Efficient Cluster Detection by Ordered Neighborhoods

Emin Aksehirli[1], Emmanuel Müller[1,2], and Bart Goethals[1]

[1] University of Antwerp, Belgium, firsname.lastname@uantwerpen.be
[2] Karlsruhe Institute of Technology, Germany, emmanuel.mueller@kit.edu

Abstract. Detecting cluster structures seems to be a simple task, i.e. separating similar from dissimilar objects. However, given today's complex data, (dis-)similarity measures and traditional clustering algorithms are not reliable in separating clusters from each other. For example, when too many dimensions are considered simultaneously, objects become unique and (dis-)similarity does not provide meaningful information to detect clusters anymore. While the (dis-)similarity measures might be meaningful for individual dimensions, algorithms fail to combine this information for cluster detection. In particular, it is the severe issue of a combinatorial search space that results in inefficient algorithms.

In this paper we propose a cluster detection method based on the *ordered neighborhoods*. By considering such ordered neighborhoods in each dimension individually, we derive properties that allow us to detect clustered objects in dimensions in linear time. Our algorithm exploits the ordered neighborhoods in order to find both the similar objects and the dimensions in which these objects show high similarity. Evaluation results show that our method is scalable with both database size and dimensionality and enhances cluster detection w.r.t. state-of-the-art clustering techniques.

1 Introduction

In the information era that we live in, there are a huge amount of data about almost anything. Institutions, both government and private, realized the importance of data and started to collect any kind of information on their business objects, hoping that they will derive useful knowledge out of it one day. Considering the amount, annotating and labeling the data is often infeasible. Therefore, unsupervised methods such as clustering are more suitable for knowledge extraction.

One of the challenging effects of this "data hoarding" is the increased number of attributes associated with each object. Unfortunately, due to the phenomenon of the so called *curse of dimensionality*, the similarity between objects becomes meaningless in high dimensional data spaces [5]. This means that the clusters cannot be separated from each other by (dis-)similarity assessment in high dimensional space, which in turn poses a serious challenge for traditional clustering algorithms such as k-means or hierarchical clustering. Nevertheless, (dis-)similarity in the individual dimensions can still be exploited and clusters can be detected in combinations of these dimensions. The open challenge is how to exploit this (dis-)similarity information in individual dimensions for any combination of dimensions while not falling prey to the exponential nature of this combinatorial search space.

For example, nowadays a customer can be associated with credit ratings, shopping habits, travel patterns, entertainment choices, sport habits, etc. Even medical conditions or other private information might be available due to the data integration with a multitude of data sources. Considering all the aforementioned data, each customer becomes very unique and almost equally dissimilar to any other customer. This makes the notion of "similar customers" meaningless. Nevertheless, a meaningful customer segmentation can still be achieved by looking at a subset of dimensions, e.g. travel and sport habits only.

Although similarity between high dimensional objects is not meaningful, similarity according to subsets of attributes is still meaningful. To address this issue, subspace clustering [20] aims at cluster detection in any combination of the given attributes. Even though they enhance clustering quality compared to traditional clustering methods, they come with their own challenges. They suffer from noise sensitivity [1,19], density estimation challenges [9,10,4], many complex parameters [4,10,12,13], or inefficient search through the combinatorial search space [2,9].

In contrast to these methods, we tackle the problem by exploiting local neighborhoods of objects in individual dimensions. These neighborhoods are used as means for cluster detection in combinations of dimensions. We show that clusters can be detected directly from these local neighborhoods. Moreover, exploiting their natural orders, we can avoid common scalability pitfalls in the algorithmic computation of clusters. In particular, we propose a linear-time algorithm for detecting cluster structures in dimensions. Key characteristics of our approach are its scalability w.r.t. both database size and dimensionality, the detection of clusters with variable scales, and its few user-friendly parameters. Further, it allows for individual (dis-)similarity measures for each dimension, which is important for data with different data types, complex distance functions, and the lack of joint (dis-)similarity measures in combinations of different data types and dimensions with complex distance functions.

In summary, our contributions are as follows: - A formal analysis of ordered neighborhoods in the context of cluster detection. - Prove completeness of cluster detection based on ordered neighborhoods. - A scalable algorithm exploiting ordered neighborhoods for cluster detection in data projections. - Exhaustive experiments showing that our method (1) works well with variable densities, (2) scales well with database size and dimensionality, (3) robust w.r.t. noise and irrelevant dimensions, (4) suitable for exploratory data analysis in real world scenarios.

Please find the supplementary for the paper, in which we cover the topics in detail, along with the implementation and datasets on our website.[3]

2 Formal Properties

We use the ordered local neighborhoods of objects in each individual dimension as indicators for clustered structures. This idea is based on the observation that similarity of objects is captured by the order of objects contained in the local neighborhood [17]. To the best of our knowledge, we are the first ones to exploit this theoretic principle for scalable cluster detection in high dimensional data.

[3] `http://adrem.uantwerpen.be/clon`

For our solution, objects only need to be sorted once according to the given (dis-)similarity measure. This might even be given (for free) due to the index structures maintained in the relational database system that stores the data. Based on these ordered neighborhoods, we detect clusters by mining object sets that re-occur in a similar order in multiple dimensions. Starting with one dimensional clusters and iteratively refining them allows us to detect clusters in projections (i.e. dimensions which agree in the similarity of objects) of a high dimensional data space.

Although the general idea seems simple to implement, it has several open research questions: Can we guarantee to detect all hidden clusters? How to capture the neighborhood information, and how can it be exploited for clustering high dimensional data? In order to answer all of these questions one by one, we structure our main contributions as follows: We first show that ordered neighborhoods ensure complete cluster detection in Section 2.1. Second, we propose a data transformation from relational data to an ordered neighborhood space and investigate its properties in Section 2.2. We introduce a scalable algorithm that exploits these properties for cluster detection in Section 3.

2.1 Clustering in Neighborhoods

For a *relational database* $\mathscr{D}$ we represent each object $\mathbf{o}$ in $\mathscr{D}$ as a vector defined over attributes $\mathscr{A} = \{A_1, A_2, \cdots, A_m\}$, with $\mathbf{o}_i$ the value of the object for the attribute A_i. For each attribute A_i, the projected database is denoted as $\mathscr{D}^{A_i}$. Further, we use $\delta(\mathbf{o}, \mathbf{p})$ as the dissimilarity between the objects $\mathbf{o}$ and $\mathbf{p}$, which can vary for each individual attribute. $|S|$ denotes the cardinality of set S.

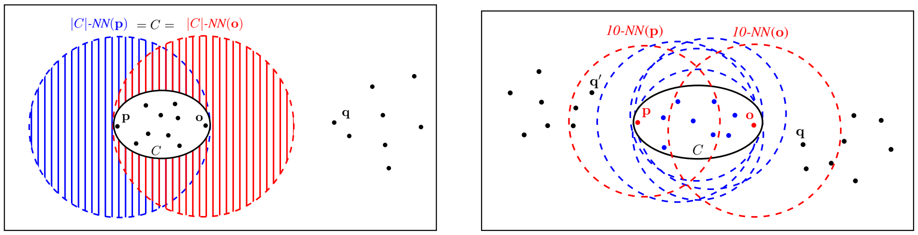
As basic notion we use local neighborhoods based on k-nearest neighborhood (*k-NN*), which have already been exploited for lazy classification [7], dimensionality reduction [21], collaborative filtering [16] and many other techniques in various communities [6,11]. For clustering, one has used shared nearest neighborhoods [8], neighborhoods in random projections [18], and frequent co-occurrence of neighborhoods [2]. In our work, we build the basic clustering principle of our method on the k-nearest neighborhood of an object, which is the set of objects that are the most similar to it:

Definition 1 **(k-Nearest Neighborhood (*k-NN*)).** *Let* $\mathbf{o} \in \mathscr{D}$, δ *a dissimilarity measure, and the* $NN_k(\mathbf{o})$ *be the* k^{th} *nearest object to* $\mathbf{o}$ *according to* δ, *the k-nearest neighborhood (k-NN) of* $\mathbf{o}$ *is defined as*

$$k\text{-}NN(\mathbf{o}) = \{\mathbf{p} \in \mathscr{D} \mid \delta(\mathbf{o}, \mathbf{p}) \leq \delta(\mathbf{o}, NN_k(\mathbf{o}))\}.$$

$k\text{-}NN(\mathbf{o})$ captures the similarities of objects near object $\mathbf{o}$, which we can use for cluster detection.

Figure 1 shows the nearest neighborhoods of the objects in clusters. Cluster sized nearest neighborhoods of objects in a *separable cluster* are equal to the cluster itself. In Figure 1a, $|C|$-*NN* of all 10 objects in C are equal the C itself. Although the *k-NN* of some of the objects in a non-separable cluster include objects that are not in the cluster, the *k-NN* of the majority of objects in the cluster include the whole cluster. For example in Figure 1b, the neighborhoods of the data points $\mathbf{p}$ and $\mathbf{q}$ are different from each other although they are in the same cluster. On the other hand, they are in the $|C|$-*NN* of the



(a) Separable cluster

(b) Non-separable cluster

Fig. 1: Nearest neighborhoods of the objects in a cluster

8 out of 10 objects in the cluster, shown as blue. Therefore, they can be identified as being in the same cluster by checking their co-occurrences in neighborhoods of other objects instead of comparing their neighborhoods. This is one of the advantages of using co-occurrences in neighborhoods instead of shared nearest neighborhoods.

2.2 Ordered Neighborhoods

In the original space, computing the co-occurrence of object sets in neighborhoods of objects in $\mathscr{D}$ requires expensive neighborhood computations. Instead, we compute the neighborhood of every object once and conduct co-occurrence search in these neighborhoods.

Definition 2 (Ordered Neighborhood). *The ordered neighborhood of* $\mathbf{o} \in \mathscr{D}^{A_i}$ *is the ordered list of the objects in* $k\text{-}NN(\mathbf{o})$*, in which the order reflects the order of objects in* A_i.

$$k\text{-}NN(\mathbf{o}) = (\cdots \mathbf{p}, \cdots \mathbf{q}, \cdots) \iff \mathbf{p} < \mathbf{q}$$

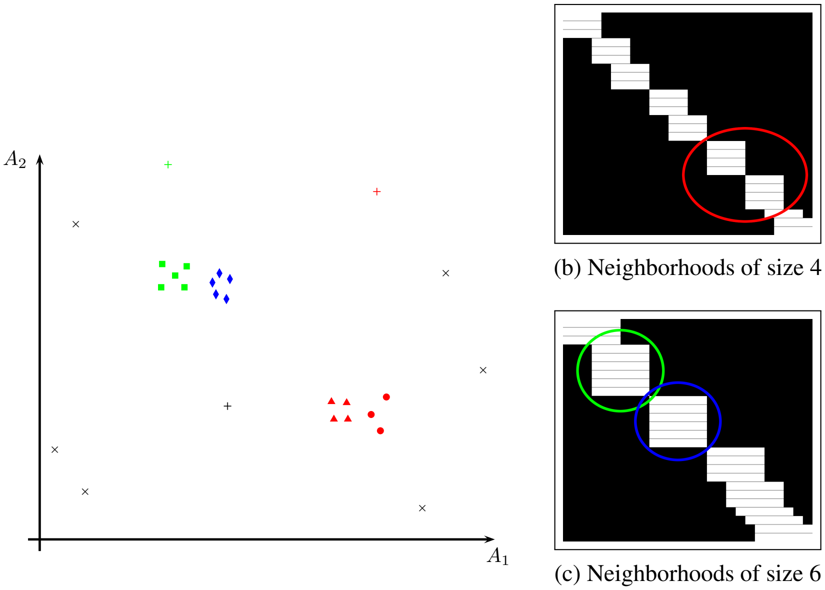
Definition 3 (Ordered Neighborhood Database). *An ordered neighborhood database* $\mathscr{N}$ *is the ordered list of ordered neighborhoods. Order of the neighborhoods is the order of objects in* $\mathscr{D}^{A_i}$*. If* $\mathbf{o} < \mathbf{p}$*, then*

$$\mathscr{N}^{A_i} = (\cdots k\text{-}NN(\mathbf{o}), \cdots k\text{-}NN(\mathbf{p}), \cdots)$$

For the remaining of the paper, we use *neighborhood* and $k\text{-}NN(\mathbf{o})$ to refer to the ordered neighborhood, and *neighborhood database* to refer to the ordered neighborhood database unless it is noted otherwise. Further, if the projection attribute is clear from the context or it is irrelevant, we drop the attribute and use $\mathscr{N}$ instead.

Let us look at the example dataset in Figure 2a. Figures 2b and 2c show $\mathscr{N}^{A_1}$ of the example dataset with $k = 4$ and $k = 6$. Each column in a neighborhood database represents an object while each row represents the neighborhood of an object. If an object occurs in a neighborhood, the corresponding cell is white. E.g., in Figure 2b each horizontal white line, which denotes 4-*NN*, consists of exactly 4 cells.

With a formal analysis, the following properties hold for the neighborhood DBs, their proofs are provided in Section S.1 (in Supplementary):



(b) Neighborhoods of size 4

(c) Neighborhoods of size 6

(a) An example dataset

Fig. 2: An example dataset and its neighborhood databases for different sizes

Property 1 (Consecutive Objects). If two objects are in a neighborhood, all the objects in between them are also in that neighborhood. Let $\mathbf{o},\mathbf{p},\mathbf{q},\mathbf{r} \in \mathscr{D}^{A_i}$ and $\mathbf{o} < \mathbf{p} < \mathbf{q}$. $\mathbf{o} \in k\text{-}NN(\mathbf{r}) \wedge \mathbf{q} \in k\text{-}NN(\mathbf{r}) \implies \mathbf{p} \in k\text{-}NN(\mathbf{r})$.

Property 2 (Consecutive Neighborhoods). If an object is in the neighborhood of two objects, then it is in the neighborhood of all the objects in-between them. Let $\mathbf{o},\mathbf{p},\mathbf{q},\mathbf{r} \in \mathscr{D}^{A_i}$ and $\mathbf{o} < \mathbf{p} < \mathbf{q}$. If $\mathbf{r} \in k\text{-}NN(\mathbf{o}) \wedge \mathbf{r} \in k\text{-}NN(\mathbf{q}) \implies \mathbf{r} \in k\text{-}NN(\mathbf{p})$.

We can see Properties 1 and 2 in Figure 2b. While the objects in a neighborhood are consecutive, an object only appears in consecutive neighborhoods, hence, they respectively form straight horizontal and vertical lines.

Theorem 1 (Recurrent Neighborhoods). *Clusters form square structures on the diagonal of the neighborhood databases.*

Proof. According to Properties 1 and 2, neighborhoods form continuous shapes. As we discuss in Section 2.1, a cluster is repeated as the neighborhoods of its objects, hence, the shape should be a square. Since each object is its own nearest neighbor, squares should be on the diagonal of the neighborhood DB. □

The two squares in the lower right quarter of the Figure 2b, which are circled in red, are the neighborhoods of the objects in the clusters that are denoted by ▲ and ●. Similarly, squares on the upper left quarter of Figure 2c, which are circled in green and blue, are respectively from the clusters 🟩, ♦.

Property 3 (Baseline). If there are no clusters in a segment of the dataset, i.e., the objects are distributed uniformly, all of the object sets for that segment have the same amount of recurrency in the neighborhood DB, which is a function of k, cf. Figure S.1.

Property 4 (Transitivity). If an object $\mathbf{o}$ is not in the neighborhood of $\mathbf{p} > \mathbf{o}$, then it is not in the neighborhood of any $\mathbf{q} > \mathbf{p}$. Formally let $\mathbf{o}, \mathbf{p}, \mathbf{q} \in \mathscr{D}^{A_i}$ and $\mathbf{o} < \mathbf{p} < \mathbf{q}$, $\mathbf{o} \notin k\text{-}NN(\mathbf{p}) \implies \mathbf{o} \notin k\text{-}NN(\mathbf{q})$.

In Figure 2b, we can see that the 6^{th} object occurs in neighborhoods 4 to 9. It does not occur in any neighborhood before 4 and not in any neighborhood after 9. We use the transitivity to limit the search space of our Algorithm, cf. Section 3, while the baseline property helps us to discard the segments without a cluster structure and to eliminate the artifacts that are caused by the transformation.

3 Mining the Neighborhoods

In this section we introduce our proposed algorithm CLON, which efficiently finds all subspace clusters by using ordered neighborhoods. A detailed explanation and pseudo code of the algorithm can be found in Section S.3 (in the supplementary).

If a cluster exists only in a subset of dimensions, the irrelevant dimensions hinder the search and the quality of the cluster. As we mentioned before, this is often the case in high dimensional spaces. To eliminate the negative effects of irrelevant dimensions, CLON follows a bottom-up strategy and exploits the fact that clusters in lower dimensional projections are supersets of clusters in higher dimensional spaces [9]. Hence, CLON finds the one dimensional clusters first and then iteratively refines them by evaluating their higher dimensional subsets.

In its first phase, CLON converts the relational database into a neighborhood database. Values in each dimension have to be sorted only once, after which the neighborhoods can be computed very fast by using a sliding window. A neighborhood DB is created for each of the dimensions. CLON already benefits from the ordered neighborhoods while creating the neighborhood DBs: It exploits the consecutiveness properties to store the neighborhood information so that instead of storing the whole neighborhoods, only the start and end of them are stored. This provides a dramatic memory saving, cf. Algorithm S.1.

In the second phase, the clusters in individual projections are found. As Theorem 1 states, clusters form recurrent object sets in the neighborhood DB. It has been proposed to find these recurrent objects sets by using frequent itemset mining methods, however, since these methods are computationally expensive, only approximate methods are feasible for non-trivial datasets [2]. Our method CLON exploits the properties of ordered neighborhoods, so that it can mine these recurrent object sets in linear time instead of exponential time (both in the number of objects). Thanks to this speed improvement, *all* maximally large objects sets that are more recurrent than a certain threshold can be found in a reasonable time, cf. Section 4.3. Therefore in this phase, CLON can identify all of the recurrent objects sets in all individual projections, that is all one dimensional clusters, cf. Algorithm S.2.

In the third and the last phase, one dimensional clusters are used to find higher dimensional clusters. For any higher dimensional cluster, there are clusters that have more or equal number of objects, in the subsets of the dimensions of the original cluster [2,9,10]. For example, if a cluster exists in dimensions 1, 2, and 3, then a superset of its objects form clusters in dimension pairs (1,2), (2,3) and (1,3). Therefore, we can find higher dimensional clusters by refining lower dimensional ones, i.e. removing the objects that are not in the cluster. CLON refines each one dimensional cluster by checking whether any of its subsets are recurrent in other dimensions. If there are any, CLON continues with a depth-first search of higher dimensional clusters by traversing all the possible dimensions until none of the subsets are large or recurrent enough. Since CLON uses neighborhood DBs also for higher dimensional search, properties of ordered neighborhoods are exploited during this phase too. Lastly, the recurrent object sets are output as clusters along with the dimensions that they are recurrent in, cf. Algorithm S.3.

CLON requires two user-friendly parameters: (1) Minimum size of a cluster and (2) the neighborhood size. Selecting the minimum size should be straightforward, it depends on the size of the cluster that the user would like to find. The neighborhood size should be larger than the cluster size. Because of the recurrency oriented mining, CLON is robust to these parameters. A detailed discussion about parameter selection can be found in Section S.2.

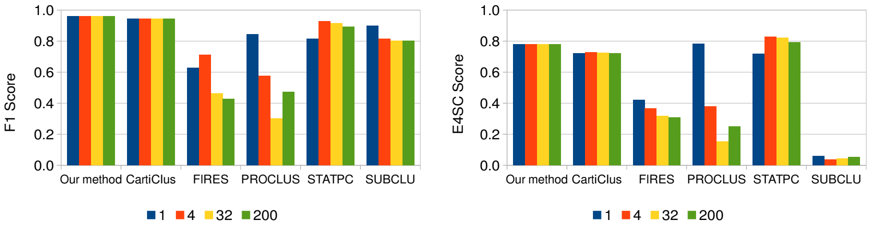
3.1 Complexity Analysis

For a numeric $\mathscr{D}$ that has n objects and d attributes, the computational complexity of transforming it to the neighborhood DB is $O(d \times n \times \log(n) \times k)$. Thanks to the consecutiveness properties we do not have to store the entire database but only the start and end of the neighborhoods are enough. Therefore, the space requirement for the neighborhood DBs is $O(n \times d)$ which is equivalent to the storage of the original $\mathscr{D}$. Exploiting the properties of the neighborhood DB allows us to find one dimensional clusters extremely efficiently, in $O(n)$. Therefore, the complexity of finding all one dimensional clusters is $O(d \times n)$. Total time to find all subspace clusters depends on the dimensionality of the clusters in the data. Although the computational complexity to search a cluster in a dimension is linear, the number of dimensions to search is combinatorial. Empirical results in Section 4.3 show that the scalability and efficiency of CLON is not hindered by the dimensionality of clusters.

4 Empirical Evaluation

4.1 Experimental Setup

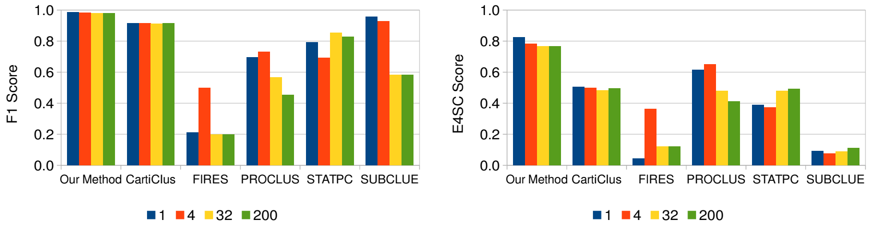
We evaluate our method on heterogeneous datasets, i.e., datasets with different scales, both in terms of dimension scale and cluster distribution. We also conduct a set of experiments to evaluate the scalability of our method according to the number of objects, number of dimensions and the amount of noise. To see whether our method is a good fit for real world scenarios, we conduct experiments on very high dimensional real world



(a) F1 Scores

(b) E4SC Scores

Fig. 3: Quality scores of the methods on datasets with dimension of various scales



(a) F1 Scores

(b) E4SC Scores

Fig. 4: Quality scores of methods on datasets that have clusters of various scales

datasets. We compare the quality of the clusters that are found by our method with the state of the art subspace clustering methods, such as CartiClus [2], FIRES [10], PROCLUS [1], STATPC [12], and SUBCLU [9], cf. Section S.4.

Object cluster finding capabilities of the methods are evaluated by the supervised F1 Measure. Where appropriate, we evaluate the subspace discovery capabilities by using the established E4SC score [14]. Runtime results are given for the performance versus scale experiments. More information about the experimental setup can be found in Section S.5. For the reproducibility of the experiments, a cross-platform implementation of the method and the information about the datasets are available on our website.

4.2 Heterogeneous Datasets

We measure the cluster discovery capabilities of our method on datasets that (1) have different scaling in different dimensions and (2) have clusters with different spreads. To evaluate these capabilities we generate two sets of datasets: One with different scales of dimensions and another one with clusters of different scales. Details about these datasets can be found in Section S.5.

Figures 3a and 4a show the F1 scores of the methods on a selected subset of these datasets. Different scale factors are indicated with different colors. On both set of experiments, CLON produces very stable and high quality clusters. CartiClus is comparable with CLON, which shows the effectiveness of nearest neighborhood based clustering. Radius based neighborhood evaluation of FIRES and distance sensitive projections of PROCLUS cannot cope with scaling. SUBCLU and STATPC produce high quality, although unstable, results thanks to their thorough search. Considering the execution times of CLON and its closest competitors, cf. Figure 5, we can see the advantages of cluster refining through dimension search in ordered neighborhoods: CLON consistently produces better or comparable clusters in an order of magnitude less time.

Figure 3b and Figure 4b show the E4SC scores for the same sets of datasets. While some of the methods perform poorly on dimension detection, dimension finding capabilities of the most of them are in parallel with the object cluster finding capabilities. CLON performs better than CartiClus in dimension detection although they both mine recurrent object sets, cf. Figure 4b. This is because CLON is fast enough to mine all subspace clusters instead of a sample of them.

4.3 Scalability Results

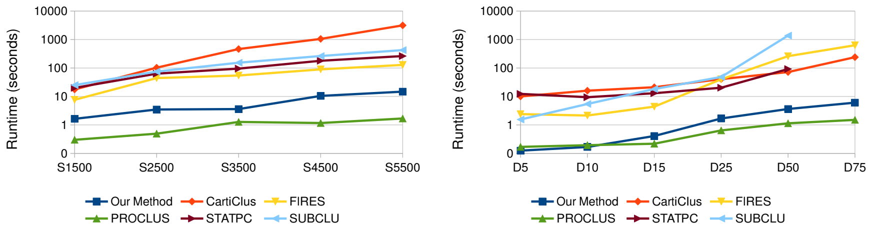
We conduct experiments to understand the scalability of our algorithm according to the data size, dimensionality, and the noise ratio. For these experiments we use the datasets that are used in the literature for similar comparative studies [2,14].

Figure 5a shows the run times of the methods on datasets with an increasing number of objects. This set of datasets include 5 different datasets which have approximately 1500, 2500, 3500, 4500, and 5500 objects and 20 dimensions. Our method scales well compared to the algorithms that search the combinations of dimensions.

Figure 5b shows the run times for different methods on datasets with an increasing number of dimensions. We conduct experiments on 6 different datasets that have approximately 1500 objects in 5, 10, 15, 25, 50, and 75 dimensions. In all of these datasets, each cluster exists in approximately 80% of the dimensions. Therefore, these experiments also evaluate the performance of the algorithms regarding dimensionality of clusters. Our method scales well with the increasing number of dimensions while utilizing the information in combinations of dimensions. Note that the missing values indicates that the method did not complete in a practical time or failed because of excessive memory requirements.

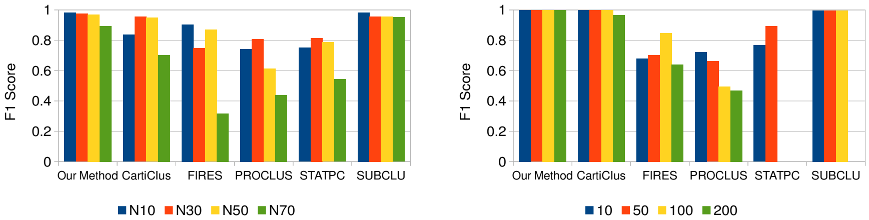
Figure 6a shows the quality results for the datasets that contain 10%, 30%, 50%, and 70% noise. Our method and CartiClus can effectively discard noise thanks to their recurrency and neighborhood based foundation. Only SUBCLU is on par with these methods because of its thorough search which also makes it slower around 100 times than our method, cf. Figure 5.

Figure 6b shows the capability of irrelevant dimension detection. A dataset which has 10 clusters hidden in subsets of 10 dimensions is generated. Then, 10, 50, 100 and 200 uniformly randomly generated dimensions are added to the dataset. Here again, we see that our method, CartiClus and SUBCLU can effectively discard irrelevant dimensions even though the meaningful structures are hidden in noise that is 20 times of its size. Some of the methods did not complete in meaningful time for some of the datasets, including SUBCLU for more than 100 dimensions.



(a) Increasing the number of objects (b) Increasing the number of dimensions

Fig. 5: Execution times



(a) Increasing the number of noise objects (b) Increasing the number of irrelevant dimensions

Fig. 6: Noise detection capability experiments

4.4 Real World Datasets

To evaluate our method, we used two gene expression datasets, *Nutt* and *Alon*, along with a movie rating dataset, *movies*. *Alon* is a dataset of 2000 gene expression across 62 tissues from a colon cancer research. The tissues are grouped into 2 categories: 40 healthy tissues and 22 tissues with tumor [3]. *Nutt* dataset contains expressions of 1377 genes on 50 glioma tissue samples that are grouped into 4 different pathology categories [15]. High dimensional nature of both datasets make the clustering challenging even for the subspace clustering methods. Table 1 shows the F1 scores of the methods. "n/a" indicates that the method did not complete in a practical time or failed because of excessive memory requirements. These results show that, although our method searches the clusters in combinations of dimensions, it keeps being scalable even for very high dimensional datasets.

We conduct an exploratory data analysis on a movie ratings dataset from GroupLens.[4] We use 10M movielens dataset which includes 10000054 ratings applied to 10681 movies by 71567 users from the website `http://movielens.org`. We use

[4] `http://grouplens.org/`

	Alon	*Nutt*
Our method	**0.78**	**0.75**
PROCLUS	0.46	0.44
FIRES	0.52	0.55
SUBCLU	0.58	n/a
STATPC	n/a	n/a
CartiClus	n/a	n/a

Table 1: F1 scores for gene expression datasets

movies as objects and users as attributes. We start with finding the most similar 3 movies according to the user ratings. The result is, not surprisingly, the original Star Wars trilogy. When we lower the similarity threshold, we start to see a cluster of the Lord of the Rings Trilogy along with some other high profile movies in clusters of science fiction movies, action movies and crime movies. Some of the selected clusters are given in Table S.4a.

We continue our analysis by increasing the number of similar movies to 6. Considering the similarity of the original Star Wars trilogy, we search for a cluster of all 6 of the released Star Wars movies, with no luck. Actually, lack of this 6 pack of Star Wars movies cluster does not contradict with the general opinion of the fans of the franchise because the last 3 movies are not as popular as the originals. Table S.4b shows some of the interesting clusters of size 6, which includes a cluster of two most popular trilogies, a cluster of distopian movies, a cluster of popular thrillers, and a cluster of very popular classic movies.

5 Conclusion

In this paper, we tackle the problem of finding clusters in high dimensional databases. We make a formal analysis of ordered neighborhoods and their intrinsic properties. We show that co-occurrences in local neighborhoods of objects are indicators of cluster formations, and hence, clusters can be found by mining recurrent neighborhoods. We propose a scalable algorithm that exploits these intrinsic properties to find all of the clusters and their relevant dimensions. Along with its scalability, key properties of our algorithm include its intuitive and user friendly parameters, its noise detection capabilities, its adaptivity to datasets with various scales, and its capability to exploit multiple (dis-)similarity measures.

Besides conducting experiments on scalability, noise detection, and adaptivity; we tested our method on some very high dimensional gene expression datasets. Further, we did an exploratory analysis on a movie rating dataset to show that our method is a good fit for real world scenarios. Finally, we supply a software tool that can be used to interpret the data for further analysis.

References

1. Aggarwal, C.C., Wolf, J.L., Yu, P.S., Procopiuc, C., Park, J.S.: Fast algorithms for projected clustering. SIGMOD Rec. 28(2), 61–72 (Jun 1999)
2. Aksehirli, E., Goethals, B., Müller, E., Vreeken, J.: Cartification: A neighborhood preserving transformation for mining high dimensional data. In: ICDM. pp. 937–942 (Dec 2013)
3. Alon, U., Barkai, N., Notterman, D.A., Gish, K., Ybarra, S., Mack, D., Levine, A.J.: Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. PNAS 96(12), 6745–6750 (Jun 1999)
4. Assent, I., Krieger, R., Müller, E., Seidl, T.: INSCY: Indexing subspace clusters with in-process-removal of redundancy. In: ICDM. pp. 719–724 (Dec 2008)
5. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: ICDT, pp. 217–235. Springer (1999)
6. Emrich, T., Kriegel, H.P., Mamoulis, N., Niedermayer, J., Renz, M., Zfle, A.: Reverse-nearest neighbor queries on uncertain moving object trajectories. In: DASFAA, vol. 8422, pp. 92–107 (2014)
7. Goldstein, M.: k_n-nearest neighbor classification. IEEE TIT 18(5), 627–630 (Sep 1972)
8. Jarvis, R., Patrick, E.: Clustering using a similarity measure based on shared near neighbors. IEEE TC C-22(11), 1025 – 1034 (Nov 1973)
9. Kailing, K., Kriegel, H.P., Kröger, P.: Density-connected subspace clustering for high-dimensional data. In: SDM. vol. 4. SIAM (2004)
10. Kriegel, H.P., Kröger, P., Renz, M., Wurst, S.: A generic framework for efficient subspace clustering of high-dimensional data. In: ICDM. pp. 8 pp.– (Nov 2005)
11. McCann, S., Lowe, D.: Local naive bayes nearest neighbor for image classification. In: CVPR. pp. 3650–3656 (June 2012)
12. Moise, G., Sander, J.: Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering. In: KDD. pp. 533–541. KDD '08, ACM, New York, NY, USA (2008)
13. Müller, E., Assent, I., Günnemann, S., Krieger, R., Seidl, T.: Relevant subspace clustering: Mining the most interesting non-redundant concepts in high dimensional data. In: ICDM. pp. 377–386 (Dec 2009)
14. Müller, E., Günnemann, S., Assent, I., Seidl, T.: Evaluating clustering in subspace projections of high dimensional data. PVLDB 2(1), 1270–1281 (2009)
15. Nutt, C.L., Mani, D.R., Betensky, R.A., Tamayo, P., Cairncross, J.G., Ladd, C., Pohl, U., Hartmann, C., McLaughlin, M.E., Batchelor, T.T., Black, P.M., Deimling, A.v., Pomeroy, S.L., Golub, T.R., Louis, D.N.: Gene expression-based classification of malignant gliomas correlates better with survival than histological classification. Cancer Res. 63(7), 1602–1607 (Apr 2003)
16. Park, Y., Park, S., Jung, W., goo Lee, S.: Reversed cf: A fast collaborative filtering algorithm using a k-nearest neighbor graph. Expert Syst. Appl. 42(8), 4022 – 4028 (2015)
17. Rodriguez, A., Laio, A.: Clustering by fast search and find of density peaks. Science 344(6191), 1492–1496 (Jun 2014)
18. Schneider, J., Vlachos, M.: Fast parameterless density-based clustering via random projections. In: CIKM. pp. 861–866. CIKM '13, ACM, New York, NY, USA (2013)
19. Sequeira, K., Zaki, M.: SCHISM: A new approach for interesting subspace mining. In: ICDM. vol. 0, pp. 186–193. IEEE Computer Society (2004)
20. Sim, K., Gopalkrishnan, V., Zimek, A., Cong, G.: A survey on enhanced subspace clustering. Data Min. Knowl. Disc. 26(2), 332–397 (Mar 2013)
21. Weinberger, K., Saul, L.: Unsupervised learning of image manifolds by semidefinite programming. Int. J. Computer Vision 70(1), 77–90 (2006)

Supplemental Materials: Detecting Cluster Structures by Ordered Neighborhoods

S.1 Proofs

Property 1 (Consecutive Objects). If two objects are in a neighborhood, all the objects in between them are also in that neighborhood. Let $\mathbf{o},\mathbf{p},\mathbf{q},\mathbf{r} \in \mathscr{D}^{A_i}$ and $\mathbf{o} \leq \mathbf{p} \leq \mathbf{q}$. $\mathbf{o} \in k\text{-}NN(\mathbf{r}) \wedge \mathbf{q} \in k\text{-}NN(\mathbf{r}) \implies \mathbf{p} \in k\text{-}NN(\mathbf{r})$.

Proof (of Property 1). Our proof is in two complementary parts, $x \in \mathscr{D}^{A_i}$:
1. If $\mathbf{r} < \mathbf{p} < \mathbf{q}$, then $|\{x \mid \mathbf{r} \leq x \leq \mathbf{p}\}| < |\{x \mid \mathbf{r} \leq x \leq \mathbf{q}\}| \leq k$. Which means $\mathbf{p} \in m\text{-}NN(\mathbf{r})$ where $m < k$, thus $\mathbf{p} \in k\text{-}NN(\mathbf{r})$.
2. If $\mathbf{o} < \mathbf{p} < \mathbf{r}$, then $|\{x \mid \mathbf{p} \leq x \leq \mathbf{r}\}| < |\{x \mid \mathbf{o} \leq x \leq \mathbf{r}\}| \leq k$. Which means $\mathbf{p} \in m\text{-}NN(\mathbf{r})$ where $m < k$, thus $\mathbf{p} \in k\text{-}NN(\mathbf{r})$. □

Property 2 (Consecutive Neighborhoods). If an object is in the neighborhood of two objects, then it is in the neighborhood of all the objects in-between them. Let $\mathbf{o},\mathbf{p},\mathbf{q},\mathbf{r} \in \mathscr{D}^{A_i}$ and $\mathbf{o} \leq \mathbf{p} \leq \mathbf{q}$. If $\mathbf{r} \in k\text{-}NN(\mathbf{o}) \wedge \mathbf{r} \in k\text{-}NN(\mathbf{q}) \implies \mathbf{r} \in k\text{-}NN(\mathbf{p})$.

Proof (of Property 2). Our proof is in two complementary parts, $x \in \mathscr{D}^{A_i}$:
1. If $\mathbf{r} < \mathbf{p} < \mathbf{q}$, then $|\{x \mid \mathbf{r} \leq x \leq \mathbf{p}\}| < |\{x \mid \mathbf{r} \leq x \leq \mathbf{q}\}| \leq k$. Which means $\mathbf{r} \in m\text{-}NN(\mathbf{p})$ where $m < k$, thus $\mathbf{r} \in k\text{-}NN(\mathbf{p})$.
2. If $\mathbf{o} < \mathbf{p} < \mathbf{r}$, then $|\{x \mid \mathbf{p} \leq x \leq \mathbf{r}\}| < |\{x \mid \mathbf{o} \leq x \leq \mathbf{r}\}| \leq k$. Which means $\mathbf{r} \in m\text{-}NN(\mathbf{p})$ where $m < k$, thus $\mathbf{r} \in k\text{-}NN(\mathbf{p})$. □

Property 3 (Baseline). If there are no clusters in a segment of the dataset, i.e., the objects are distributed uniformly, all of the object sets for that segment have the same amount of recurrency, which is a function of k, cf. Figure S.1. Therefore, the segments without a cluster structure can easily be discarded.

Proof (of Property 3). Let us represent the objects as $\{\mathbf{o}^0,\mathbf{o}^1,\cdots,\mathbf{o}^i,\cdots\} = \mathscr{D}$. Since the data is uniformly distributed:

$$\cdots = \delta(\mathbf{o}^{i-1},\mathbf{o}^i) = \delta(\mathbf{o}^i,\mathbf{o}^{i+1}) = \delta(\mathbf{o}^{i+1},\mathbf{o}^{i+2})\cdots$$

Without loss of generality, let us assume that k is an odd number and $m = (k-1)/2$. For $m < i < |\mathscr{D}|-m$, $\delta(\mathbf{o}^i,\mathbf{o}^{i-m}) = \delta(\mathbf{o}^i,\mathbf{o}^{i+m})$, and therefore $k\text{-}NN(\mathbf{o}^i) = \{\mathbf{o}^{i-m},\mathbf{o}^{i-m+1},\cdots,\mathbf{o}^{i+m}\}$. Recurrency of a single object is a function of m, hence k, because $\mathbf{o}^i \in k\text{-}NN(\mathbf{o}^j)$ iff $i-m \leq j \leq i+m$. Similarly, recurrency of an object set depends on k: $\{\mathbf{o}^i,\mathbf{o}^{i+1},\cdots,\mathbf{o}^{i+m}\} \in k\text{-}NN(\mathbf{o}^j)$ iff $i-\lfloor m/2 \rfloor \leq j \leq i+\lceil m/2 \rceil$. It can be shown in a similar way for the case of k is an even number. □

Property 4 (Transitivity). Let $\mathbf{o},\mathbf{p},\mathbf{q} \in \mathscr{D}^{A_i}$ and $\mathbf{o} \leq \mathbf{p} \leq \mathbf{q}$. $\mathbf{o} \notin k\text{-}NN(\mathbf{p}) \implies \mathbf{o} \notin k\text{-}NN(\mathbf{q})$.

Proof (of Property 4). By definition, $\mathbf{o} \in k\text{-}NN(\mathbf{o})$. If $\mathbf{o} \in k\text{-}NN(\mathbf{q})$, then $\mathbf{o} \in k\text{-}NN(\mathbf{p})$ because of Property 2 ($\mathbf{o} \leq \mathbf{p} \leq \mathbf{q}$). Thus, $\mathbf{o}$ cannot be a neighbor of $\mathbf{q}$ if it is not a neighbor of $\mathbf{p}$. □

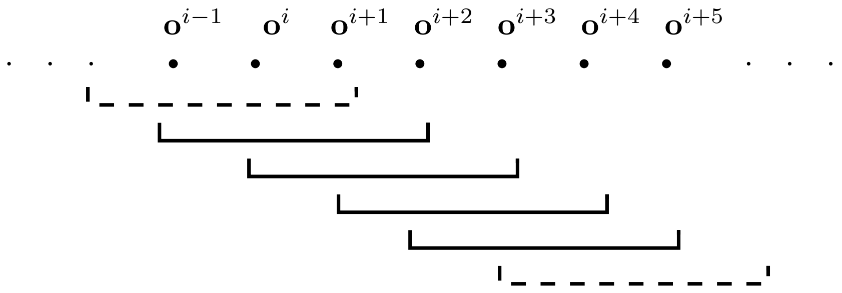


Fig. S.1: Neighborhoods of a uniformly distributed data

S.2 Parameter Selection

CLON requires two user-friendly parameters: (1) Minimum size of a cluster and (2) the neighborhood size. Selecting the minimum size should be straightforward. It depends on the size of the cluster that the user would like to find. Unlike the parameters such as density or significance interval, minimum cluster size is fairly easy to decide.

Obviously, the basic rule for the neighborhood size is: it should be greater than the cluster size. The question is: how much? The answer to this question depends on the characteristics, and in particular the noise ratio, of the data. It should be large enough to cover the expected cluster size *and* the noise on the projected dimension. In Figure 2a the points marked with $+$ are noise for projection, and to find the clusters $C_{\blacksquare}$ and $C_{\blacklozenge}$, the neighborhood size could be 6, cf. Figure 2c.

In favor of the robustness of our method, the neighborhood size does not have to be exact. If the neighborhood size is larger than the cluster size, then the neighborhoods still include all objects from the cluster *in addition to* the objects that are not in the cluster. Formally, for $\exists \mathbf{r} \in C$,

$$k \geq |C| : \forall \mathbf{o}, \mathbf{p} \in C \implies \mathbf{o}, \mathbf{p} \in k\text{-}NN(\mathbf{r})$$

If the noise or the other clusters are uniformly distributed around a cluster C, then the neighborhoods of the objects in the cluster do not include the same set of objects. Therefore, the recurrency of the objects in the cluster stays higher than the objects that are not in the cluster. For example, when we increase the neighborhood size in Figure 1b, only some of the objects in C include $\mathbf{q}$ in their neighborhoods while some others include $\mathbf{q}'$. Since the objects in the cluster, such as $\mathbf{p}$ and $\mathbf{o}$, will be included in all neighborhoods, their recurrency will stay higher than $\mathbf{q}$ and $\mathbf{q}'$.

If increasing the neighborhood size increases the recurrency for a set of objects, it is possible that the cluster under consideration is part of a larger cluster. Moreover, if the neighborhood size is smaller than the cluster size, i.e. $k < |C|$, then a subset of the cluster becomes recurrent in the neighborhoods. For example $C_{\blacksquare}$ is partitioned into two little clusters in Figure 2b but they are merged again when we increase the neighborhood size, cf. Figure 2c. In summary, the neighborhood size does not have to be exact, if it is within an interval around the cluster size, we can still find the exact cluster or the core of the cluster.

Lastly, neighborhood databases are visually interpretable. We provide a software tool with a graphical user interface that shows the neighborhood DB so that the neighborhood size that produce the most recurrency can be detected, cf. Section 4.

S.3 Algorithms

First step of the algorithm is to convert relational database into the neighborhood databases. Algorithm S.1 shows the transformation. Neighborhood databases are created for each of the projections (dimensions). The neighborhood databases are in horizontal format, meaning that they are composed of the neighborhoods. However, the consequent steps of the algorithm need the data in vertical format for efficient computation, which is composed of objects and the neighborhoods that they occur. Since both the objects and the neighborhoods are continuous, according to the Properties 1 and 2, we only require the starting neighborhood, that is the id of the neighborhood that the object first appears, and the end neighborhood, that is the id of the neighborhood that the object appears for the last time. To avoid confusion, we make the following definition: *item* is an associative array of an object reference *id*, starting neighborhood id *txS* and the end neighborhood id *txE*.

Algorithm S.1 Create neighborhood databases

```
Input: $\mathscr{D}$: Real valued high dimensional database, $k$: Neighborhood size
Output: $\mathscr{T}$: Neighborhood databases
$\mathscr{T} \leftarrow []$
for each $\mathscr{D}^{A_i} \in \mathscr{D}$ do
    sort $\mathscr{D}^{A_i}$ in ascending order
    $\mathscr{N}^{A_i} \leftarrow []$
    for each $\mathbf{o} \in \mathscr{D}^{A_i}$ do
        append $k\text{-}NN(\mathbf{o}_i)$ to $\mathscr{N}^{A_i}$
    $\mathscr{T}_i \leftarrow \mathscr{N}^{A_i}$
return
```

Algorithm S.2 shows the MaxiBand algorithm, our efficient maximal recurrent object set miner that we use to find one dimensional clusters. Since each subset of a cluster is also a cluster, we search for maximally recurrent object sets to minimize the redundant clusters.

MaxiBand takes a sorted list of items $\mathscr{I}$ and a minimum length λ as input. λ parameter is used to determine how large and recurrent a set of objects should be. Note that, since the clusters form square structures, only one parameter is enough. To compute in how many neighborhoods a set of objects occur together, we use the support function σ, which returns the number of neighborhoods a set of objects occur.

MaxiBand scans through a neighborhood DB with an elastic frame to find all of the maximal recurrent object sets. It starts with creating an object set from the first λ

objects (lines 2–3), and it is extended until non of its supersets are recurrent (lines 6–7). If the object set itself is recurrent, then it is added to the list of maximal recurrent object sets (lines 8–9). Since the order of the objects is fixed and the object sets are always continuous we only keep the first and last objects. If the first object of the shifted frame does not add any additional neighborhoods, it skips the object (lines 11–12). It shifts frame by 1 item (lines 13–15) and continues until it scans the whole database (line 5).

Algorithm S.2 MaxiBand: Maximal frequent itemset miner

```
Input: $\mathscr{I}$: sorted items, $\lambda$: minumum size
Output: $\mathscr{F}$: maximal recurrent object sets
 1: $\mathscr{F} \leftarrow \emptyset$
 2: $s \leftarrow 0$  // Index of the first item of the itemset
 3: $e \leftarrow s + \lambda$  // Index of the last item of the itemset
 4: $\sigma : \mathscr{I}^2 \rightarrow \mathbb{N}^+, \sigma(i_a, i_b) = i_a.txE - i_b.txS$
 5: while $e < len(\mathscr{I})$ do
 6:     while $\sigma(i_s, i_{e+1}) > \lambda$ do
 7:         $e \leftarrow e + 1$
 8:     if $\sigma(i_s, i_e) > \lambda$ then
 9:         $\mathscr{F} \leftarrow \mathscr{F} \cup (i_s, i_e)$
10:         $e \leftarrow e + 1$
11:     while $i_{s+1}.txE = i_s.txE$ do
12:         $s \leftarrow s + 1$
13:     $s \leftarrow s + 1$
14:     if $e - s < \lambda$ then
15:         $e \leftarrow s + \lambda$
```

Algorithm S.3 shows the CLON algorithm for finding all the subspace clusters. It takes a numeric high dimensional database $\mathscr{D}$, a cluster size λ, and a neighborhood size k as inputs and outputs all the subspace clusters in $\mathscr{D}$. A subspace cluster is represented as a tuple, composed of a set of objects and a set of relevant dimensions. The algorithm starts with converting the $\mathscr{D}$ into the neighborhood DBs (line 4). Then, the maximal recurrent objects sets in each neighborhood DB is found (line 5–6). Note that, instead of directly passing λ as an argument to MaxiBand, we pass the maximum of λ and $k \times 0.6$. Because the baseline property, Property 3, states that there is always a minimum amount of recurrency in a neighborhood DB although there are no cluster formations. Computing this value reveals that if there are no cluster structures in the data, almost all of the neighborhoods satisfy $\lambda = k \times 0.5$. If the λ value is lower than this value, then there will be too many false positive clusters in the output. Therefore, we empirically select $k \times 0.6$ as the minimum recurrency threshold to eliminate the insignificant recurrencies. Further discussion of the parameter selection is in Section S.2.

Each recurrent object set in an individual neighborhood DB is a 1 dimensional cluster (line 8), which can possibly be a superset of a cluster in a higher dimensional space. Therefore, to further process these clusters, we keep them along with their known dimensions and possible extensions (line 9). We keep them in a stack to employ a deep-first search and thus minimize the memory requirement.

Algorithm S.3 CLON Algorithm

```
Input: $\mathscr{D}$: Numeric high dimensional database, $\lambda$: Cluster size, $k$: Neighborhood size
Output: $\mathscr{C}$: Subspace clusters
 1: $\mathscr{C} \leftarrow \emptyset$
 2: initialize empty stack $\mathscr{S}$
 3: $A = \{1, 2, \cdots, |\mathscr{A}|\}$
 4: $\mathscr{T} \leftarrow$ Convert $\mathscr{D}$ to neighborhood DBs
 5: for each $\mathscr{N}^{A_i} \in \mathscr{T}$ do
 6:     $F \leftarrow$ MaxiBand $(\mathscr{I} \in \mathscr{N}^{A_i}, \max(\lambda, k \times 0.6))$
 7:     for each $f \in F$ do
 8:         $\mathscr{C} \leftarrow \mathscr{C} \cup (f, \{i\})$
 9:         push $(f, \{i\}, A - \{i\})$ to $\mathscr{S}$
10: repeat
11:     $(f, D, P) \leftarrow$ pop $\mathscr{S}$
12:     for each $p \in P$ do
13:         $I' \leftarrow$ get ordered items of $f$ from $\mathscr{N}^{A_p}$
14:         $F \leftarrow$ MaxiBand $(I', \lambda)$
15:         for each $f' \in F$ do
16:             $\mathscr{C} \leftarrow \mathscr{C} \cup (f, D \cup \{p\})$
17:             push $(f, D \cup \{p\}, P - \{p\})$ to $\mathscr{S}$
18: until $\mathscr{S}$ is empty
```

Second phase of the algorithm refines the clusters by checking their possible extensions in higher dimensions (lines 10–18). Search space within a neighborhood DB of a possible dimension extension is limited to the objects in the cluster (line 13–14). Each maximal recurrent object set that is a subset of these objects is a cluster in this dimension in addition to the dimensions of the starting cluster (line 16). Newly found clusters are added both to the found cluster list and to the stack for further refinement (lines 15–17).

S.4 State of the Art

In this section we review some of the subspace clustering techniques related to our approach. For a good overview on subspace clustering techniques readers are kindly referred to surveys [20] and comparative analysis articles [14] in the literature. We do not include any of the traditional clustering methods because it has been shown that they cannot cope with the subspace clusters in high dimensional spaces [2].

PROCLUS [1] is a widely used projected clustering algorithm that produces very good results despite its simplicity. It starts with a random seed selection and iteratively assigns points to clusters and dimensions. It is very fast but it is sensitive to noise and it cannot find overlapping clusters [14].

FIRES [10] starts by identifying the density based clusters in 1 dimensional projections. Then, it prunes the search space by using the information from these 1D clusters and finally, detects clusters in this pruned search space. The projection and density based approach of FIRES is similar to our method. FIRES uses ε-neighborhood based density which cannot cope well with variable density databases. Moreover, the parameters of FIRES are difficult to set. For example, the implementation in the OpenSubspace package [14] requires 9 different parameters, none of which is trivial to set.

CartiClus [2] uses a neighborhood transformation that is similar to that we use and employs frequent itemset mining methods to find subspace clusters. However, as FIM methods are combinatorial in the number of objects and since it does not exploit the ordering of the objects as we do here, mining for all clusters becomes intractable. Therefore, it uses a stochastic method and can detect only a subset of clusters.

SUBCLU [9] is a density based subspace clustering algorithm. It starts by detecting clusters in 1 dimensional projections and then it iteratively extends the search space by combining dimensions. At each iteration, it prunes the search space by using the available cluster information. It finds all of the clusters in each combination of dimensions. However, not only its density approach cannot cope well with varying density datasets but also its subset-based pruning strategy does not scale.

STATPC [12] searches statistically significant areas in axis-parallel regions. It approaches the problem of finding significant clusters as an optimization problem and selects a representative sample instead of doing an exhaustive search. Although STATPC produces satisfactory results, its approximative nature prevents the algorithm from producing stable results, cf. Section 4.

S.5 Experimental Evaluation

The F1 score is the harmonic mean of *precision* and *recall*. *precision* between a found cluster C_i and a true cluster $\mathscr{C}_j$ is $\frac{|C_i \cap \mathscr{C}_j|}{|C_i|}$, and the *recall* is $\frac{|C_i \cap \mathscr{C}_j|}{|\mathscr{C}_j|}$. Following the practice in literature [12,14], we match the found clusters to the true clusters that has the largest common points.

For CartiClus, we use the implementation from the authors [2] and for the other methods we use OpenSubspace package [14]. Our method is implemented in Java. All the experiments are run on a computer with Intel i7 CPU and 8 GB of memory running with a GNU/Linux operating system. Since CartiClus and PROCLUS involve stochastic processes, we run them multiple times and report the avarage reults.

Creating the heterogeneous datasets: (1) After creating a dataset with 8 clusters that are hidden in a total of 12 dimensions, half of these dimensions are scaled with factors of 2, 4, 8, 16, 32, 100, and 200. (2) Two equivalent clustering formations that have 4 clusters hidden in 6 dimensions are created. One of these datasets is scaled with factors of 2, 4, 8, 16, 32, 100, and 200, then merged with the other one along with some uniformly random dimensions.

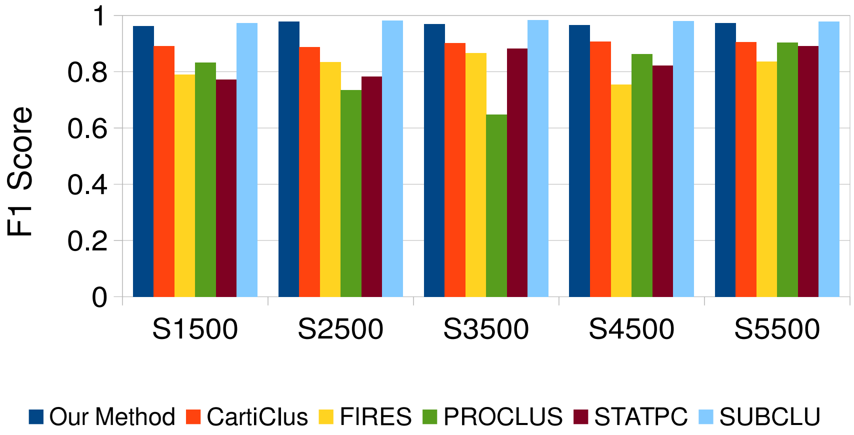


Fig. S.2: F1 Scores for the datasets with increasing number of objects

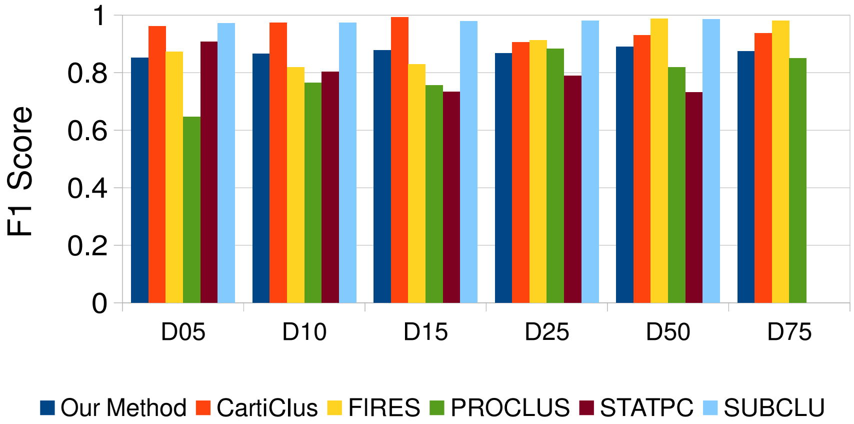


Fig. S.3: F1 Scores for the datasets with increasing number of dimensions

Star Wars: A New Hope (1977) Star Wars: The Empire Strikes Back (1980) Star Wars: Return of the Jedi (1983)
LotR: The Fellowship of the Ring, The (2001) LotR: The Two Towers, The (2002) LotR: The Return of the King, The (2003)
Back to the Future (1985) Terminator, The (1984) Terminator 2: Judgment Day (1991)
Die Hard (1988) Terminator, The (1984) Terminator 2: Judgment Day (1991)
Usual Suspects, The (1995) Pulp Fiction (1994) Silence of the Lambs, The (1991)

(a) Clusters of size 3

Star Wars: A New Hope (1977) Star Wars: The Empire Strikes Back (1980) Star Wars: Return of the Jedi (1983) LotR: The Fellowship of the Ring, The (2001) LotR: The Two Towers, The (2002) LotR: The Return of the King, The (2003)
Brazil (1985) Dr. Strangelove (1964) Clockwork Orange, A (1971) 2001: A Space Odyssey (1968) Blade Runner (1982) Alien (1979)
Chinatown (1974) Rear Window (1954) North by Northwest (1959) Vertigo (1958) Psycho (1960) Silence of the Lambs, The (1991)
Third Man, The (1949) Citizen Kane (1941) Godfather: Part II, The (1974) Chinatown (1974) Godfather, The (1972) Taxi Driver (1976)

(b) Clusters of size 6

Fig. S.4: Clusters of Movies