Finding Subspace Clusters using Ranked Neighborhoods

Emin Aksehirli[•], Siegfried Nijssen[○,*], Matthijs van Leeuwen[○,*] and Bart Goethals[•]
[•]University of Antwerp, Belgium; email: firstname.lastname@uantwerpen.be
[○]Department of Computer Science, KU Leuven, Belgium; email: firstname.lastname@cs.kuleuven.be
[*]Leiden Institute for Advanced Computer Science, University of Leiden, The Netherlands

***Abstract*—Clustering high dimensional datasets is challenging due to the curse of dimensionality. One approach to address this challenge is to search for *subspace clusters*, i.e., clusters present in subsets of attributes. Recently the *cartification* algorithm was proposed to find such subspace clusters. The distinguishing feature of this algorithm is that it operates on a neighborhood database, in which for every object only the identities of the k closest objects are stored. Cartification was shown to produce better results than other state-of-the-art subspace clustering algorithms; however, which clusters it detects was also found to depend heavily on the setting of the parameters. In other words, it is not robust to input parameters.**

In this paper, we propose a new approach called *ranked cartification* that produces more robust results than ordinary cartification. We develop a transformation that creates *ranked matrices* instead of neighborhood databases; we identify clusters in these ranked matrices. We demonstrate that this method is more robust than cartification in terms of cluster detection.

I. Introduction

Clustering is one of the core techniques in data mining; however, one of the challenges when clustering high dimensional data is the *curse of dimensionality*: when the number of dimensions is high, points in many realistic types of data become equidistant to each other, which makes it impossible to apply standard clustering techniques based on such distances [1].

One solution to this challenge is to perform *subspace clustering*. In subspace clustering, the data points are not clustered based on all attributes, but only on a subset of the attributes. By looking for subspaces in which good clusters can be identified, subspace clustering allows for the discovery of useful clusters even in high dimensional data.

Many subspace clustering techniques have been proposed in the literature [2], [3], [4], [5]. However, many of these approaches require many parameters to be set, and furthermore can still be sensitive to the scale of the distance function that is used [6]. This makes it harder to get reliable results using these clustering techniques.

To address this challenge, the *Cartification* approach to subspace clustering was developed [2]. In a nutshell, Cartification detects clusters by finding the co-occurring object sets in local neighborhoods. The most recent version of *Cartification* [7] exploits a well-known property of most subspace clusters: if a set of objects forms a cluster structure in a combination of dimensions, then this set of objects must also form a cluster structure in subsets of these dimensions. Therefore, Cartification first finds the cluster structures in one-dimensional projections and then iteratively refines these clusters to find higher-dimensional subspace clusters.

More precisely, for one attribute, Cartification operates as follows:

1) Calculate pairwise distances between all data points based on this attribute.
2) For each data point, calculate the k nearest data points based on these distances, resulting in a neighborhood database in which the i^{th} row contains a set of objects representing the k points that are closest to the i^{th} point. This neighborhood database is a *transaction database* and can be represented as a binary database in which each row has k columns that have the value 1; the other columns have value 0.
3) Search for large sets of objects that are frequently repeated in the rows of the resulting binary matrix; each such set represents a set of data points that are close to each other, and hence represents a cluster for this attribute.

For one attribute, the above algorithm returns a set of (possibly overlapping) clusters. By applying the approach recursively on the points within each of these clusters, Cartification finds subspaces consisting of sets of attributes and sets of points.

An advantage of the Cartification approach is that in step 3, it does not use the distance measure itself. This makes the approach less scale dependent; for instance, whether a logarithmic scale or a linear scale is used for an attribute has a smaller impact on the results.

Another advantage of Cartification is the interpretability of its parameters: neighborhood size k and *minimum cluster size*. Setting k and a lower bound on the size of clusters, both of which are functions of the expected cluster size, is much easier than determining settings for parameters such as density or distribution. Our earlier work showed that after parameter tuning of all methods, Cartification performs better than other subspace clustering approaches, in the sense that the clusters it finds are better in terms of an F1 score [2], [7].

On the other hand, earlier work [7] also showed that the quality of the results of Cartification is highly dependent on selecting an optimum value for the parameter k. Moreover, in some cases there is no optimum value for k because the database contains clusters of different sizes.

In this paper, we address the challenge of setting robust parameters for Cartification. We propose a new method that inherits many of the ideas of Cartification, but is less sensitive to a good choice for the parameter k. Roughly speaking, our method is as follows:

- instead of creating a binary transaction database in step 2 above, we create a new *rank-based* matrix, in which for every data point i, we rank the other points according to the distance to the point i;
- in the rank-based matrix, we apply a *rank-based tiling* method to identify points that are all close to each other. This rank-based method has only one parameter θ that influences the size of the tiles that can be found.

Note that by creating a rank-based matrix, we still maintain the advantage of Cartification that it does not rely on the scale of the original distance function; indeed, if we would put a threshold for all the ranks at k, we could easily create the binary matrix that was used in the original Cartification approach.

Furthermore, we will experimentally show that the proposed method is less sensitive to the choice of the parameter θ and produces clusters that are equally well as the original Cartification method, or better.

The outline of this paper is as follows: In Section II, we introduce Cartification; in Section III we introduce our proposed modification. We compare the binary Cartification method to our algorithm in Section IV. And we conclude in Section V.

II. Subspace Clustering using Cartification

This paper improves the Cartification approach for subspace clustering. In this section we will summarize the approach of the most recent version of Cartification [7]; this provides the foundations for our improvements.

Cartification operates on relational databases. A *relational database* $\mathcal{D}$ is defined as a set of data objects $\{\mathbf{o}_1, \ldots, \mathbf{o}_n\}$. Each data object $\mathbf{o}_i$ consists of a vector defined over a set attributes $\mathcal{A} = \{a_1, a_2, \cdots, a_m\}$. We are interested in finding subsets of attributes $A \subseteq \mathcal{A}$ and subsets of object identifiers $O \subseteq \{1, \ldots, n\}$ such that for each attribute $a \in A$, the values in the selected set of objects O are similar.

The Cartification algorithm is an instance of the SubspaceSearch procedure in Algorithm 1, where the procedure is initialized with $A = \emptyset$ and $O = \{1, \ldots, n\}$. The SubspaceSearch procedure iteratively considers all individual attributes, and identifies clusters in each individual attribute by using the function FindClusters. The function FindClusters(O, a_d) is the core of the algorithm and looks for good clusters for attribute a_d, restricting the search to objects in set O. For each resulting cluster, the search continues recursively if the cluster is large enough.

Due to its recursive nature, Cartification can identify a large number of subspace clusters. Important aspects in how many clusters it identifies are the choice for the parameter μ and the exact implementation of function FindClusters.

Algorithm 1 SubspaceSearch(A, O) procedure

Input: A subspace cluster as a pair of attributes A and object identifiers O

Output: Higher dimensional subspace clusters, being supersets of A and subsets of O

```
1: output subspace cluster $(A, O)$
2: for each $d \in \{1, \ldots, m\}$ do
3:    if $a_d \notin A$ then
4:       $S :=$ FindClusters $(O, a_d)$
5:       for each $O' \in S$ do
6:          if $|O'| \geq \mu$ then
7:             SubspaceSearch$(A \cup \{a_d\}, O')$
```

	A_1
$\mathbf{o}_1$	3000
$\mathbf{o}_2$	2000
$\mathbf{o}_3$	2300
$\mathbf{o}_4$	2600
$\mathbf{o}_5$	7
$\mathbf{o}_6$	11
$\mathbf{o}_7$	2100
$\mathbf{o}_8$	3500
$\mathbf{o}_9$	4
$\mathbf{o}_{10}$	16
$\mathbf{o}_{11}$	2
$\mathbf{o}_{12}$	1

(a) Database

6-*NN*$(\mathbf{o}_1)$	$\{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_7, \mathbf{o}_8\}$
6-*NN*$(\mathbf{o}_2)$	$\{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_7, \mathbf{o}_8\}$
6-*NN*$(\mathbf{o}_3)$	$\{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_7, \mathbf{o}_8\}$
6-*NN*$(\mathbf{o}_5)$	$\{\mathbf{o}_5, \mathbf{o}_6, \mathbf{o}_9, \mathbf{o}_{10}, \mathbf{o}_{11}, \mathbf{o}_{12}\}$
6-*NN*$(\mathbf{o}_6)$	$\{\mathbf{o}_5, \mathbf{o}_6, \mathbf{o}_9, \mathbf{o}_{10}, \mathbf{o}_{11}, \mathbf{o}_{12}\}$
6-*NN*$(\mathbf{o}_7)$	$\{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_7, \mathbf{o}_8\}$
6-*NN*$(\mathbf{o}_8)$	$\{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_7, \mathbf{o}_8\}$
6-*NN*$(\mathbf{o}_9)$	$\{\mathbf{o}_5, \mathbf{o}_6, \mathbf{o}_9, \mathbf{o}_{10}, \mathbf{o}_{11}, \mathbf{o}_{12}\}$
6-*NN*$(\mathbf{o}_{10})$	$\{\mathbf{o}_5, \mathbf{o}_6, \mathbf{o}_9, \mathbf{o}_{10}, \mathbf{o}_{11}, \mathbf{o}_{12}\}$
6-*NN*$(\mathbf{o}_{11})$	$\{\mathbf{o}_5, \mathbf{o}_6, \mathbf{o}_9, \mathbf{o}_{10}, \mathbf{o}_{11}, \mathbf{o}_{12}\}$
6-*NN*$(\mathbf{o}_{12})$	$\{\mathbf{o}_5, \mathbf{o}_6, \mathbf{o}_9, \mathbf{o}_{10}, \mathbf{o}_{11}, \mathbf{o}_{12}\}$

(b) Neighborhood Database

Fig. 1: A simple database of 12 object with one attribute and its Neighborhood DB

In Cartification [7], FindClusters relies on a database of k-nearest neighborhoods. The k-nearest neighborhood of one object in a database is the set of k objects that are closest to it; we can construct such a neighborhood for all objects in the database. More formally, we can define the k-nearest neighborhood for one object as follows.

Definition 1 (k-Nearest Neighborhood): Let $NN_k(\mathbf{o}_i)$ be the k^{th} closest object to $\mathbf{o}_i$, then the k-nearest neighborhood of $\mathbf{o}_i$ is defined as

$$k\text{-}NN(\mathbf{o}_i) = \{\mathbf{o}_j | \delta(\mathbf{o}_i, \mathbf{o}_j) \leq \delta(\mathbf{o}_i, NN_k(\mathbf{o}_i))\}$$

For example, the 6-nearest neighborhood of $\mathbf{o}_1$ in Figure 1a is 6-*NN*$(\mathbf{o}_1) = \{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_7, \mathbf{o}_8\}$.

A neighborhood database is created based on the k-nearest neighborhoods.

Definition 2 (Neighborhood Database): Given a parameter k, the neighborhood database is a vector

$$(k\text{-}NN(\mathbf{o}_1), k\text{-}NN(\mathbf{o}_1), \ldots, k\text{-}NN(\mathbf{o}_n)),$$

where k-*NN*$(\mathbf{o}_i)$ represents the set of neighbors of object i. The neighborhood database of the database in Figure 1 is shown in Figure 1b.

Note that the *neighborhood database* is a *transaction database*, as widely used in the domain of frequent itemset mining.

The idea behind Cartification is that the objects that form clusters co-occur in each others' neighborhoods. For example in Figure 1b, objects 1, 2, and 3 are members of the same cluster and they co-occur in the neighborhoods of the objects 1, 2, 3, 4, 7, and 8.

Consequently, Cartification uses an *itemset mining*-like approach to identify sets of frequently co-occurring objects. This is done by looking for frequent sets of objects in the neighborhood database [7]. Each such frequent set identifies a cluster, and is returned by the FINDCLUSTERS function used in Algorithm 1.

Characteristic for Cartification is that the order of the objects in the neighborhood matrix determines the clusters that are found; the distances in the data matrix are not used directly. This makes the approach less scale dependent.

An issue is, however, how to set k properly: for a large value of k, the neighborhood of each object will be large; indeed, for $k = |O|$, all neighborhoods would be identical and would contain all objects in O. Within the resulting database, we could potentially identify every subset of O as a cluster. If we set k too low, on the other hand, we also limit the maximum size of clusters that can be found. Finding a value for k that is neither too low nor too high is hence not easy. To address this problem, we propose a new approach in this paper that operates on the ordered neighborhoods.

III. RANKED CARTIFICATION

In this paper, we use the distances between objects in the data to define a ranked neighborhood matrix. We claim that in this matrix clusters can be identified more robustly.

We first formalize the problem of finding clusters in ranked data, followed by a new type of the FINDCLUSTERS algorithm that finds this type of cluster.

A. Problem Definition

Following the transformation idea of Cartification, we transform a relational database into a ranked neighborhood matrix, and then use this matrix to detect subspace clusters.

Definition 3 (Ranked Neighborhood): The *Ranked Neighborhood* of an object $\mathbf{o}_i$ is an n-dimensional vector, where $n = |\mathcal{D}|$, and is denoted as $\mathbf{N}(\mathbf{o}_i)$. The j^{th} value of the vector is defined as

$$\mathbf{N}(\mathbf{o}_i)_j = |\{\mathbf{o}_k \in \mathcal{D} \mid \delta(\mathbf{o}_i, \mathbf{o}_k) < \delta(\mathbf{o}_i, \mathbf{o}_j)\}|,$$

i.e., each dimension j represents the number of objects between the object $\mathbf{o}_i$ and the object $\mathbf{o}_j$ in the ordered neighborhood.

	$\mathbf{o}_1$	$\mathbf{o}_2$	$\mathbf{o}_3$	$\mathbf{o}_4$	$\mathbf{o}_7$	$\mathbf{o}_8$	$\mathbf{o}_5$	$\mathbf{o}_6$	$\mathbf{o}_9$	$\mathbf{o}_{10}$	$\mathbf{o}_{11}$	$\mathbf{o}_{12}$
$\mathbf{o}_6$	10	6	8	9	7	11	1	0	2	3	4	5

Fig. 2: Ranked neighborhood of $\mathbf{o}_6$

Figure 2 shows the *ranked neighborhood* of $\mathbf{o}_6$. The columns of the vector represent the objects 1 through 12. For example, $\mathbf{o}_9$ is the 2^{nd} closest object to $\mathbf{o}_6$, after $\mathbf{o}_5$, hence, the 9^{th} value of the vector is 2.

TABLE I: Ranked Neighborhood Matrix of Figure 1a

	$\mathbf{o}_1$	$\mathbf{o}_2$	$\mathbf{o}_3$	$\mathbf{o}_4$	$\mathbf{o}_5$	$\mathbf{o}_6$	$\mathbf{o}_7$	$\mathbf{o}_8$	$\mathbf{o}_9$	$\mathbf{o}_{10}$	$\mathbf{o}_{11}$	$\mathbf{o}_{12}$
$\mathbf{o}_1$	0	5	3	1	8	7	4	2	9	6	10	11
$\mathbf{o}_2$	4	0	2	3	8	7	1	5	9	6	10	11
$\mathbf{o}_3$	4	2	0	3	8	7	1	5	9	6	10	11
$\mathbf{o}_4$	1	5	3	0	8	7	4	2	9	6	10	11
$\mathbf{o}_5$	10	6	8	9	0	2	7	11	1	3	4	5
$\mathbf{o}_6$	10	6	8	9	1	0	7	11	2	3	4	5
$\mathbf{o}_7$	4	2	1	3	8	7	0	5	9	6	10	11
$\mathbf{o}_8$	1	5	3	2	8	7	4	0	9	6	10	11
$\mathbf{o}_9$	10	6	8	9	1	2	7	11	0	3	4	5
$\mathbf{o}_{10}$	10	6	8	9	2	1	7	11	3	0	4	5
$\mathbf{o}_{11}$	10	6	8	9	3	4	7	11	2	5	0	1
$\mathbf{o}_{12}$	10	6	8	9	3	4	7	11	2	5	1	0

Based on the ranked neighborhoods we define a ranked neighborhood matrix.

Definition 4 (Ranked Neighborhood Matrix): The *Ranked Neighborhood Matrix* is the matrix of ranked neighborhoods, i.e., it is the matrix

$$\mathbf{M} = \begin{pmatrix} \text{———} & \mathbf{N}(\mathbf{o}_1) & \text{———} \\ \text{———} & \mathbf{N}(\mathbf{o}_2) & \text{———} \\ & \vdots & \\ \text{———} & \mathbf{N}(\mathbf{o}_n) & \text{———} \end{pmatrix}$$

Table I shows the *ranked neighborhood matrix* for the database in Figure 1a. It has 12 rows and 12 columns representing the objects in the database. Each row is the ranked vector of the respective object: compare the row of $\mathbf{o}_6$ to Figure 2.

The ranked matrix preserves the neighborhood relations in the original database. If a set of objects is close to each other, their respective *rank*s will be under a certain threshold. In Table I, the mutual ranks of objects 1, 2, 3, 4, 7, 8 are all below 6. The same observation holds for the objects 4, 6, 9–12. Considering the original database in Figure 1a, we can see that both sets of these objects form clusters. Therefore, we can detect cluster structures by finding the minimum values in a ranked matrix.

Note the similarity to *Cartification*: the objects that have a ranking score below a certain threshold are the *neighbors* of the respective object. For example, the objects in 6-*NN*$(\mathbf{o}_1)$, cf. Figure 1b, have values below 6 at the first row of Table I. Nevertheless, in contrast to the set-based approach of Cartification's neighborhoods, ranked neighborhoods preserve the relative similarity information *in* the neighborhoods. Using the complete neighborhood information, we can make the cluster detection more robust, without fixing a threshold k.

The idea is to formalize the discovery of clusters as the discovery of *tiles* of low rank in the ranked neighborhood matrix.

Definition 5 (Tile): Given a set of object identifiers O, a *tile* in a ranked neighborhood matrix $\mathbf{M}$ is a square submatrix of the ranked neighborhood matrix. That is, for a given set of object identifiers O, it consists of all cells $\mathbf{M}_{ij}$ of $\mathbf{M}$ with $i, j \in O$.

We define the quality of a tile identified by a set of objects identifiers O as

$$f(O) = \sum_{i \in O, j \in O} (\mathbf{M}_{ij} - \theta),$$

where θ is a scalar value for thresholding.

A *minimal tile* is a tile that has minimal score $f(O)$.

The idea behind the scoring function is that cells in the ranked neighborhood matrix with a value higher than the threshold will contribute a positive term to the summation, while cells below the threshold contribute a negative term. Clusters are preferred that have as many cells below the threshold θ in the corresponding tile.

Note that we need to solve this minimization problem multiple times, as we need to solve it for the different attributes independently. The FINDCLUSTERS implementation for ranked Cartification hence solves the problem of finding the minimal tile for each attribute independently, while restricting the search to only those objects that it is allowed to put in a cluster, as identified by the O parameter of the FINDCLUSTERS function.

Indirectly, the parameter θ influences the size of the clusters that can be found. In our experimental evaluation, we will compare the effect of this parameter to that of the k parameter in the original Cartification method.

B. Properties

The problem of ranked tiling was studied in earlier work by Le Van et al. [8]. However, given the origin of the ranked matrix studied in ranked Cartification, we can use a more efficient algorithm in this particular setting. Our algorithm relies on the particular properties of the sorted ranked neighborhood matrix.

Definition 6 (Sorted Ranked Neighborhood Matrix): Given an attribute a_d, the *sorted ranked neighborhood matrix* for this attribute is the ranked neighborhood matrix obtained by sorting the objects in the rows and columns according to their values for attribute a_d.

For example, if we sort the rows and columns of the ranked neighborhood matrix in Table I according to the object values in a_1, then we will obtain the matrix in Table II.

TABLE II: Sorted Ranked Neighborhood Matrix for a_1

	$\mathbf{o}_{12}$	$\mathbf{o}_{11}$	$\mathbf{o}_9$	$\mathbf{o}_5$	$\mathbf{o}_6$	$\mathbf{o}_{10}$	$\mathbf{o}_2$	$\mathbf{o}_7$	$\mathbf{o}_3$	$\mathbf{o}_4$	$\mathbf{o}_1$	$\mathbf{o}_8$
$\mathbf{o}_{12}$	0	1	2	3	4	5	6	7	8	9	10	11
$\mathbf{o}_{11}$	1	0	2	3	4	5	6	7	8	9	10	11
$\mathbf{o}_9$	5	4	0	1	2	3	6	7	8	9	10	11
$\mathbf{o}_5$	5	4	1	0	2	3	6	7	8	9	10	11
$\mathbf{o}_6$	5	4	2	1	0	3	6	7	8	9	10	11
$\mathbf{o}_{10}$	5	4	3	2	1	0	6	7	8	9	10	11
$\mathbf{o}_2$	11	10	9	8	7	6	0	1	2	3	4	5
$\mathbf{o}_7$	11	10	9	8	7	6	2	0	1	3	4	5
$\mathbf{o}_3$	11	10	9	8	7	6	2	1	0	3	4	5
$\mathbf{o}_4$	11	10	9	8	7	6	5	4	3	0	1	2
$\mathbf{o}_1$	11	10	9	8	7	6	5	4	3	1	0	2
$\mathbf{o}_8$	11	10	9	8	7	6	5	4	3	2	1	0

The following properties hold for a *sorted ranked matrix* $\mathbf{M}$:

Property 1: For all i: $\mathbf{M}_{ii} = 0$.

Proof: The diagonals are always zero since the order of rows and columns is the same and each point is most similar to itself. ■

Property 2: For the values in the i^{th} row, the following statement holds:

- for j and k with $i < j < k$: $\mathbf{M}_{ij} \leq \mathbf{M}_{ik}$;
- for j and k with $j < k < i$: $\mathbf{M}_{ij} \geq \mathbf{M}_{ik}$.

Proof: By definitions 3 and 4, each row of $\mathbf{M}$ is a ranked neighborhood, i.e., $\mathbf{M}_{ij} = \mathbf{N}(\mathbf{o}_i)_j$. Furthermore, each column corresponds with a value x_j for the attribute a_d. Since the columns are sorted according to these values, $i < j < k \implies x_i \leq x_j \leq x_k$, and hence $\delta(x_i, x_j) \leq \delta(x_i, x_k)$. The second half of the property can be proven in a similar way. ■

We prove a similar property for the columns.

Property 3: For the values in the i^{th} column, it holds that:

- for j and k with $i < j < k$: $\mathbf{M}_{ji} \leq \mathbf{M}_{ki}$;
- for j and k with $j < k < i$: $\mathbf{M}_{ji} \geq \mathbf{M}_{ki}$.

Proof: Similar to the proof of Property 2, $i < j < k \implies x_i < x_j < x_k \implies \mathbf{M}_{ji} \leq \mathbf{M}_{ki}$. Here, the additional argument is that by moving lower down or higher up in a column from a particular $\mathbf{M}_{ii}$, the number of objects ranked lower than the object in column i can only increase, as the objects lower down and higher up the matrix for this column are further away from the object $\mathbf{o}_i$. ■

From these properties follows an important new property that allows us to improve the search significantly:

Property 4: Minimal tiles are contiguous, i.e., the set of objects chosen in a minimal tile O always consists of a range $O = \{i, i+1, \ldots, j-1, j\}$ for some $1 \leq i \leq j \leq n$.

Proof: This follows from the Properties 2 and 3. ■

All of these properties can be observed in Table II: (1) diagonals are all zero, (2) the value for $\mathbf{o}_5$ at the row of $\mathbf{o}_6$ is smaller than the value for $\mathbf{o}_9$, because $\mathbf{o}_5$ is closer to the diagonal, (3) similarly, the value for $\mathbf{o}_5$ is smaller at the row of $\mathbf{o}_2$ than the value for $\mathbf{o}_5$ at the row of $\mathbf{o}_6$.

C. Algorithm

To find subspace clusters, we propose the CARTIRANK algorithm, which follows the procedure in Algorithm 1. Similar to Cartification, CARTIRANK is initialized with an object set containing all possible objects and an empty dimension set. Firstly, it finds the clusters in one dimensional projections, and then, iteratively refines them to find higher dimensional subspace clusters. The main difference between CARTIRANK and Cartification is the approach taken to detect one dimensional clusters, i.e., it uses a different FINDCLUSTERS function. Instead of mining frequently co-occurring object sets in neighborhood databases, CARTIRANK uses sorted ranked neighborhood matrices.

Property 4 allows us to develop an efficient search algorithm to find minimal tiles: instead of searching over all possible subsets of objects, we search over all possible combinations of i and j with $1 \leq i \leq j \leq n$. This yields the algorithm in Algorithm 2 for finding a minimal tile. SQUARETILER takes the sorted ranked neighborhood matrix and the threshold θ

Algorithm 2 SQUARETILER: Tiling on Sorted Ranked Matrix

```
Input: $\mathbf{M}$: Rank Matrix, $\theta$
Output: $min\_t$: Minimum tile
 1: $min\_t := (0,0)$  // Minimum tile
 2: $min\_ts := -\theta$  // Minimum tile sum
 3: for $i := 1$ to $n$ do
 4:     $ts := \mathbf{M}_{ii} - \theta$  // Tile sum
 5:     for $j := i+1$ to $n$ do
 6:         $ns := (\mathbf{M}_{ij} - \theta) + (\mathbf{M}_{(i+1)j} - \theta) + \cdots + (\mathbf{M}_{jj} - \theta)$
 7:             $+ (\mathbf{M}_{ji} - \theta) + (\mathbf{M}_{j(i+1)} - \theta) + \cdots + (\mathbf{M}_{j(j-1)} - \theta)$
 8:         if $ns > 0$ then
 9:             Continue with the next $i$
10:         $ts := ts + ns$
11:         if $ts < min\_ts$ then
12:             $min\_ts := ts$
13:             $min\_t := (i,j)$
14: return $min\_t$
```

as parameter. Exploiting the fact that ranks will only increase when increasing j, it stops growing a tile if the new addition of values does not decrease the sum (lines 8 and 9). This optimization exploits Properties 2 and 3.

IV. Experimental Evaluation

In this section, we empirically evaluate the cluster finding capabilities of CARTIRANK. We compare the robustness of CARTIRANK and Cartification with respect to parameter settings and data distributions. The overall cluster quality is evaluated in comparison with other state-of-the-art subspace clustering methods. For the reproducibility of the results, we provide the implementation and the datasets on our web site.[1]

A. Robust Cluster Detection

We compare the cluster detection capabilities of Cartification and CARTIRANK. Both of the algorithms exploit one-dimensional projections to detect clusters. Therefore, they should be able to find clusters with various sizes and properties in projections without requiring strict parameter settings. In other words, the quality of the clusters should be robust to both parameters and the characteristics of the data. To evaluate the robustness of one-dimensional cluster finding capabilities, we generated 6 synthetic datasets. Properties of these dataset are shown in Table III. **ns25** and **ns50** have three clusters that are not separable, i.e., the diameters of the clusters are larger than the distances between clusters. **s3** has 3 clusters with sizes 25, 50, and 25. **vs4**, **vs5**, and **vs6** have clearly separated clusters of different sizes.

On these 6 datasets, we ran Cartification [7] and CARTIRANK with different neighborhood sizes, by setting θ and k respectively, and report the quality of the found clusters. Since we know the true clusters beforehand, we use a supervised F1 score to assess the quality of the found clusters. The F1 score is the harmonic mean of the *precision* and *recall* between

[1]http://adrem.uantwerpen.be/cartirank

TABLE III: Datasets for robustness tests

Name	# of clusters	Cluster Sizes	Cluster Radius	Distance Between Clusters
ns25	3	25-25-25	4	3
ns50	3	50-50-50	4	3
s3	3	25-50-25	4	4
vs4	4	25-40-55-70	4	8
vs5	5	50-25-50-25-50	4	8
vs6	6	15-30-45-45-30-15	4	8

two cluster sets. *Precision* between a known cluster C_1 and a detected cluster C_2 is the size of the intersection of C_1 and C_2 divided by the size of C_2. Likewise, *recall* is the number of common objects between C_1 and C_2 divided by the size of C_1. We map each known cluster to the detected cluster which produces the maximum F1 score and take the average of the scores. In other words, we measure whether the true clusters are among the clusters found using CARTIRANK and Cartification.

Figure 3 shows the F1 scores of the algorithms. We see the benefits of using the similarity information in the neighborhoods, i.e., using ranks, on all of the datasets: CARTIRANK produces better quality clusters for a wider range of θ values. When the clusters are the same size but not separated, Cartification can detect the perfect clusters only for one parameter value, while CARTIRANK can produce the perfect clustering for a wide range of values.

The benefits are more visible on datasets that have clusters of various sizes, namely **vs4** and **vs6**. Given that k defines a limit on the size of clusters that can be found, cartification can find only one cluster size at a time, and thus, can never find all of the clusters at once. CARTIRANK can produce a perfect clustering for all of the datasets.

Our experiments with a wide range of values for the parameter μ show that the effect of this parameter is negligible. For the same θ, in these experiments, the μ parameter does not change the outcome at all, or changes the F1 score of the found clusters not more than a value of 0.05.

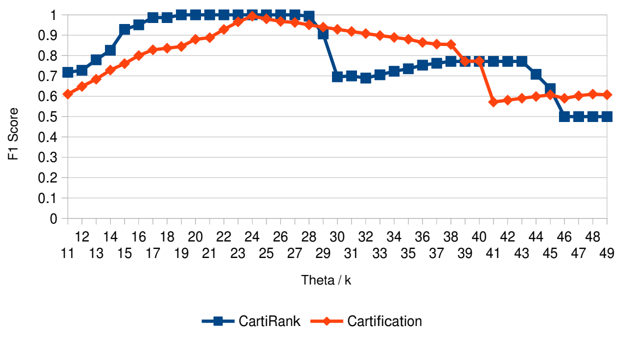
B. Subspace Clustering

To evaluate subspace cluster finding capabilities of the methods, we generated a set of datasets. Each dataset has five sets of overlapping clusters which are hidden in five sets of attributes. An object can be a member of one cluster according to one attribute and a different cluster in another attribute.

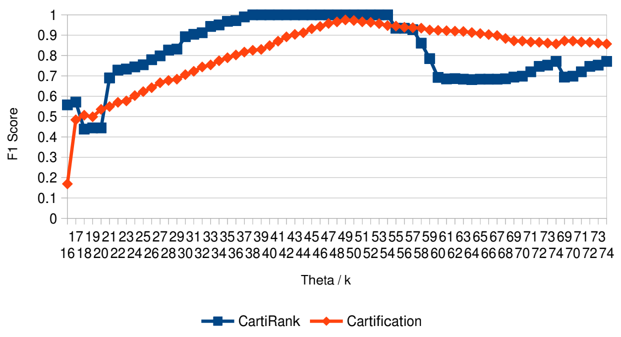
An example cluster formation is visualized in Figure 6, in which the columns represent attributes and the rows represent objects. Cluster assignments in different attributes are shown as C_i. For example $\mathbf{o}_1$ and $\mathbf{o}_i$ are in cluster C_1 according to attributes a_1 and a_2, but they are in different clusters in a_k.

Generation parameters for the datasets are shown in Table IV. After generating datasets using these parameters, we added two redundant dimensions with uniform random values along with 5% random noise.

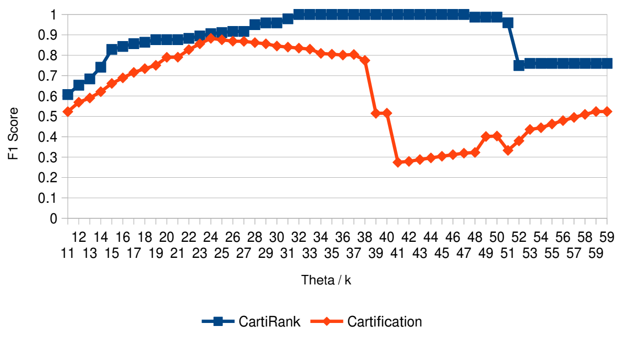
The F1 scores of the found clusters for a range of parameters are shown in Figure 4. As in one-dimensional clustering, the



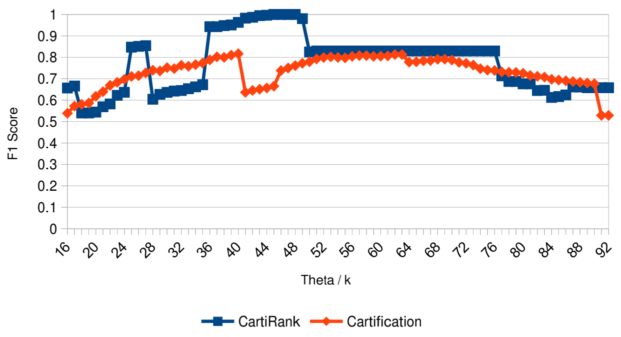
(a) **ns25**



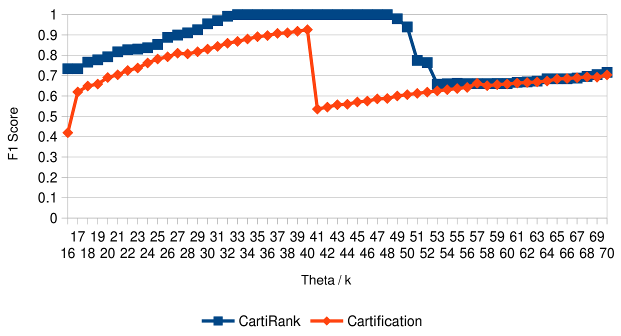
(b) **ns50**



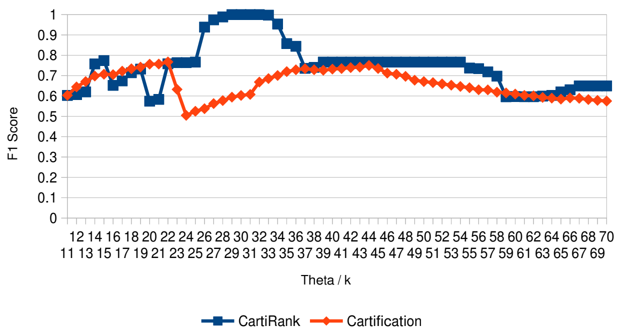
(c) **s3**



(d) **vs4**



(e) **vs5**



(f) **vs6**

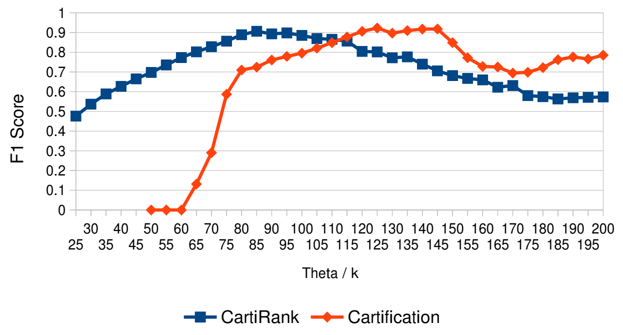
Fig. 3: F1 Scores of the methods for a range of parameters

TABLE IV: Generation parameters of subspace cluster datasets

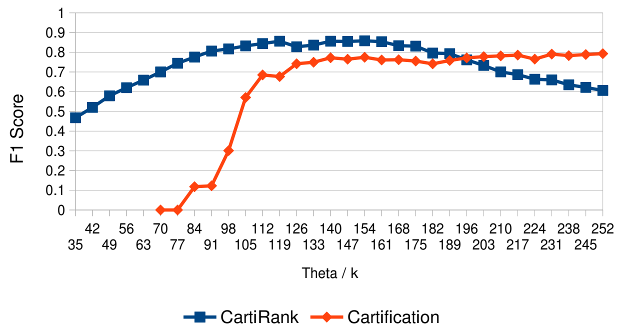
Name	Cluster #	Attributes/Cluster	Cluster Sizes	Attribute #
SVS-1	15-25	2-6	50-100	10-30
SVS-2	15-25	2-6	50-150	10-30
SVS-3	15-25	2-5	50-200	10-30

qualities of the clusters produced by CartiRank are always in a certain interval. For the optimal parameters settings, the quality of the found clusters are comparable. There is a catch however: Figure 7 shows the number of clusters found by each of the methods. We can see that cartification outputs so many clusters that they can not be put to good use in practical scenarios. Clusters found by CartiRank are orders of magnitude less redundant.

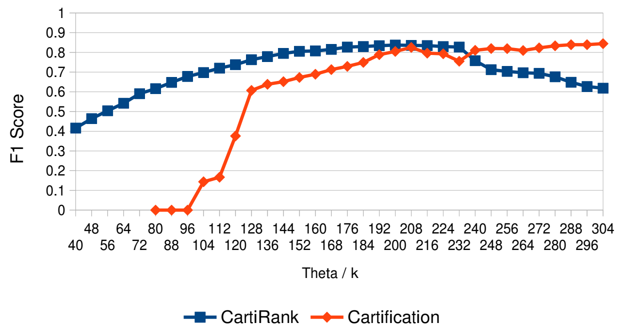
We compare the quality of CartiRank with two other state-of-the-art subspace clustering algorithms. Figure 5 shows the F1 scores of CartiRank, Cartification, PROCLUS [4], and STATPC [5] for subspace clustering datasets. We optimized the parameters of the algorithms for the best results. As expected, PROCLUS can cope neither with overlapping clusters nor with noise. STATPC uses an approximation, and obviously, its assumptions do not hold for the datasets under consideration. Moreover, optimizing the complex parameters of STATPC is not trivial.



(a) SVS-1



(b) SVS-2



(c) SVS-3

Fig. 4: Quality of the subspace clusters

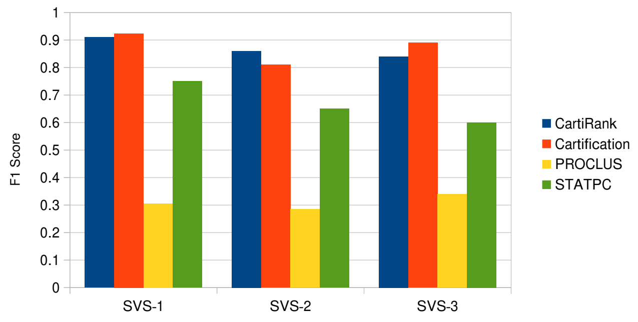


Fig. 5: The best clusters found by the different algorithms

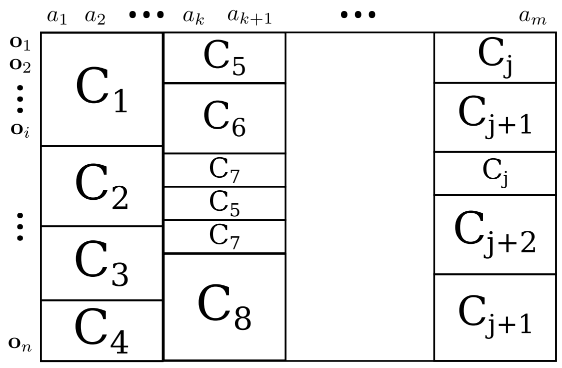


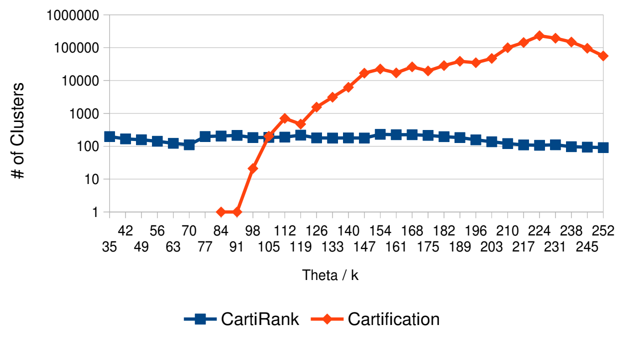
Fig. 6: Subspace clusters in synthetic datasets

V. Conclusion

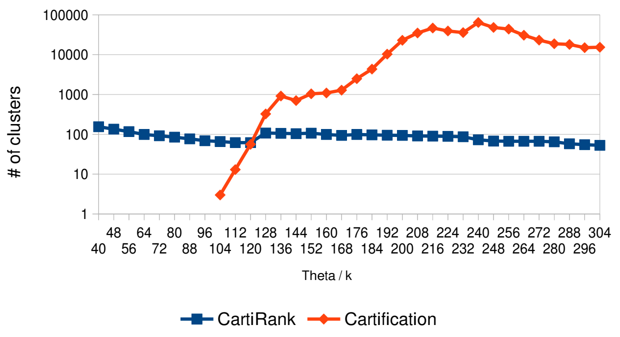
In this paper, we tackle the problem of detecting subspace clusters. Following our previous work [2], [7], [8], we exploit local neighborhoods by transforming a relational database into sorted ranked neighborhood matrices. We study the properties of this matrix and show the relation between minimal tiles in the matrix and the cluster structures in the data. We propose a method that exploits the intrinsic properties of the matrix to efficiently find interesting tiles in them.

In contrast to the binary neighborhood databases used in *cartification*, ranked matrices preserve the similarity information in the neighborhoods. Therefore, mining subspace clusters by using the ranked neighborhood matrices is more robust to both input parameters and the formation of the clusters. Moreover, CartiRank produces a manageable number of clusters, making it a better fit for practical use.

The use of ranked neighborhoods matrices however comes at a cost: the computations on these matrices are more costly and require more memory. Although exploiting the properties of sorted ranked neighborhood matrices improves the core performance drastically, CartiRank is still slower than the



(a) SVS-2



(b) SVS-3

Fig. 7: The number of clusters found by the methods

most recent Cartification method [7]. The optimization of memory and processor use is left as a future work.

References

[1] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is ”nearest neighbor” meaningful?” in *Proceedings of the 7th International Conference on Database Theory (ICDT), Jerusalem, Israel*. Springer, 1999, pp. 217–235.

[2] E. Aksehirli, B. Goethals, E. Müller, and J. Vreeken, “Cartification: A neighborhood preserving transformation for mining high dimensional data,” in *Proceedings of the 13th IEEE International Conference on Data Mining (ICDM), Dallas, TX*, Dec. 2013, pp. 937–942.

[3] K. Kailing, H.-P. Kriegel, and P. Kröger, “Density-connected subspace clustering for high-dimensional data,” in *Proceedings of the 4th SIAM International Conference on Data Mining (SDM), Lake Buena Vista, FL*, vol. 4. SIAM, 2004.

[4] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park, “Fast algorithms for projected clustering,” *ACM SIGMOD Record*, vol. 28, no. 2, pp. 61–72, Jun. 1999.

[5] G. Moise and J. Sander, “Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering,” in *Proceedings of the 14th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Las Vegas, NV*, ser. KDD ’08. New York, NY, USA: ACM, 2008, pp. 533–541.

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

[7] E. Aksehirli, B. Goethals, and E. Müller, “Efficient cluster detection by ordered neighborhoods,” in *Proceedings of the 17th International Conference on Big Data Analytics and Knowledge Discovery (DaWaK), Valencia, Spain*, 2015, pp. –.

[8] T. Le Van, M. v. Leeuwen, S. Nijssen, A. C. Fierro, K. Marchal, and L. D. Raedt, “Ranked Tiling,” in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, T. Calders, F. Esposito, E. Hllermeier, and R. Meo, Eds. Springer Berlin Heidelberg, Sep. 2014, no. 8725, pp. 98–113.