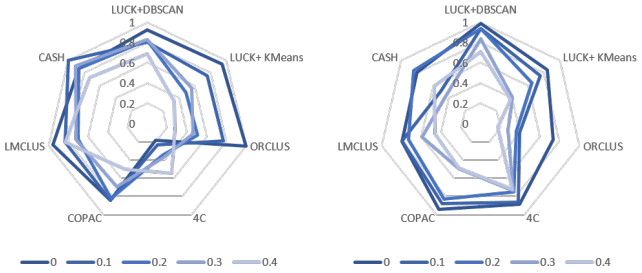

Figure 4: ARI for different percentages of noise

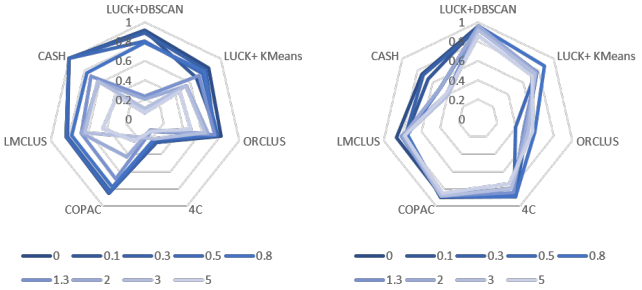

Figure 5: ARI for different levels of jitter

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.