
# Finding Subspace Clusters using Local Neighborhoods

Proefschrift

voorgelegd tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica
aan de Universiteit Antwerpen
te verdedigen door

**Mehmet Emin AKŞEHİRLİ**

Promotor: prof. dr. Bart Goethals

Antwerpen, 2015

*Finding Subspace Clusters using Local Neighborhoods*
Nederlandse titel: *Het Vinden van Subspace Clusters met Lokale Buurten*


Front cover photo by Pictometry, courtesy of Go iTalk
Back cover photo by Emre Akşehirli

*“Tell me who your friends are and I will tell you who you are.”*

*— Proverb*

# Acknowledgements

I am having a hard time to believe that it has already been four years since I have started my PhD. I have not been alone during these last four years of my 26-year journey of being an *official* student. I would like to thank and give my regards to the people that influenced me during these four years.

I would like to express my deepest gratitude for my supervisor and promotor Prof. Bart Goethals. Not only because it has been very enjoyable and fruitful to work with him, but also he let me to investigate, in his words, *his most exciting idea ever*. This thesis includes only a small portion of our many exciting theories, less exciting experiments, in-deep discussions, and our work in general.

I would like to thank to my jury members, Floris Geerts, Kris Laukens, Toon Calders, and Emmanuel Müller for their constructive comments. Their inputs considerably improved the thesis.

I would like to thank to my co-authors. This thesis could not have been produced without them. I would like to thank Emmanuel for his professional advices to improve my research and showing me the exit path when I got lost in the labyrinth of abstract ideas. I consider myself very lucky to have spend time with Jilles. His limitless perfectionism, which also applies to our work beyond *strictly business*, has set new standards for me. It has been inspirational to work with Sandy. Our casual but in-depth discussions and his work ethics provided me the daily boosts when I most needed. I would like to thank Siegfried and Matthijs for putting up with me while they guided me through a whole new dimension of doing research. At the end, I learned to ask 4 questions: What would... - Bart innovate? - Emmanuel write? - Sandy do? - Jilles fix?

Tayena offered her friendship when we most needed it: when we were still new in Belgium, and she still does. I feel very lucky to know Tayena and Jens. Cheng carries the title of being my office-mate for the longest time, ever. I will not forget his endless energy, dynamism and sincere friendship. It has been a pleasure to meet both with Antonio and Hernán. Although we could not spent enough time together, they were my go-to guys when I needed the Mediterranean warmth.

I know that it is very important to be surrounded with good people. It has been a pleasure to work with the people of ADReM: Aida, Alvaro, Anthony, Bart C., Boris, Charlie, Elyne, Floris, Jan, Jeroen, Joeri, Koen S., Koen V., Kris, Len, Martin, Michael, Nghia, Nicolas, Nikolaj, Pieter, Reuben, Stefan, Thomas, Tim, Wout.

I would like to thank my family for their support and understanding. They set the high academic standards in the family that I could have barely reached. I thank my sister Yasemin for her academic and moral support and my brother Emre for letting me use one his photos on the cover.

Speaking of families, I also thank our *expat family* here for providing the warmth of a family: Alla, Hyojin, Jan, Kelly, Matt, Otto, Raphaël, Vica, and new members Charlotte, Eric, Luca. Let me not forget my friends, who have been motivating me with academic, technical, and casual discussions, although I have never seen faces of some of them. One of the hidden heroes behind this thesis is a specific friend who has provided me access for the resources of their university library. Thank you.

I would like to thank the University of Antwerp for giving me the opportunity to pursue my PhD. I could not help but think about how hard to do this without the *fantastic* Belgian weather, which almost always gives a reason to stay at home and work. I would like to thank Belgium for supporting research and Turkey for keeping me motivated to be more successful.

Lastly, but of course not the least, I thank my wife Güneş. She jumped on this train of PhD adventure with me without any hesitation. And now I know that I could not have done half of what I have done without her. I feel like I can take any adventure with her being on my side. Thank you for being in my life.

*Mehmet Emin Akşehirli*
*Antwerpen, 2015*

# Contents

# Publications

- Sandy Moens, **Emin Aksehirli** and Bart Goethals. Frequent Itemset Mining for Big Data. In *Proceedings of the IEEE International Conference on Big Data, 2013* , pages 111–118, 2013.
- **Emin Aksehirli**, Bart Goethals, Emmannuel Müller, and Jilles Vreeken Cartification: A Neighborhood Preserving Transformation for Mining High Dimensional Data. In *Proceedings of the 13th IEEE International Conference on Data Mining, 2013 (ICDM 2013)*, pages 937–942, 2013.
- **Emin Aksehirli**, Bart Goethals, and Emmannuel Müller. Visual Interactive Neighborhood Mining on High Dimensional Data. In *Proceedings of the ACM SIGKDD 2015 Workshop on Interactive Data Exploration and Analytics) (IDEA 2015)*, pages 10–19, 2015.
- **Emin Aksehirli**, Bart Goethals, and Emmannuel Müller Efficient Cluster Detection by Ordered Neighborhoods. In *Proceedings of 17th International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2015)*, pages 15–27, 2015.
- **Emin Aksehirli**, Siegfried Nijssen, Matthijs van Leeuwen, and Bart Goethals Finding Subspace Clusters using Ranked Neighborhoods. In *Proceedings of IEEE ICDM Workshop on High Dimensional Data Mining (HDM 2015)*, (to appear)

CHAPTER 1

# Introduction

For quite a few years now, data became more influential then it has ever been. Considering how it shapes the world around us, be it more effective decisions or the services that became part of our lives, one can see that the importance of data will not fade away soon.

Organizations around the world have already understood the importance of data and for the last few years, they have been trying to keep all the information they can access. Almost none of the organizations that has a future plan spanning more than 5 years can ignore the importance of data: commercial companies, governmental organizations, not-for-profit organizations, and even political candidates. Of course, this new trend of data-oriented approaches have been going hand in hand with the advancements in data acquisition and storage techniques in the recent years.

Technological advancements in data acquisition was so fast and the prices for data storage dropped so quickly that people started to collect data without considering how to structure it or even whether it will be useful. We can see the effects of this *data hoarding* as the technologies that have recently become very popular: while NoSQL and column-based databases enable collection of unstructured data, simple parallelization platforms, such as MapReduce, provide tools for wrangling the data. All of these trends are addressed by one umbrella term: *BigData*.

Since it is not trivial to annotate the massive amounts of data, unsupervised methods are a good fit for BigData analysis. While cluster analysis can be used as a pre-processing step to summarize and prepare data for more elaborate supervised data analysis techniques, frequent pattern mining methods can be used to sift through data to discover interesting relations.

One of the characteristics of the BigData is, rather unsurprisingly, the amount of data, and more precisely, the amount of data that are associated with a single data object. Although, intuitively, more data bring better and more accurate understandings, it is also possible that redundancy in data just hinder exploration of interesting relations. Moreover, it has been mathematically shown that when the number of dimensions increase, data objects become more and more similar to each other. This phenomenon is known as *the curse of dimensionality*.

Because the notion of similarity is distorted in higher dimensional spaces, traditional clustering methods, which heavily depend on similarity measures, cannot be as effective. On the other hand, similarity according to subsets of attributes is still meaningful and can be used to discover interesting relations.

*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 be achieved by looking at individual dimensions, e.g. travel and sport habits only. In both of these individual attributes customers might be clustered. Even in the combination of these two attributes, the same customer structure might be detected as clusters.*

Subspace clustering algorithms tackle this high dimensionality problem by defining a cluster as a pair of two sets: the objects that are similar to each other and the dimensions in which they are similar. Consequently, limiting the measurement space degrades the effects of curse of dimensionality, and highlights the similarities of objects by producing more compact clusters. However, dimension search makes the already NP-hard problem of cluster analysis even more complicated.

In this thesis, we approach the high dimensionality problem from a neighborhood-oriented point of view. We investigate various ways to transform a relational database, so that we can expand our tool set that we can use to extract knowledge. In a nutshell, *cartification* transformation enables the utilization of frequent pattern mining methods on relational databases while neighborhood-based representation opens new possibilities for visualization and interactiveness.

The thesis is organized as follows:

**Chapter 2** introduces the main theme of this thesis: *Cartification* transformation. Cartification transforms a relational database into a transaction database by preserving the localities of individual data objects as neighborhoods. By transforming the data space, cartification enables the whole domain of frequent itemset mining (FIM) methods to be used for relational data analysis. We show that the *curse of dimensionality* can be diminished by using localities and applying even the most basic FIM methods on cartified data produces results that beat the state of the art cluster analysis methods.

**Chapter 3** introduces CLON algorithm. Cartification is a general transformation that can be applied on any type of data as long as a similarity measure between data objects are defined. When a total order of the objects is possible, i.e., the attribute under consideration is univariate, then the transformed database gains some intrinsic properties. These properties can be exploited to eliminate the computationally expensive process of frequent itemset mining. CLON is orders of magnitude faster

than the original cartification and gives comparable or better results on synthetic and real datasets.

**Chapter 4** approaches to the caritification from a more sophisticated perspective. Instead of creating a binary transaction database, CARTIRANK creates ranked neighborhood matrices. We study the properties of these ranked neighborhood matrices and propose an algorithm that exploits them to efficiently mine tiles, which are the representations of cluster structures. We show that preserving the order of objects yields more robust results.

**Chapter 5** is about our tool VINeM, which makes the neighborhood mining available for a wider audience. VINeM, the name of which stands for Visual Interactive Neighborhood Miner, provides a user friendly interface that combines an intuitive representation of neighborhoods with unsupervised algorithms. Main focus of VINeM is to show the relations of objects among different attributes, so that a user can quickly get a grasp of the data. Moreover, the interactivity enables the user to perform basic data manipulations and to create different views on the data.

**Chapter 6** introduces two frequent itemset mining algorithms that are implemented on the Hadoop framework. One of the key constraints of interactive data exploration settings is the reaction time. Fortunately, heavy mining algorithms can be *outsourced* to cloud services, so that the limited resources of the client can be better utilized. In this manner, Dist-Eclat and BigFIM are two parallel algorithms that can be run on cloud services. We show that performance of mining algorithms in a MapReduce paradigm depends heavily on balanced distribution of data among the nodes. We did exhaustive experiments and investigate the ways to distribute data and computational burden optimally.

**Chapter 7** gives an overview of our contributions and concludes the thesis with an outlook for future possibilities.

CHAPTER 2

# Cartification

*Analyzing high dimensional data comes with many intrinsically hard challenges. In real world data, for instance, cluster structure is typically only found in subsets of all dimensions—that is, many dimensions are irrelevant for each individual cluster. When one knows the right subspace, finding a cluster is relatively straightforward, as standard algorithms apply. Finding the right spaces, however, is an open combinatorial problem, while by the curse of dimensionality, if irrelevant dimensions are included, distances measured between objects become very similar, making cluster detection all but impossible.*

*In this chapter*[1] *we propose* Cartification*, a new transformation to circumvent these problems. We transform each object into an itemset, which represents the neighborhood of the object. We do this for multiple views on the data, resulting in multiple neighborhoods per object. This transformation enables us to preserve the essential pairwise-similarities of objects over multiple views, and hence, to improve knowledge discovery in high dimensional data. In particular, by mining frequent itemsets in these neighborhoods, data clusters hidden in the original data are identified. Moreover, as neither irrelevant dimensions nor noise add to these frequencies, they do not hinder cluster detection.*

*Empirical evaluation shows the effectiveness of our transformation, as it allows a straightforward itemset mining technique to outperform the state of the art, including traditional clustering, random projections, principle component analysis, subspace clustering methods, and clustering ensemble on data projections.*

[1] This chapter is based on work published in ICDM 2013 as "Cartification: A neighborhood preserving transformation for mining high dimensional data" by Emin Aksehirli, Bart Goethals, Emmanuel Müller and Jilles Vreeken [10].

## 2.1 Introduction

Many knowledge discovery tasks rely on some notion of similarity between objects. For example, in clustering, the goal is to group similar objects into a cluster, while separating dissimilar objects into different clusters. For many real world applications, such as in customer segmentation, gene expression analysis, and health surveillance, we observe that their inherently high dimensional data spaces hinder the application of traditional clustering algorithms. In particular, one of the main problems with high dimensional data is that the distances between pairs of objects, measured over the full data space, rapidly grow more and more alike for higher numbers of dimensions—which essentially renders the notion of neighborhoods meaningless in the high dimensional space [20]. This effect is well-known as the *curse of dimensionality*, and affects all methods that consider the full data space.

Loosely speaking, on the one hand we find existing approaches that aim to alleviate this problem by finding subspaces over which to calculate similarities. Examples include unsupervised feature selection [30, 89], dimensionality reduction [86], metadistance measures [55], or subspace search [25]. Each of these proposes a fixed notion of similarity and is not flexible in adapting to the hidden cluster structure, or, local neighborhoods, of the data. On the other hand we find subspace clustering, which proposes to detect individual projections for each relevant local neighborhood [84, 78]. Unfortunately, however, finding the relevant dimension sets is a combinatorial problem that adds to the intrinsic complexity of the cluster analysis itself. Overall, existing subspace approaches lack efficient computation [6, 56, 76], are proven to be NP-hard [77], or lack local subspace projections [25, 59, 79].

In this chapter we propose a new transformation to circumvent these problems. In particular, we tackle the dual challenge of (1) considering multiple local subspaces and (2) exponential complexity of the search space. We propose to not directly consider the original data, but to transform it such that all local neighborhoods are preserved, allowing for example, clustering data in multiple projections of the data. Intuitively, we consider multiple views on the original data space (i.e., a high dimensional database) and transform these views into a novel itemset space. We call our transformation *cartification*, following the illustrative idea of building shopping carts, and storing the bought products in the transactional data scheme known from frequent itemset mining.

In a nutshell, we transform each object into a *cart* containing all $k$ nearest neighbors of that object. For each object, we exploit multiple views on the original data space by using different (dis)similarity measures for our neighborhood assessment. By using multiple views on the data, neighborhood information, i.e. similarity, is preserved with increasing number of dimensions in at least some of these views. This leads to a robust preservation of local neighborhoods in subspace projections of the original data space. Applying frequent itemset mining then reveals hidden cluster structures. Frequency of items, and itemsets, implicitly measures the similarity of co-occurring objects, and hence, automatically excludes irrelevant dimensions with noisy data distributions. That is, in our transformed data, an itemset with a high frequency means that the corresponding objects share a local neighborhood. By checking which transactions support the itemset, we can easily infer the subspace in which these objects are clustered. Moreover, as neither noisy objects nor irrelevant dimensions will add significantly to the supports, they do not interfere with the

detectability of subspace clusters.

In our experiments we focus on clustering as the underlying data mining task and show that our transformation is more stable for cluster detection than using the original data space. We are able to detect clusters and their individual subspace projections. For each cluster, our method excludes locally irrelevant dimensions, and hence, is robust w.r.t. increasing numbers of dimensions. Overall, we show that our transformation outperforms traditional clustering, random projections, principle component analysis, subspace clustering methods, and clustering ensemble on data projections.

## 2.2 Related Work

In this chapter we consider clustering as an application domain for our transformation. Traditional clustering methods [51] have been shown to be affected by increasing numbers of dimensions in real databases. Clustering models that measure similarities over all given dimensions (i.e., the full data space) are hindered by irrelevant dimensions and loss of contrast in high dimensional data spaces [20, 19]. In particular, traditional distances such as Euclidean or $L_p$ norms are affected by the concentration of norms with increasing number of dimensions [63]. We address this general problem and propose a data transformation for more robust cluster detection in subspaces of high dimensional data.

Dimensionality reduction techniques [86, 63] or unsupervised feature selection [30, 89] are general techniques for projecting a database to single lower-dimensional projections. Such projections, however, incur information loss. Moreover, these methods are limited to single views of the data. That is, clusters are detected only in a single projection although other projections might reveal additional cluster structures and novel knowledge as well. We propose a different processing, which captures multiple views on the data, and hence, allows for cluster detection in arbitrary subspace projections.

Bi-clustering [100], and subspace clustering [84, 78] search for clusters in subsets of dimensions. However, every clustering model and algorithm needs special adaptation. For example, the well-known K-Means algorithm has been extended to subspace projections, namely PROCLUS [5]. Other clustering notions such as DBSCAN need a totally different processing [56]. Recently, more general approaches have been proposed for subspace search [25, 57, 59, 79], leaving the discovery of clusters or outliers to specialized algorithms. However, all of these methods are intrinsically computationally expensive as they search in an exponential set of subspaces. With our method we follow the idea of mining multiple subspace projections. However, in contrast to the existing approaches we aim at a general transformation of the problem into a single itemset space that preserves neighborhood similarity over all subspaces.

Preservation of neighborhood distances has been studied in the area of Shared Nearest Neighbor (SNN) distances for traditional full space clustering with SNN [55, 33, 48] or with reverse-kNN [103]. Our transformation shares the idea of considering neighborhoods. We, however, are more general than SNN methods as we consider *multiple* views over neighborhoods in our transformation. This allows us to detect lower-dimensional clusters missed by SNN in the full attribute space.

As we incorporate multiple views into one, consensus methods are related. A naive approach is to cluster each of the views, and then use clustering ensemble methods [36, 99, 34] to derive a consensus solution. We include this as a baseline in our experiments. Such clustering ensemble techniques, however, focus on the combination of different results and do not address the high dimensionality problem systematically. In contrast to the consensus approach, we start at one-dimensional projections and systematically search for the correlated dimensions by itemset mining on the transformed data space. This allows us to find multiple overlapping clusters in different subspaces, in contrast to the single clustering obtained by consensus techniques.

## 2.3 Cartification

In this section we explain our transformation from a high dimensional numeric space into a transaction database. In particular we show how this transformation preserves neighborhood information by co-occurrence of objects.

### 2.3.1 Basic Notions

Our transformation turns the neighborhood information of a *high dimensional data space* into a *transaction database*.

INPUT: HIGH DIMENSIONAL NUMERIC DATABASE

Let $\mathcal{A} = \{A_1, \ldots, A_d\}$ be a set of $d$ dimensions. A data object, $o$, is defined as a tuple of values over $\mathcal{A}$. A $d$-dimensional database, $\mathcal{DB}$, is a collection of $n$ data objects such that every object in $\mathcal{DB}$ is represented by a $d$-dimensional vector $\mathbf{o} \in \mathbb{R}^d$. $o_i$ represents the value of $\mathbf{o}$ for the attribute $A_i$.

Usually, similarity is assessed by distance functions such as the Euclidean distance between a pair of objects $\mathbf{o}, \mathbf{p} \in \mathcal{DB}$:

$$dist_{\mathcal{A}}(\mathbf{o}, \mathbf{p}) = \sqrt{\sum_{A_i \in \mathcal{A}} (o_i - p_i)^2}$$

The curse of dimensionality [20] intuitively states that if the number of dimensions increase, distances between objects grow more and more alike. Formally:

$$\lim_{d \to \infty} \frac{\max_{\mathbf{p} \in \mathcal{DB}} \delta_{\mathcal{A}}(\mathbf{o}, \mathbf{p}) - \min_{\mathbf{p} \in \mathcal{DB}} \delta_{\mathcal{A}}(\mathbf{o}, \mathbf{p})}{\min_{\mathbf{p} \in \mathcal{DB}} \delta_{\mathcal{A}}(\mathbf{o}, \mathbf{p})} = 0$$

Thus, local neighborhood becomes meaningless for a large number of dimensions. Our transformation tries to preserve the local neighborhoods inside clusters that show high similarity. Therefore, we use multiple views on the data by simply using different similarity measures for each view.

We use the notation of $\mathcal{M}$ as a set of different measures to induce all of these views. An individual measure $m$ is simply a function that represents the similarity relation between pairs of objects. For example, we use $m_S$ for a measure over arbitrary attribute set $S \subseteq \mathcal{A}$, i.e., $m_{\mathcal{A}}(\mathbf{o}, \mathbf{p})$ represents the similarity of objects $\mathbf{o}$ and $\mathbf{p}$ over all attributes. Since we exploit the relative similarities of objects, a measure can be a dissimilarity measure (e.g. Euclidean distance), or a similarity measure

(e.g. Jaccard similarity), as long as the objects can be ordered by their pairwise similarity. Intuitively, one can think of $m$ as the projection scheme, which, obviously, can model any arbitrary transformation of the data.

OUTPUT: TRANSACTION DATA REPRESENTATION
For a set of items $\mathscr{I}$, an *itemset* $X$ is defined as $X \subseteq \mathscr{I}$. A transaction $t = (tid, X)$ is a pair of unique transaction id and an itemset. A transaction database $\mathscr{T}$ is a set of transactions over $\mathscr{I}$. The *support* of an itemset $X$ in database $\mathscr{T}$ is the number of transactions in $\mathscr{T}$ in which $X$ occurs, i.e.,

$$supp_{\mathscr{T}}(X) = |\{t \in \mathscr{T} \mid X \text{ occurs in } t\}|.$$

Support satisfies the monotonicity property [7], i.e.

$$Y \subseteq X \Rightarrow supp(Y) \geq supp(X),$$

An itemset is called *frequent* if its support is larger than some user-defined threshold called the *minimum support* or *minsup* for short.

A cartification database $\mathscr{C}$ is a special type of transactional database that has ordered transactions. Cartification is explained in detail in the following section.

### 2.3.2 Formal Transformation

Cartification transforms the original relational dataset into a coalition of different views on the data, such that each of them contributes with the neighborhood information of a given similarity measure.

**Definition 2.1** (View of an object)**.** *Let $\mathbf{o}, \mathbf{p}_i \in \mathscr{DB}$, $|\mathscr{DB}| = n$ the number of objects in $\mathscr{DB}$, and $m$ a dissimilarity measure. The cart $C_m(\mathbf{o})$ of object $\mathbf{o}$ is a tuple of its neighbors, ordered descending by their similarity to $\mathbf{o}$. Formally,*

$$C_m(\mathbf{o}) = (\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n)$$

*provided that,*

$$m(\mathbf{o}, \mathbf{p}_i) \leq m(\mathbf{o}, \mathbf{p}_j) \iff i < j \quad .$$

Neighborhood information on objects $\mathbf{p}$ that are far away from $\mathbf{o}$, contributes very little towards the discernibility of local cluster structures, therefore, far away neighbors can be excluded from the carts. That is, carts can be trimmed to include just the first $k$ elements. Note that the first $k$ elements of the cart represent the *top-k Nearest Neighbors* (*kNN*) of object $\mathbf{o}$ w.r.t a similarity measure $m$. Hence, a cart reflects the local neighborhood of a single object by using the perspective of the given measure $m$. It provides a specific view on the data. Intuitively, one can think of a projection of the database with $m$ as the projection scheme. Obviously, $m$ can model any arbitrary transformation of the data. This is used to capture one specific view on the neighborhood of each object. For the entire database we model the view of a measure as follows.

**Definition 2.2** (View of a measure)**.** *A local view $\mathscr{C}_m$ is a reflection of neighborhood information from the view of a single similarity measure $m$:*

$$\mathscr{C}_m = \bigcup_{\mathbf{o} \in \mathscr{DB}} C_m(\mathbf{o})$$

Using the transformation of only one measure would be too restrictive, as it is the case with the single projection of typical dimensionality reduction techniques [86, 63] or feature selection methods [30, 89]. In contrast to this, we employ multiple views to collect information about neighborhoods. Intuitively, we use multiple projections instead of a single one. However, again this can be more general space transformations, depending on the used measures $\mathscr{M}$.

**Definition 2.3** (Transformed database)**.** *Let $\mathscr{M}$ be the set of measures, and $\mathscr{C}_m$ a* local view *of the similarity measure $m$, then a cartified database $\mathscr{C}$ is defined as*

$$\mathscr{C} = \bigcup_{m \in \mathscr{M}} \mathscr{C}_m$$

After the transformation, neighbors of each object for each measure become clearly visible on their corresponding carts. Aggregating this neighborhood information from a selection of measures will reveal the cluster structures. We utilize the intrinsic property of neighborhoods, that is, the strong dependence of objects with high similarity inside a cluster is reflected in their co-occurrence in both their own neighborhoods, as well as in the neighborhoods of objects close-by.

### 2.3.3 Co-occurrence of objects

Frequent occurrence of an object set $X$ in $\mathscr{C}$ indicates that these objects are often spotted in the same neighborhood together in the original space. That is, they are likely to be related. Furthermore, this effect is added to by co-occurrence in different views. Every measure reflects a different set of relations in the data and cartification extracts the neighborhood information for each similarity measure separately. As such, extra measures add information; the accuracy of individual measures is not disturbed by irrelevant views.

**Definition 2.4** (co-occurring object set)**.** *The occurrence of an object set $X$ is the number of carts $C \in \mathscr{C}$ that are a superset of $X$. Formally,*

$$\mathit{occurrence}(X) = |\{C \mid X \subset C, C \in \mathscr{C}\}|$$

*A cluster $X$ is then defined as a set of co-occurring objects w.r.t. parameter minsup:*

$$\mathit{occurrence}(X) \geq \mathit{minsup}$$

We measure the occurrence of an object set $X$ by simply counting the co-occurrences of these objects in any cart in our cartified database $\mathscr{C}$. This can be considered as an implicit similarity assessment on the objects in $X$. Clustered objects have to share a large amount of objects in their respective neighborhoods. Hence they show high mutual similarity to each other. Please note that this clustering criterion can be considered as a generalization of Shared Nearest Neighborhood (SNN) clustering [55].

Essentially, SNN is a simplified version of our co-occurrence analysis. In contrast to SNN, we use multiple local views, which in turn better grasp the structures that exist only in data projections. Furthermore, we keep the order of the neighbors, that can be used to improve the following analysis steps after our data transformation,

e.g., by interestingness measures based on frequency or locality within carts. SNN simply uses the neighborhood of objects in the full space as a single view on the data. Thus, SNN can be written as a specialized version of our transformation:

$$SNN(\mathbf{o},\mathbf{p}) = |C(\mathbf{o}) \cap C(\mathbf{p})|$$

According to SNN-clustering [55], objects in a cluster has high mutual similarity $SNN(o,p) \geq k_t$, i.e., each object pair exceeds a similarity threshold $k_t$. In our generalized notion such a cluster is detected as a co-occurring object set $X$ with

$$occurrence(X) \geq minsup = k_t \quad .$$

Given this mapping to SNN [55], we can detect all clusters according to this cluster definition. However, we are more general than SNN-clustering. We overcome limitations of SNN. Since we use multiple views, we are not limited to the full space using $m_{\mathscr{A}}(\mathbf{o},\mathbf{p})$ as the only measure. Our transformation, on the other hand, can exploit lower dimensional neighborhood information and so circumvent the curse of dimensionality of considering all dimensions at once.

EXAMPLE: Suppose a high dimensional database having a set of objects $X$ form a cluster in dimensions $A_1, A_2, A_3$ but not in dimensions $A_4, A_5, A_6$. The hidden cluster will become a frequent pattern in the views from the measures that capture the data distributions in the first three dimensions. Although attributes $A_4, A_5, A_6$ do not support this cluster, e.g., show scattered data distribution for the object set $X$, they will not affect the support from $A_1, A_2, A_3$, therefore the cluster will remain detectable. On the contrary, in the original space, taking all the dimensions into account would distort the distances, obstructing the detection of the cluster.

FORMALLY:
Clustered objects $X = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ will be clearly separated from residual objects $\mathbf{q} \notin \mathscr{DB} \setminus X$ in at least some dissimilarity measures of $\mathscr{M}$:

CARTIFIED SPACE:

$$occurrence(X) \geq minsup \overset{\text{(Def. 2.4)}}{\iff} \exists V \subseteq \mathscr{DB} \times \mathscr{M} \wedge |V| \geq minsup$$

$$\forall (\mathbf{o}, m) \in V : X \subset C_m(\mathbf{o}) \overset{\text{(Def. 2.1)}}{\iff} \exists V \subseteq \mathscr{DB} \times \mathscr{M} \wedge |V| \geq minsup$$

$$\forall (\mathbf{o}, m) \in V : m(\mathbf{o},\mathbf{p}_j) < m(\mathbf{o},\mathbf{q}) \quad \forall \mathbf{p}_j \in X \wedge \mathbf{q} \notin C_m(\mathbf{o})$$

Thus, there exists a set of carts that reflect the similarity of all clustered objects $X$ in some of the views $\mathscr{M}$. The similarity of a cluster is preserved in these views, i.e., objects $\mathbf{p}_j$ are clustered together in the neighborhood of a central object $\mathbf{o}$, while other objects $\mathbf{q}$ are clearly separated from this central object. Please note that there might be other contradicting views, however, these not have a negative effect on the number of co-occurrences of $X$. Therefore, similarity and neighborhoods are preserved. In contrast to this, the original data space is effected by irrelevant dimensions, and thus, does not reveal any cluster structures.

ORIGINAL SPACE:

$$\forall \mathbf{o},\mathbf{p} \in X \wedge \forall \mathbf{q} \notin X \; : \; m_{\mathscr{A}}(\mathbf{o},\mathbf{p}) \approx m_{\mathscr{A}}(\mathbf{o},\mathbf{q})$$

Due to the curse of dimensionality [20, 19], all objects have similar distances to each other, thus, no cluster structure can be observed.

### 2.3.4 Instantiation: One-Dimensional Data Projections

In order to preserve neighborhoods and circumvent the curse of dimensionality [20, 19], we follow the idea of lower dimensional projections as meaningful instantiation of different measures. Lower dimensional projections have shown to be effective for query processing [47] and subspaces clustering [84] in high dimensional databases. They capture neighborhood information w.r.t. different views on the data and provide us the required similarity measures on the database:

**Definition 2.5** (A Projection Measure)**.**
*Given a set of numeric dimensions $S \subseteq \mathscr{A}$, we use the projection of $\mathscr{DB}$ specified by the following measure:*

$$m_S(\mathbf{o},\mathbf{p}) = dist_S(\mathbf{o},\mathbf{p}) = \sqrt{\sum_{A_i \in S} (o_i - p_i)^2}$$

W.l.o.g. we use Euclidean distance as dissimilarity measure, however, any other distance restricted to a subset of attributes will work here as well. Using this definition of projection measures, the set of all measures $\mathscr{M}$ is simply a collection of different projections of the data. Choosing this set of projections can be instantiated by correlation analysis or subspace search methods [25, 57, 59].

However, we will show that such prior analysis and exponential overhead is not required in our case. As minimum requirement the similarity measures have to identify neighborhoods for each individual dimension only. Then, possible correlation in higher-dimensional projections is assessed by our co-occurrence.

We simply use $\mathscr{M} = \{m_{A_1}, \ldots, m_{A_d}\}$ as initial set of measures. Each measure $m_{A_i}$ represents a one-dimensional projection of the data w.r.t. attribute $A_i$. It is simply computed by the Euclidean distance in this dimension. Using these measures, we allow for similarity assessment in any projection of the data. A co-occurring object set $X$ can show mutual similarity in an arbitrary subspace $S \subseteq \mathscr{A}$:

$$\exists A_i \in S : X \subset C_{m_i}(\mathbf{o}) \quad \forall \mathbf{o} \in X \quad \Longleftrightarrow |o_i - p_i| < |o_i - q_i| \quad \forall \mathbf{o},\mathbf{p} \in X \wedge \mathbf{q} \notin X$$

Please note that we might consider more complex similarity measures as alternative to the current one-dimensional instantiation, e.g. by using correlation analysis or subspace search [25, 59, 79]. Our current solution abstracts from such instantiations and provides the general processing for any set of similarity measures $\mathscr{M}$.

Let us now give an example of how we can identify central elements of a cluster. Assume we are given the two-dimensional dataset with $\mathscr{A} = \{x, y\}$ and data objects as shown in Figure 2.1. For this example, as a cartification strategy we use $k$-nearest neighbors on separate dimensions to create carts. Formally, inverse of Euclidean distance on separate dimensions are our similarity measures,

$$m_{\{x\}}(\mathbf{o},\mathbf{p}) = |o_x - p_x|$$
$$m_{\{y\}}(\mathbf{o},\mathbf{p}) = |o_y - p_y|$$

where $o_x$ and $o_y$ are the $x$ and $y$ attribute values of the data object $\mathbf{o}$, respectively. Using these measures, the CARTIFY algorithm (cf. Section 2.3.5) creates a database

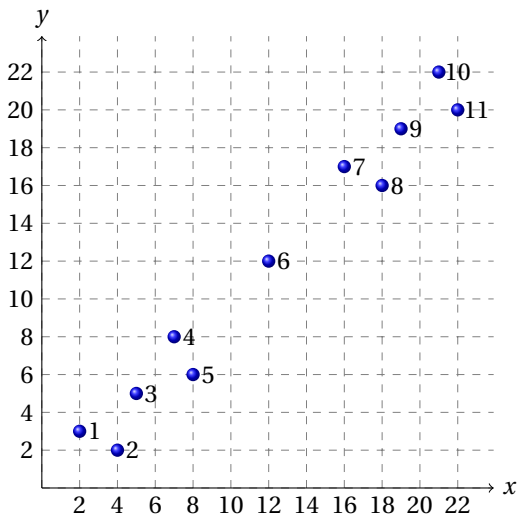



Figure 2.1: Example dataset

| $x$ | **cart** | $y$ | **cart** |
|---|---|---|---|
| 1 | {1,2,3} | 1 | {1,2,3} |
| 2 | {1,2,3} | 2 | {1,2,3} |
| 3 | {2,3,4} | 3 | {1,3,5} |
| 4 | {3,4,5} | 4 | {3,4,5} |
| 5 | {3,4,5} | 5 | {3,4,5} |
| 6 | {5,6,7} | 6 | {4,6,8} |
| 7 | {7,8,9} | 7 | {7,8,9} |
| 8 | {7,8,9} | 8 | {7,8,9} |
| 9 | {8,9,10} | 9 | {7,9,11} |
| 10 | {9,10,11} | 10 | {9,10,11} |
| 11 | {9,10,11} | 11 | {9,10,11} |

Table 2.1: Cartification of the data in Figure 2.1, for $k = 3$.

where each cart is composed of the items ordered by one dimensional distances to each item.

On this toy example, essential information regarding the formation of items can be extracted with a rather simple method. Instead of taking into account detailed ordering in the carts, we truncate each transaction to the same size, $k$, disregard the inner order, and treat each transaction as a set.

Note that, this process converts the cartified database to a regular transactional database where each transaction is the $k$-nearest neighbors of each item on each

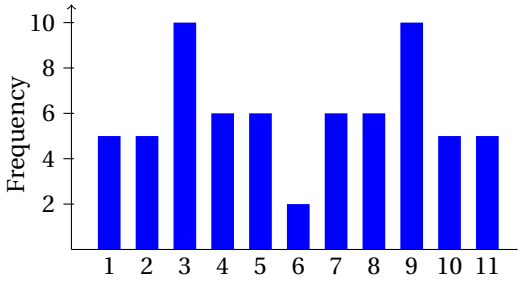



Figure 2.2: Item Frequencies for $k = 3$

**Algorithm 1** The CARTIFY Algorithm

**Input:** A database $\mathscr{DB}$, set of similarity measures $\mathscr{M}$
**Output:** The cartified database $\mathscr{C}$ of $\mathscr{DB}$ over $\mathscr{M}$

1: $\mathscr{C} \leftarrow \emptyset$
2: **for each** $m \in \mathscr{M}$ **do**
3: $\quad \mathscr{C}_m \leftarrow \emptyset$
4: $\quad$ **for each** $\mathbf{o} \in \mathscr{DB}$ **do**
5: $\quad\quad C \leftarrow$ sort $\mathbf{p} \in \mathscr{DB}$ according to $m(\mathbf{o}, \mathbf{p})$
6: $\quad\quad \mathscr{C}_m \leftarrow \mathscr{C}_m \cup C$
7: $\quad \mathscr{C} \leftarrow \mathscr{C} \cup \mathscr{C}_m$
8: **return** $\mathscr{C}$

dimension.

There is only one parameter needed for the transformation: the truncation threshold $k$, i.e., the number of the $k$ nearest neighbors that will be added to each cart. For example, the resulting database for $k = 3$ is shown in Table 2.1. The first two columns show the cartification for the $x$-dimension, and the second two columns for the $y$-dimension. Every row corresponds to the cart generated from one of the data points. For instance, for point 3, the three nearest neighbors in the $x$-dimension are points 2, 3, and 4, while for the $y$-dimension these are 1, 3, and 5.

In Figure 2.2, we plot the frequencies of all items (points) in this cartified database. The plot shows that are two points, 3 and 9, have high frequencies. As Figure 2.1 shows, these points are in the centers of the clusters $\{1,2,3,4,5\}$ and $\{7,8,9,10,11\}$ respectively. Additionally, point 6 has a very low frequency which corresponds to the point being an outlier w.r.t. all other points in the figure.

### 2.3.5 Algorithm and Complexity Analysis

With the above, we can now formally introduce the CARTIFY algorithm. Suppose we are given a database $\mathscr{DB}$ over attributes $\mathscr{A}$ and a set of similarity measures $\mathscr{M}$, defined over $S \subseteq \mathscr{A}$ as given in Definition 2.5. The pseudo-code is given as Algorithm 1.

For each similarity measure (line 2) and for each object $\mathbf{o}$ in $\mathscr{DB}$ (line 4) a cart $C$ is created. Each cart is an ordered list of the objects in $\mathscr{DB}$, sorted descending by their similarity to $\mathbf{o}$. The collection of all carts together forms the final cartified

database, $\mathscr{C}$ (line 8). During the cartification process, local views for each measure $\mathscr{C}_{m_i}$ are kept as well (line 6).

Theoretically, space requirement of CARTIFY algorithm is $n \times n \times |\mathscr{M}|$, where $n$ is the number of items and $|\mathscr{M}|$ is the number of measures. However, for practical applications, it is not required to cartify the whole database. Instead, carts can be trimmed to a constant value to limit the size of the database.

Worst case time complexity of CARTIFY is $O(n^2 \log(n)|\mathscr{M}|)$. For some measures, e.g., 1D projections, sorting can be done per measure, in the outer loop, which results in a time complexity of $O(n \log(n)|\mathscr{M}|)$. Moreover the processing of the cartification can easily be parallelized over shared memory or distributed systems.

## 2.4 An Application of Cartification to Subspace Clustering

As an example of how the information in the transformed data can be used, in this section we introduce CARTICLUS. CARTICLUS, short for cartification-based subspace cluster detection, is an algorithm for mining subspace clusters by means of a straightforward itemset mining based approach. Even though relatively simple method, the experiments in Section 2.5 show that, it outperforms competing approaches on high dimensional data.

Measures that are used to transformation the original datasets are just Euclidean distances on one-dimensional projections. Thus, every local view, $\mathscr{C}_m$, represents the neighborhood information of an individual dimension. After the transformation, carts are trimmed to a user-specified length $k$. To detect subspace clusters we then stochastically generate $n$ maximal frequent itemsets. After a simple post-processing, we report the sufficiently frequent object sets as subspace clusters. Subspaces for a cluster are the dimensions that object set has non-zero support.

There are two important parameters for the process: $k$, the size of each cart and *minsup*, the minimum number of co-occurrences of an object set in the cartified database $\mathscr{C}$. While parameter $k$ influences the cluster size and robustness against noise, parameter *minsup* regulates the purity of the found clusters.

On an ideal clustering dataset, i.e., clearly separated clusters in every dimension, every object in the same cluster is closer to the objects in the same cluster than the objects outside of the cluster. Thus, after trimming the carts to the size of a cluster, only objects of the same cluster should remain in the carts. The cluster can hence be identified by finding the object set occurring in the carts corresponding the cluster objects, over each of the dimensions relevant to the cluster. We can reformulate these parameters into terms of expected cluster size and expected dimensionality, which are more standard parameters in clustering research.

$$\begin{aligned} k &= |C| \times k\text{-}ratio \\ minsup &= |C| \times d \times minsup\text{-}ratio \end{aligned}$$

where $|C|$ is the expected size of a cluster and $d$ is the expected number of dimensions, both of which are the user-defined parameters. Since most data is noisy, for convenience, we introduce *k-ratio* and *minsup-ratio* as relaxation parameters for improved stability. Experiments show that *minsup-ratio* can be fixed to a value of 0.75 and *k-ratio* can be selected as 1.25, 1.5, 1.75, or 2 depending on the amount of noise of the data (cf. Section 2.5.4). These ratio values can be regarded as a

rule of thumb. We give the pseudo-code of CARTICLUS as Algorithm 2. It has two input parameters: $k$, the size of each cart and *minsup*, the minimum number of co-occurrences of an object set in the cartified database $\mathscr{C}$. There are two important remarks. First, since a sample of the maximal frequent itemsets (MFIs) is enough, CARTICLUS efficiently mines a sample of the MFIs instead of generating all of them. Lines 3 to 12 of Algorithm 2 shows this mining process. Random seed selection (line 4) is biased toward selecting a not used item and random selection of the object for expansion (line 7) is biased toward the objects that will increase the support most. To increase robustness, object sets that are smaller than a certain length are discarded (line 10–11). To limit the time of the random mining process the search is stopped after a certain number of iterations line 12. Another important remark is the post-processing of MFIs. Our experiments show that MFIs tend to identify *subsets* of clusters. In order to be able to report the key cluster structure without redundancy, CARTICLUS merges MFIs that are highly similar to each other, i.e., if they share at least *minlen* objects, lines 13–20.

CARTICLUS takes cartified database as input. Therefore, additional analysis tasks, e.g., using different parameters or even different mining algorithms, can be performed on the same cartified database, with a minimal data processing.

Every subspace cluster is expressed in terms of objects and subspaces. In order to detect the subspaces relevant for a discovered subspace cluster, CARTICLUS calculates the supports from the subspaces for the cluster. Subspaces that have a non-zero support are reported as the subspaces of the cluster (line 20).

## 2.5 Experiments

In this section we discuss empirical results and demonstrate the capabilities of cartification for subspace cluster detection.

**Competitors:** We compare against five well-known clustering algorithms: (1) *Proclus* [5] is a projected clustering algorithm. According to the recent evaluation study on subspace clustering [78], it can be considered to be one of the best performing subspace clustering approaches. (2) *K-Means* [69] over all dimensions (full space), (3) *RP-KM* is K-Means on a set of random projections (RP), (4) *PCA-KM* is K-Means on a transformed data space by Principal Component Analysis (PCA), and (5) *CSPA* [99] is an ensemble clustering algorithm. We use 10 runs of K-Means on separate dimensions to produce input clusterings and combine them with CSPA. To improve the stability of the clusters, we use k-means++ [15] instead of the original K-Means.

**Parameters:** Since each of the methods we consider are stochastic, we run each setup for 10 times and report the average results. For all competitors, we try to optimize the respective parameter settings. For example, we use the number of expected dimensions per cluster as input for Proclus and RP-KM. In case of K-Means we provide the true number of clusters as input parameter. As default parameter setting of our approach, we set *minlen* parameter to half of to the *expected cluster size* parameter. Since it is just a minimum, this setting increased the robustness of the algorithm without limiting the ability of finding the whole clusters. The parameter $n$ is set to 100 for all of the experiments.

**Experiment Setup:** We explore the influence of dimensionality on performance,

**Algorithm 2** CARTICLUS Algorithm

**Input:** Certified database $\mathscr{C}$, minimum support *minsup*, minimum length *minlen*, number of requested object sets $n$
**Output:** Set of subspace clusters $\mathscr{X}$ as pairs of object and subspace sets

$\mathscr{X} \leftarrow \emptyset$ // *MFIs*
$\Theta \leftarrow$ all objects in $\mathscr{C}$
**repeat**
  $X \leftarrow \{\mathbf{o}\}$ // *randomly select a frequent object* $\mathbf{o} \in \Theta$
  $P \leftarrow \{\mathbf{o} \mid supp_{\mathscr{C}}(X \cup \{\mathbf{o}\}) > minsup, \mathbf{o} \in \Theta\}$
  **repeat**
    $X \leftarrow X \cup \{\mathbf{p}\}$ // *randomly select an object* $\mathbf{p} \in P$
    $P \leftarrow \{\mathbf{o} \mid supp_{\mathscr{C}}(X \cup \{\mathbf{o}\}) > minsup, \mathbf{o} \in P \setminus X\}$
  **until** $P = \emptyset$
  **if** $|X| \geq minlen$ **then**
    $\mathscr{X} \leftarrow \mathscr{X} \cup \{X\}$
**until** $|\mathscr{X}| \geq n$ *or* maximum iterations are reached
$\mathscr{X}' \leftarrow \emptyset$ // *output clusters*
**for each** $X \in \mathscr{X}$ **do**
  **for each** $X' \in \mathscr{X}'$ **do**
    **if** $|X' \cap X| > minlen$ **then**
      $X' \leftarrow X' \cup X$
      **continue** with next $X$
  $\mathscr{X}' \leftarrow \mathscr{X}' \cup (X', \{m \mid supp_{\mathscr{C}_m}(X) > 0\})$
**return** $\mathscr{X}'$

noise awareness and relevant subspace detection capabilities of the algorithms. We use both synthetically generated data and publicly available real world benchmark data, as also used in a recent evaluation study [78]. Properties of the databases are shown in Table 2.2. All of these databases are provided online,[2] together with all algorithms, which are implemented in Java. Each run of each algorithm was completed under 2 minutes on a standard PC with Intel Core-i7 processor and 8 GB of memory running GNU/Linux. Note that, since the cartification transformation has to be done only once we run CARTICLUS on cartified data.

**Evaluation Measures:** For quality assessment, we use the ground truth of the synthetic data and class labels of the real world data. To compare fairly with other methods, we use two well-established quality measures: *F1* and *E4SC*, which have also been used in a variety of subspace cluster publications [76, 75, 77, 78, 42]. *F1* measures the harmonic mean of *precision* and *recall* between a set of hidden clusters and the set of detected clusters. For a given cluster $C$ and a detected cluster $C'$, *precision* is the ratio of common objects between $C$ and $C'$ to the size of $C'$, while

[2] http://adrem.uantwerpen.be/cartification

*recall* is the ratio of the size of the intersection to the size of $C$. Formally,

$$precision(C, C') = \frac{|C \cap C'|}{|C'|}$$

$$recall(C, C') = \frac{|C \cap C'|}{|C|}$$

$$F1(C, C') = \frac{2 \times recall(C, C') \times precision(C, C')}{recall(C, C') + precision(C, C')} \tag{2.1}$$

Since the clusters has no labels, unlike a classification problem, we have to match the detected clusters to the given clusters. We follow the common practice in the literature and match the clusters that produce the highest F1 score. To keep the assessments fair and accurate, we use two intermediate measures: We first match every found cluster to a known cluster, compute their pairwise F1 scores, and then, take the average of these F1 scores. We call this measure *F1-Precision*, as it measures how *accurate* the detected clusters are. Similarly, we use *F1-Recall* measure. It matches every known cluster to a found clusters and take the average of the pairwise F1 scores. And lastly, we report the harmonic mean of *F1-Precision* and *F1-Recall* scores as the F1 score. Let $\mathbf{C}$ and $\mathbf{C}'$ respectively be a set of given clusters and a set of detected clusters, we compute the F1 scores as follows:

$$F1\text{-}Precision(\mathbf{C}, \mathbf{C}') = \frac{1}{|\mathbf{C}'|} \times \sum_{C' \in \mathbf{C}'} \max(F1(C, C')) \tag{2.2}$$

$$F1\text{-}Recall(\mathbf{C}, \mathbf{C}') = \frac{1}{|\mathbf{C}|} \times \sum_{C \in \mathbf{C}} \max(F1(C, C')) \tag{2.3}$$

$$F1(\mathbf{C}', \mathbf{C}) = \frac{2 \times F1\text{-}Recall(\mathbf{C}, \mathbf{C}') \times F1\text{-}Precision(\mathbf{C}, \mathbf{C}')}{F1\text{-}Recall(\mathbf{C}, \mathbf{C}') + F1\text{-}Precision(\mathbf{C}, \mathbf{C}')} \tag{2.4}$$

In addition to *F1*, which only measures the object-wise quality of a cluster, we use *E4SC* to assess the quality of detected subspaces as well as the quality of the detected objects. *E4SC* measures the subspace-aware overlap between clusters by applying the F1 score on the attribute level [42]. *E4SC* between two subspace clusters is computed as follows:

1. Clusters are extended by finding the Cartesian product of objects and attributes. For a subspace cluster $SC = (C, S)$, extended subspace cluster is $e(SC) = C \times S$.
2. Each object in each attribute is treated as a new *micro*-object.
3. F1 score is computed between these extended clusters using the *micro*-objects.

Formally, *E4SC* between two subspace clusters $SC$ and $SC'$ is

$$E4SC(SC, SC') = F1(e(SC), e(SC'))$$

For example let two subspace clusters be $SC_1 = (\{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_5\}, \{A_1, A_2, A_3, A_4\})$ and $SC_2 = (\{\mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_5, \mathbf{o}_6\}, \{A_2, A_3, A_4, A_5\})$. As shown in Figure 2.3; when we extend the cluster $SC_1$, we have

$$e(SC_1) = \{\mathbf{o}_1^{A_1}, \mathbf{o}_2^{A_1}, \mathbf{o}_3^{A_1}, \mathbf{o}_4^{A_1}, \mathbf{o}_5^{A_1}, \mathbf{o}_1^{A_2}, \cdots, \mathbf{o}_5^{A_2}, \mathbf{o}_1^{A_3}, \cdots, \mathbf{o}_5^{A_5}\}$$

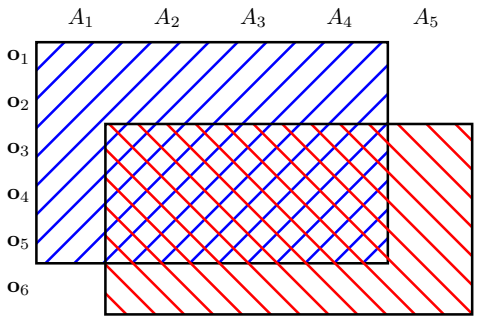



Figure 2.3: E4SC between two subspace clusters is computed by measuring the overlap of both objects and dimensions.

*E4SC* between $SC_1$ and $SC_2$ is

$$\begin{aligned} E4SC(SC_1, SC_2) &= F1(e(SC_1), ec(SC_2)) \\ &= 2 \times \frac{|e(SC_1) \cap e(SC_2)|}{|e(SC_1)| + |e(SC_2)|} \\ &= 2 \times \frac{3 \times 3}{5 \times 4 + 4 \times 4} = 2 \times \frac{9}{20 + 16} = 0.5 \end{aligned}$$

Following the computation in (2.4), *E4SC* score between two *sets* of clusters is computed as the *F1* score between the sets of extended clusters:

$$E4SC(\mathbf{SC}, \mathbf{SC}') = F1(e(\mathbf{SC}), e(\mathbf{SC}'))$$

### 2.5.1 Dimensionality

In order to assess the subspace cluster detection capabilities of the methods w.r.t. increasing number of dimensions, we conduct experiments on datasets where the number of objects and the average ratio of relevant dimensions per cluster stay constant while the number of dimensions increases. Datasets D10, D25 and D75 have 10, 25 and 75 dimensions, respectively.

The results for these experiments are shown in Figures **??** and **??**. The quality of the clusters found by CARTICLUS is not affected by the number of dimensions. This is expected, because more dimensions only add more views for our method without corrupting the existing views. Therefore, our transformation truly preserves the neighborhood information and its cluster detection capability does not degrade with an increase in the dimensionality.

### 2.5.2 Noise Awareness

Evaluation of noise awareness is done using a set of datasets that contain a fixed number of clusters and dimensions but include different ratios of noise. Datasets N30, N50 and N70 have 30%, 50% and 70% of noise objects, respectively.

| | # of Objects | # of Dimensions | Cluster Size | Dims per Cluster | Noise Ratio |
|---|---|---|---|---|---|
| D10 | 1595 | 10 | 150 | 5–8 | 9% |
| D25 | 1595 | 25 | 150 | 15–20 | 9% |
| D75 | 1596 | 75 | 150 | 45–61 | 9% |
| N30 | 2071 | 20 | 150 | 12–16 | 30% |
| N50 | 2900 | 20 | 150 | 12–16 | 50% |
| N70 | 4833 | 20 | 150 | 12–16 | 70% |
| ND10 | 1050 | 20 | 100 | 3–6 | 5% |
| ND50 | 1050 | 60 | 100 | 4–7 | 5% |
| ND200 | 1050 | 210 | 100 | 4–6 | 5% |
| Vowel | 990 | 11 | 99 | 11 | 0 |
| Shape | 160 | 17 | 11–20 | 17 | 0 |
| Alon | 62 | 2000 | 22–40 | ? | 0 |

Table 2.2: Properties of the datasets

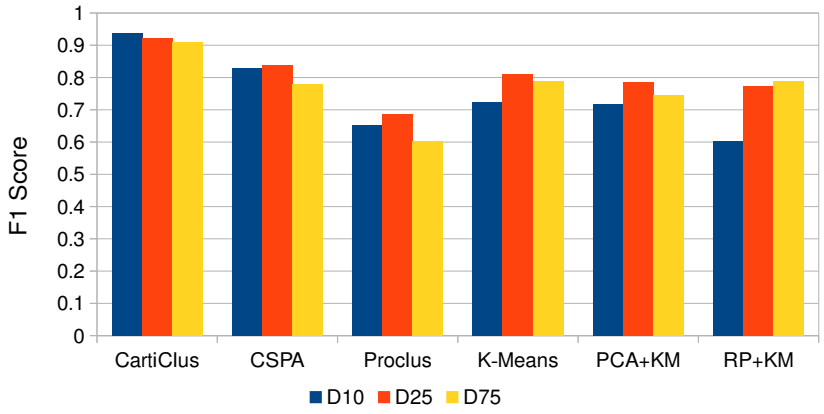



Figure 2.4: F1 Scores for datasets with increasing number of dimensions. In all of the datasets dimensionality of clusters are proportional to the total number of dimensions.

Results in Figures 2.6 and 2.7 show that CARTICLUS effectively detects high quality clusters, even if they exist in only 30% of the data. Since random objects are not nearest neighbors of most clustered objects in many views of the data, they are not expected to be in many carts, therefore, the support of object sets that they are part of are low. While CARTICLUS detects structure it ignores objects with low supports, i.e., noise. In contrast to this robust behavior, we observe the quality of

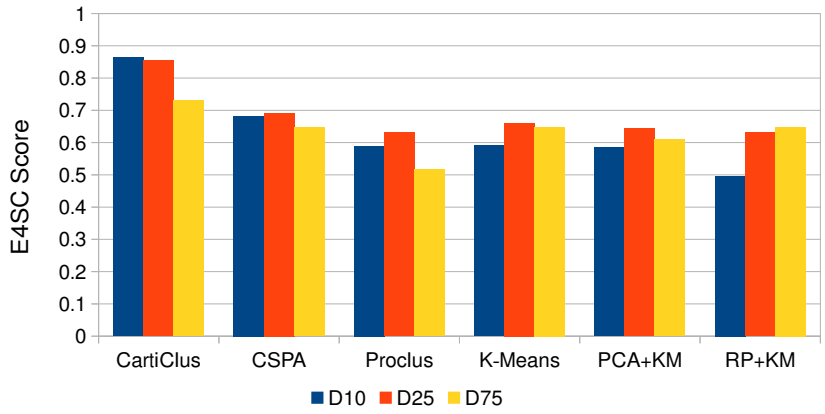



Figure 2.5: E4SC Scores for datasets with increasing number of dimensions.

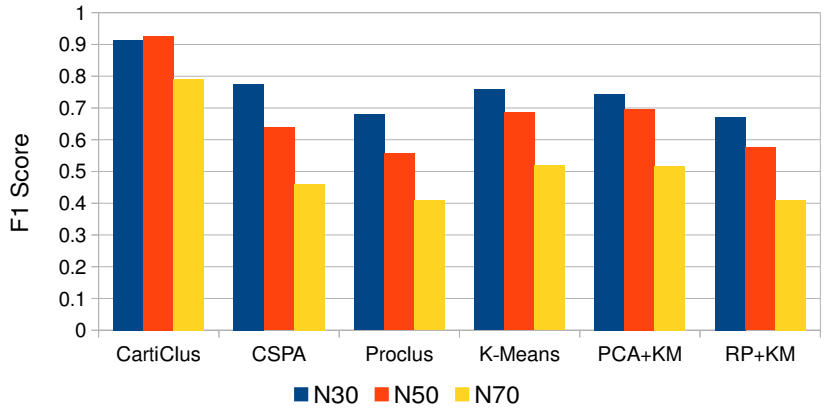



Figure 2.6: F1 Scores for datasets with increasing amounts of noise. 30%, 50%, and 70% of the objects in these datasets do not form cluster structures.

the clusters found by K-Means, Proclus, and CSPA degenerate with increasing noise.

### 2.5.3 Subspace Detection

To evaluate the correct detection of subspaces, we generate datasets with 10 hidden clusters having 2 to 7 relevant dimensions and add additional random dimensions. The datasets ND10, ND50, and ND200 have respectively 10, 50 and 200 irrelevant dimensions. All of these datasets include additional 5% random objects as noise.

Results in Figures 2.8 and 2.9 clearly show the benefits of aggregating information from multiple views of the data. CARTICLUS and CSPA can extract and efficiently use the information from the subspaces that contain structures to overcome the

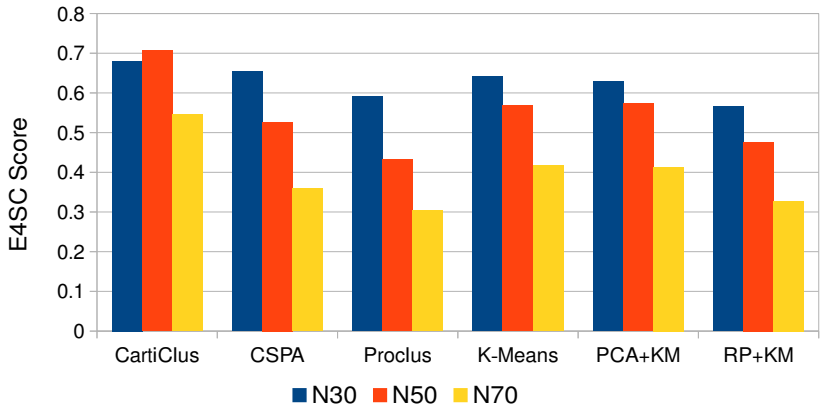



Figure 2.7: E4SC Scores for datasets with increasing amounts of noise.

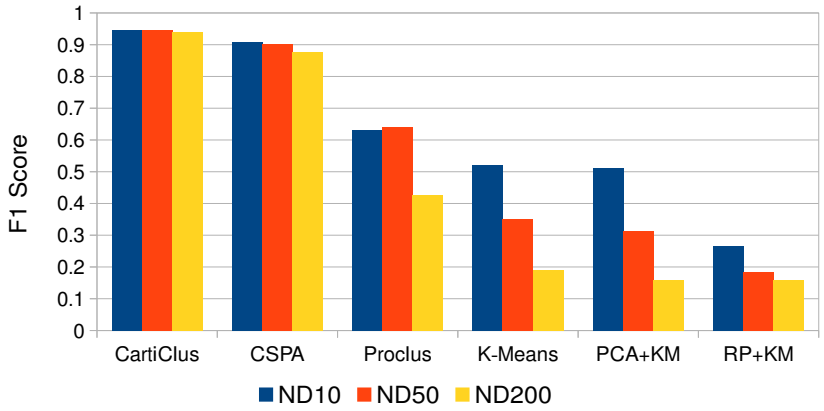



Figure 2.8: F1 scores of the methods on datasets with an increasing number of irrelevant dimensions. 10 clusters of 3 to 6 dimensions are hidden 10, 50, and 200 additional uniformly distributed dimensions.

negative effects of irrelevant dimensions. Thanks to the stricter cluster definition of CARTICLUS, i.e., *co-occurrence* instead of pair similarities, and noise detection capabilities, it can detect better clusters than CSPA. Proclus can ignore the the irrelevant dimensions up to some level. However, its greedy approach combined with random initialization cannot compensate the degrading effects of the increasing number of irrelevant dimensions. Since feature selection techniques treat all dimensions equally, increasing the number of irrelevant dimensions renders them useless. Note that, since *CSPA* cannot report subspaces for clusters, its *E4SC* score decrease with the number of dimensions.

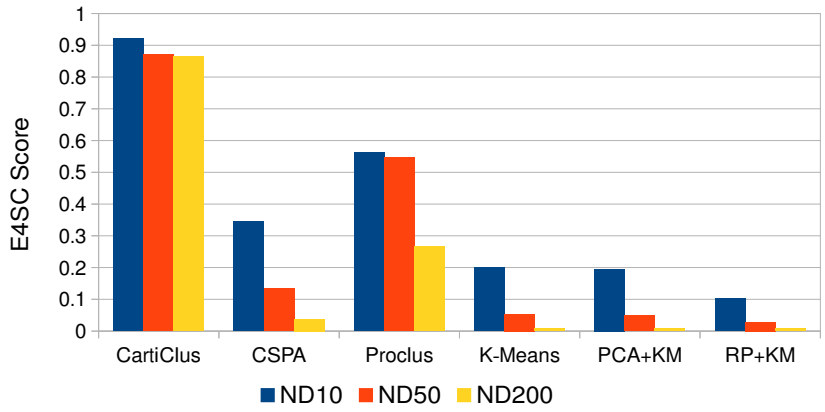



Figure 2.9: E4SC scores of the methods on datasets with an increasing number of irrelevant dimensions.

### 2.5.4 Parameter Sensitivity

The results of the experiments evaluating the sensitivity to the values of *minsup-ratio* and *k-ratio* with regard to the different ratios of noise objects and noise dimensions are shown as Figure 2.10. We see that overall *k-ratio* is robust, yet, as explained in Section 2.4, the optimal value depends on the amount of noise in the data. While decreasing the *k-ratio* decrease the size of the detected clusters, and hence, increase purity, higher values for *k-ratio* extracts larger clusters. We further see that *minsup-ratio* is very robust in terms of cluster purity, while increasing its value decreases the size of clusters, which, in turn, results in lower *F1* scores.

### 2.5.5 Real World Data

Next we evaluate the performance of CARTICLUS on real world data. As real data, we here consider three datasets. First, the *Vowel* recognition dataset consists of 11 attributes and 990 objects, spread over 10 classes of 99 instances for each. Second, the *Shape* recognition dataset has 160 instances over 17 attributes grouped under 9 known clusters. These two datasets are publicly available as benchmarks for subspace clustering [78]. Third, we use the publically available *Alon* Colon Cancer dataset [13] dataset. This is a gene expression dataset of 2000 attributes and one binary class for 62 tissue samples.

As for none of these datasets there exists a known ground truth for the subspaces, we here only consider F1 scores. Figure 2.11 gives the performance of the six methods over these datasets. We see that for the small *Shape* dataset CARTICLUS performs on par with its competitors. For *Vowel* and *Alon* it shows better performance. For the latter, the most high-dimensional dataset we consider, CARTICLUS obtains F1 scores that are 25% higher than *Proclus* and 10% higher than *CSPA*, respectively.

Note that in real data, cluster structures do not necessarily have to correlate with the class labels, for all methods the F1 scores in this experiment are not as high as

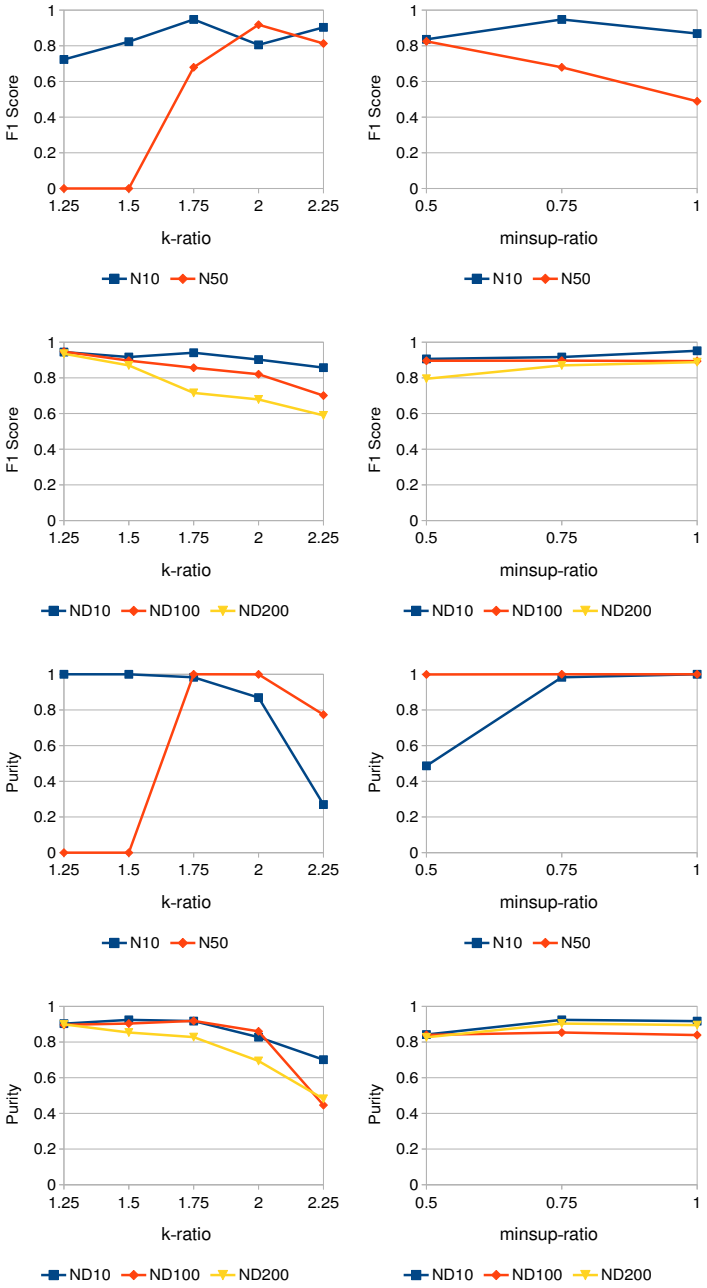



Figure 2.10: Effects of parameters over the quality of detected clusters.

for the synthetic data.

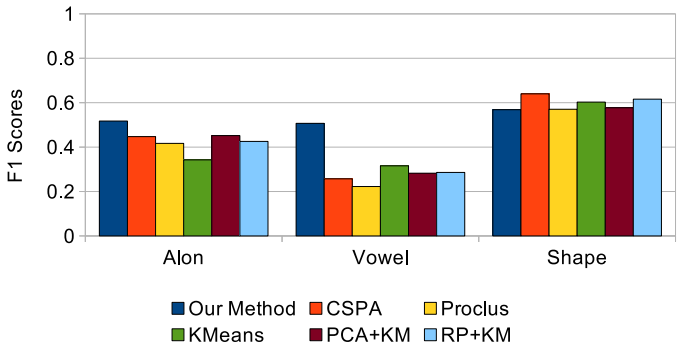



Figure 2.11: Real World Datasets

## 2.6 Conclusions

The main contribution of this chapter is *Cartification*, a novel and highly flexible framework for transforming high dimensional data into the well-studied domain of transaction data, while preserving all local neighborhood information in the respective lower-dimensional projections. Advantages include freedom in the choice of similarity measures. Our experiments show that the cartified data space maintains neighborhood information, allowing a simple itemset mining approach to outperform variety of established dedicated clustering algorithms. Moreover, the literature on mining transaction data is rich in tools to find unexpected/non-redundant relations [71] as well as very unlikely combinations [29, 105], which in turn can be used to find subspace clusters of various interests. In short, our transformation opens a wide area of applications for cartified data, in particular for those tasks that rely on neighborhood information on high dimensional data.

Future work includes the study of more advanced cluster detection techniques, as well as identifying which subspaces are most relevant to the cluster structure present in a dataset.

CHAPTER 3

# Detecting Cluster Structures by Ordered Neighborhoods

*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 chapter*[1] *we propose a cluster detection method based on the ordered neighborhoods. We study the properties of such local neighborhood information for the separability of clusters. By considering ordered neighborhoods in each dimension individually, we derive a property that allows us to detect clustered objects that show high similarity 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] This chapter is based on work published in DaWaK 2015 as "Efficient cluster detection by ordered neighborhoods" by Emin Aksehirli, Bart Goethals, and Emmanuel Müller [11].

## 3.1 Introduction

In the information era that we live in, there are 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. One of the challenging effects of this "data hoarding" is the increased number of attributes associated with each object. Unfortunately, in high dimensional data spaces, the similarity between two objects becomes meaningless [20]. This means that the clusters cannot be separated from each other by (dis-)similarity assessment in high dimensional space, which in turn pose a serious challenge for the traditional clustering algorithms [69, 91, 53]. Nevertheless, (dis-)similarity can be exploited in the individual dimensions, and clusters can be detected in the combinations of these individual 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 be achieved by looking at individual dimensions, e.g. travel and sport habits only. In both of these individual attributes customers might be clustered. Even in the combination of these two attributes, the same customer structure might be detected as clusters.*

Although similarity between high dimensional objects is not meaningful, similarity according to subsets of attributes is still meaningful. To address this issue, subspace clustering [97] 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 [5, 96], density estimation challenges [6, 56, 60, 16], many complex parameters [16, 60, 77, 75], or inefficient search through the combinatorial search space [6, 56, 10].

In contrast to these methods, we tackle the problem by exploiting ordered neighborhoods in each of the individual dimensions. These neighborhoods are used as means for cluster separability in combinations of dimensions. We prove that clusters can be detected directly from these ordered neighborhoods. Given these formal properties we can avoid common scalability pitfalls in the algorithmic computation of clusters. In particular, we propose a linear algorithm for detecting cluster structures in projections. Key characteristics of our approach are its scalability w.r.t. both database size and dimensionality, the detection of clusters of variable size, 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:

- Provide a formal analysis of ordered neighborhoods in the context of cluster separability.
- 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.

## 3.2 Ordered Neighborhoods

In a nutshell, 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 [88]. To the best of our knowledge, we are the first ones to exploit this theoretic principle for scalable cluster detection in high dimensional data.

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 3.2.1. Second, we propose a data transformation from relational data to an ordered neighborhood space and investigate its properties in Section 3.2.2. We introduce a scalable algorithm that exploits these properties for cluster detection in Section 3.3.

### 3.2.1 Clustering in Neighborhoods

For a *relational database* $\mathcal{DB}$ we represent each data object $\mathbf{o}$ in $\mathcal{DB}$ as a vector defined over attributes $\mathcal{A} = \{A_1, A_2, \cdots, A_m\}$, 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 $\mathcal{DB}^{A_i}$. Further, we use $\delta(\mathbf{o}, \mathbf{p})$ as a dissimilarity between the objects $\mathbf{o}$ and $\mathbf{p}$. $|S|$ 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 [41], dimensionality reduction [106], collaborative filtering [83] and many other techniques in various communities [32, 72]. For clustering, one has used shared nearest neighborhoods [54], neighborhoods in random projections [93], and frequent co-occurrence of

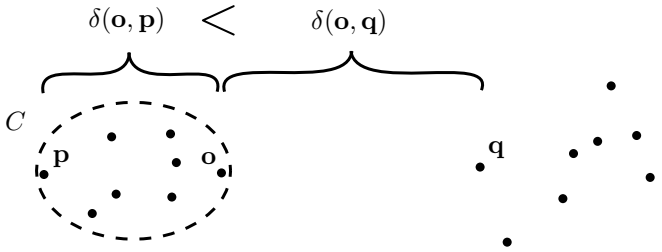



Figure 3.1: $C_i$ and $C_j$ are separable clusters

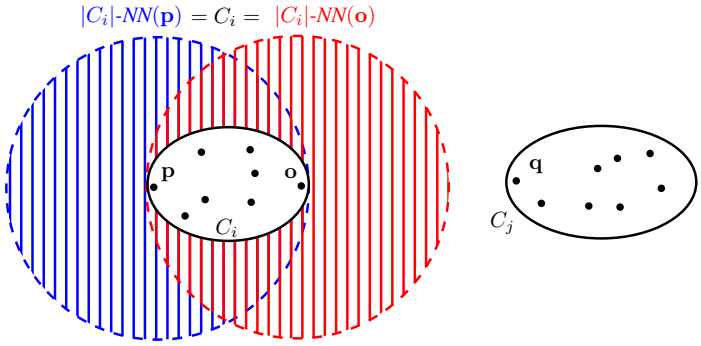



Figure 3.2: Nearest neighbors of the objects in separable clusters

neighborhoods [10]. 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 3.1** ($k$-Nearest Neighborhood ($k$-$NN$))**.** *Let $\mathbf{o} \in \mathcal{DB}$, $\delta$ a dissimilarity measure, and the $NN_k(\mathbf{o})$ be the $k^{th}$ nearest object to $\mathbf{o}$ according to $\delta$, the $k$-nearest neighborhood ($k$-$NN$) of $\mathbf{o}$ is defined as*

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

$k$-$NN(\mathbf{o})$ captures the similar objects near object $\mathbf{o}$, which we can use for cluster detection. That is, we detect a *cluster* $C \subseteq \mathcal{DB}$, which in general is a set of similar objects that are well separated from the objects in other clusters.

**Definition 3.2** (Separable Clusters)**.** *Given two clusters, $C_i$ is separable from $C_j$ if the objects in $C_i$ are more similar to each other compared to the objects in $C_j$, cf. Figure 3.1. Formally, $C_i$ and $C_j$ are separable iff*

$$\forall \mathbf{o},\mathbf{p} \in C_i \wedge \forall \mathbf{q} \in C_j : dist(\mathbf{o},\mathbf{p}) < dist(\mathbf{o},\mathbf{q})$$

Given such separable clusters (cf. Figure 3.1), we want to prove that local neighborhoods are *sufficient indicators*. This means that using ordered neighborhoods

we guarantee to find all hidden clusters. For example, in Figure 3.2 the clusters $C_i$ and $C_j$ are well separable, and this can be detected via local neighborhoods, as all objects in $C_i$ exist in each other's neighborhoods. Furthermore, objects in a well separated cluster recurrently co-occur in the neighborhoods. Formally, we can prove the following:

**Theorem 3.1** (Recurrency of Clusters)**.** *Objects in a well separated cluster recurrently co-occur in the neighborhoods of the other objects. Let $C_i$ be a well separated cluster with a size of $k$ and $\mathbf{o},\mathbf{p},\mathbf{r} \in \mathcal{DB}$,*

$$\mathbf{o},\mathbf{p} \in C_i \iff |\{\mathbf{r} \mid \mathbf{o},\mathbf{p} \in k\text{-}NN(\mathbf{r})\}| = k$$

In order to prove the theorem, we review the aforementioned concepts under the assumptions of the theorem and derive the necessary lemmas.

**Lemma 3.1.** *According to Definition 3.2, objects in the same cluster are more similar to each other than the objects from other clusters. Therefore, the objects $\mathbf{o},\mathbf{p} \in C_i$ exist in each other's neighborhood:*

$$\mathbf{o} \in k\text{-}NN(\mathbf{p})$$

**Lemma 3.2.** *Following Lemma 3.1, each object exists in the neighborhoods of the objects in the same cluster. Since the neighborhood size is equal to the cluster size, there cannot be any objects in the neighborhood that is not in the cluster. Therefore, the neighborhoods of the objects in a cluster are equal to the cluster itself, cf. Figure 3.2.*

$$k\text{-}NN(\mathbf{o}) = C_i$$

*Proof.* Since the objects in the same cluster only occur in the neighborhoods of each other,

$$\mathbf{o},\mathbf{p} \in k\text{-}NN(\mathbf{r}) \iff \mathbf{r} \in C_i$$

$\mathbf{o}$ and $\mathbf{p}$ occur in the neighborhoods of the objects in $C_i$, and hence, the number of neighborhoods that they occur is the size of the cluster. □

### 3.2.2 Ordered Neighborhoods

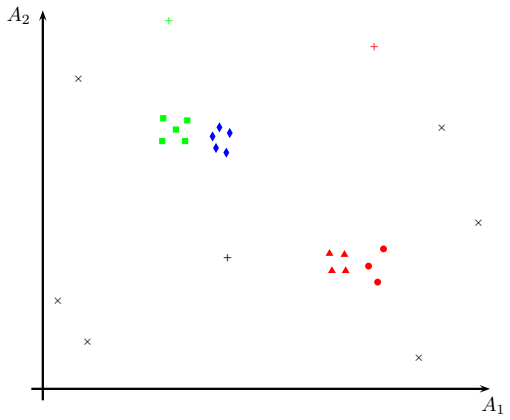
In the original space, computing the co-occurrence of object sets in neighborhoods of objects in $\mathcal{DB}$ requires expensive neighborhood computations. Instead, we compute the neighborhood of every object once and conduct co-occurrence search in these neighborhoods.

**Definition 3.3** (Ordered Neighborhood)**.** *The* ordered neighborhood *of $\mathbf{o} \in \mathcal{DB}^{A_i}$ is the ordered list of the objects in $k$-$NN(\mathbf{o})$, in which the order reflects the order of objects in $A_i$. For $\mathbf{p},\mathbf{q} \in \mathcal{DB}^{A_i}$,*

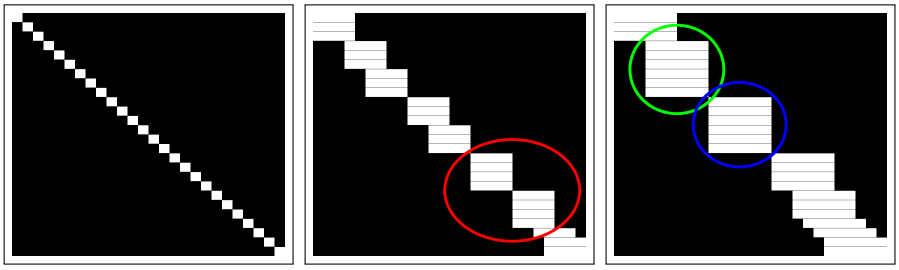
$$k\text{-}NN(\mathbf{o}) = (\cdots,\mathbf{p},\cdots,\mathbf{q},\cdots) \iff \mathbf{p} < \mathbf{q}$$

**Definition 3.4** (Ordered Neighborhood Database)**.** *An* ordered neighborhood database *$\mathcal{ND}$ is the ordered list of ordered neighborhoods. Order of the neighborhoods is the order of objects in $\mathcal{DB}^{A_i}$. For $\mathbf{o},\mathbf{p} \in \mathcal{DB}^{A_i}$, if $\mathbf{o} < \mathbf{p}$, then*

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




(a) An example dataset



(b) Neighborhoods of size 1 (c) Neighborhoods of size 4 (d) Neighborhoods of size 6

Figure 3.3: An example dataset and its neighborhood databases for different sizes

For the remaining of the paper, we use *neighborhood* and $k$-$NN(\mathbf{o})$ 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 $\mathscr{ND}$ instead.

Let us look at the example dataset in Figure 3.3a. Figures 3.3b, 3.3c and 3.3d show $\mathscr{ND}^{A_1}$ of the example dataset respectively with $k = 1$, $k = 4$ 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 3.3c each horizontal white line, which denotes 4-*NN*, consists of exactly 4 cells while there are white cells only on the diagonal in Figure 3.3b since 1-*NN* of an object is equal to the object itself.

With a formal analysis, following properties hold for the neighborhood databases:

**Property 3.1** (Consecutive Objects)**.** *If two objects are in a neighborhood, all the objects in between them are also in that neighborhood. Let* $\mathbf{o},\mathbf{p},\mathbf{q},\mathbf{r} \in \mathscr{DB}^{A_i}$ *and* $\mathbf{o} < \mathbf{p} < \mathbf{q}$. $\mathbf{o} \in k\text{-}NN(\mathbf{r}) \wedge \mathbf{q} \in k\text{-}NN(\mathbf{r}) \implies \mathbf{p} \in k\text{-}NN(\mathbf{r})$.

*Proof.* Our proof is in two complementary parts, $x \in \mathscr{DB}^{A_i}$:

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 $\mathbf{p} \in k\text{-}NN(\mathbf{r})$. □

We can see this property in Figure 3.3c: The objects in a neighborhood are always consecutive and they form a straight horizontal line.

**Property 3.2** (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* $\mathbf{o}, \mathbf{p}, \mathbf{q}, \mathbf{r} \in \mathscr{DB}^{A_i}$ *and* $\mathbf{o} < \mathbf{p} < \mathbf{q}$*. If* $\mathbf{r} \in k\text{-}NN(\mathbf{o}) \wedge \mathbf{r} \in k\text{-}NN(\mathbf{q}) \implies \mathbf{r} \in k\text{-}NN(\mathbf{p})$.

*Proof.* Our proof is in two complementary parts, $x \in \mathscr{DB}^{A_i}$:
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 $\mathbf{r} \in k\text{-}NN(\mathbf{p})$. □

Neighborhoods in Figure 3.3c are consecutive: An object only appears in consecutive neighborhoods and form a straight vertical line.

**Theorem 3.2** (Recurrent Neighborhoods)**.** *Clusters form square structures on the diagonal of the neighborhood databases.*

*Proof.* According to Properties 3.1 and 3.2, neighborhoods form continuous shapes. Theorem 3.1 and its lemmas state that a cluster is repeated as the neighborhoods of its objects, hence, the shape should be a square. And 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 3.3c, which are circled in red, are the neighborhoods of the objects in the clusters that are denoted by ▲ and • in Figure 3.3a. Similarly, squares on the upper left quarter of Figure 3.3d, which are circled in green and blue, are respectively from the clusters ■, ⧫.

**Property 3.3** (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 3.4.*

*Proof.* Let us represent the objects as $\{\mathbf{o}^0, \mathbf{o}^1, \cdots, \mathbf{o}^i, \cdots\} = \mathscr{DB}$. 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 $m < i < |\mathscr{DB}| - m$, $\delta(\mathbf{o}^i, \mathbf{o}^{i-m}) = \delta(\mathbf{o}^i, \mathbf{o}^{i+m})$, 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. □ □

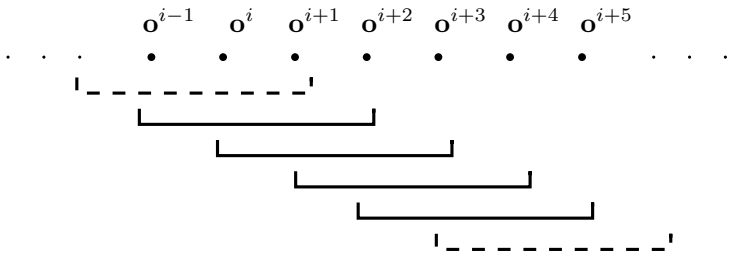



Figure 3.4: Neighborhoods of a uniformly distributed data

For example, in a uniform segment, each individual object appears in exactly $k$ neighborhoods, and since each neighborhood is unique, a set of consequent $k$ objects appear in only one neighborhood while a set of $k/2$ consecutive objects appear exactly in $k/2$ neighborhoods. Baseline property helps us to discard the segments without a cluster structure end to eliminate the artifacts that are caused by the transformation.

**Property 3.4** (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* $\mathbf{o}, \mathbf{p}, \mathbf{q} \in \mathscr{DB}^{A_i}$ *and* $\mathbf{o} < \mathbf{p} < \mathbf{q}$*,* $\mathbf{o} \notin k\text{-}NN(\mathbf{p}) \implies \mathbf{o} \notin k\text{-}NN(\mathbf{q})$.

*Proof.* 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 3.2 ($\mathbf{o} \leq \mathbf{p} \leq \mathbf{q}$). Thus, $\mathbf{o}$ cannot be a neighbor of $\mathbf{q}$ if it is not a neighbor of $\mathbf{p}$. □ □

In Figure 3.3c, 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 3.3.

### 3.2.3 Generalization and Discussion

#### Non-separable Clusters

While we discuss the properties of separable clusters, we generally deal with clusters that are not separable. Fortunately, all of the properties still apply to such clusters, while we have to relax the assumption of Theorem 3.1 and its lemmas.

In Figure 3.5, $C$ is not a separable cluster, that is, some of the points in $C$ are more similar to the data points that are not in the cluster, $\delta(\mathbf{o}, \mathbf{q}) < \delta(\mathbf{o}, \mathbf{p})$. If $k$ is selected as the size of cluster $C$, then the neighborhoods of $\mathbf{o}$ and $\mathbf{p}$ include the points that are not in the cluster, resulting in a decrease in the recurrency of the objects in neighborhoods. However, the points that are closer to the center of the cluster still produce recurrent neighborhoods, most of which still include $\mathbf{p}$ and $\mathbf{o}$. Here, we can see the advantage of looking at the co-occurrences in the neighborhoods instead of directly comparing the neighborhoods: Although the neighborhoods of $\mathbf{o}$ and $\mathbf{p}$ are different, they still co-occur in the neighborhoods of the objects in the cluster.

Figure 3.6a shows a 1-D projection for an example dataset, the x-dimension of the figure represents the values for the projection and y-dimension is used to

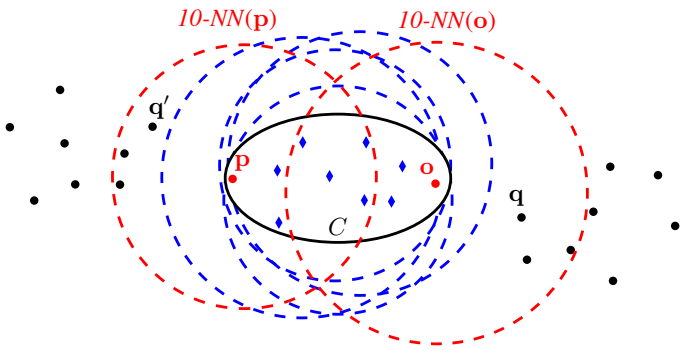
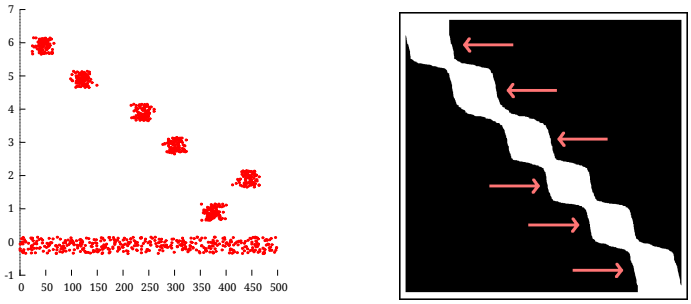



Figure 3.5: $C$ is a not a separable cluster




(a) 6 clusters

(b) Neighborhoods

Figure 3.6: 6 clusters in a projection and the neighborhood plot

separate 6 clusters (1–6) and the noise (0). Note that the y values are jittered to create a clearer view. Figure 3.6b shows the neighborhood DB for this projection, in which there are 6 square-like formations. Each of these formations corresponds to a cluster in the original data. The cluster structures are visible and computationally detectable although the clusters are not separable.

To sum up, neighborhoods with a sufficient size include the whole cluster even though the cluster is not separable. Therefore, not separable clusters can be found with the correct parameter setting, which we discuss in Section 3.2.3.

#### Cluster Detection in Data Projections

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 the irrelevant dimensions, our method first finds the clusters in each projection (dimension) separately. Clusters in one dimensional projections are the supersets of the clusters in higher dimensional spaces [6]. Therefore, we use lower dimensional clusters to

discover higher dimensional clusters.

We refine each one dimensional cluster by checking whether any of its subsets are recurrent in other dimensions. If there are any, we continue the search of higher dimensional clusters by traversing all the possible dimensions until none of the subsets satisfy the conditions.

#### Parameter Selection

Our method 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, cluster size is easy to decide and apply.

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 3.3a the points marked with + become noise for the projection and to find the cluster of size 5, neighborhood should be selected as 6, cf. Figure 3.3d.

In favor of the robustness of our method, the neighborhood size does not have to be exact. Equation in the proof of Theorem 3.1 holds for the neighborhoods of the cluster size. If the neighborhood size is greater than the cluster size, then the neighborhoods include objects from other clusters *in addition to* the objects in the cluster. Formally, for $\exists \mathbf{r} \in C_i$,

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

If the noise or the other clusters are uniformly distributed around $C_i$, 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. 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. $k < |C_i|$, then a subset of the cluster becomes recurrent in the neighborhoods. 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 interactively shows the neighborhood DB, so that the neighborhood size that produce the most recurrency can be selected. We discuss this tool in Chapter 5.

## 3.3 Efficient Mining

In this section we introduce our proposed algorithm CLON, which efficiently finds the subspace clusters by using the ordered neighborhoods. CLON takes two parameters, (1) Minimum size of a cluster *MinSize* and (2) Neighborhood size *k*, and outputs all the subspace clusters in the database.

### 3.3.1 Transformation

In its first phase, CLON converts the relational database into neighborhood databases. Neighborhood databases are created for each of the projections (dimensions). Values in each dimension has to be sorted only once, then the neighborhoods can be calculated very fast by using a sliding window. A neighborhood DB is created for each of the projections or each of the (dis)similarity measures. Algorithm 3 shows the process of creating neighborhood DBs.

**Algorithm 3** Create neighborhood databases

**Input:** $\mathscr{DB}$: Real valued high dimensional database, $k$: Neighborhood size
**Output:** $\mathscr{T}$: Neighborhood databases

```
1: T ← []
2: for each DB^{A_i} ∈ DB do
3:     sort DB^{A_i} in ascending order
4:     ND^{A_i} ← []
5:     for each o ∈ DB^{A_i} do
6:         append k-NN(o_i) to ND^{A_i}
7:     T_i ← ND^{A_i}
8: return
```

CLON already benefits from the ordered neighborhoods for the neighborhood DB creation. It exploits the consecutiveness properties to store the neighborhood information so that, instead of keeping the whole neighborhoods, only the start and end of the neighborhoods are kept. This provides the algorithm a dramatic saving on the memory.

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 in. Since both the objects and the neighborhoods are continuous, according to the Properties 3.1 and 3.2, we only require the starting neighborhood, that is the id of the neighborhood in which the object first appears, and the end neighborhood, that is the id of the neighborhood in which 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*.

### 3.3.2 Finding the Neighborhoods in Projections

Second phase is about finding the clusters in individual projections. As Theorem 3.2 states, clusters are the recurrent object sets in the neighborhood DB. CLON looks for object sets that are large and recurrent enough to satisfy *MinSize*. These object sets can be found in linear (with the number of objects) time by exploiting the properties of the ordered neighborhoods. In this phase, CLON identifies all of the recurrent objects sets in individual projections as one dimensional clusters.

Algorithm 4 shows the MaxiBand algorithm, our efficient maximal recurrent object set miner that we use to find one dimensional clusters. Since each subset of a

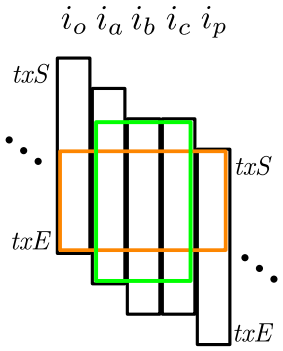



Figure 3.7: Heights of the rectangles are the supports of the respective itemsets

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 $\lambda$ as input. $\lambda$ 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 $\sigma$, which returns the number of neighborhoods a set of objects occur. Note that $\sigma$ exploits the consecutiveness properties of the neighborhood database (line 4). This support computation is visualized in Figure 3.7, which shows 5 consecutive items with their transaction lists. Start transaction and end transaction of the items $i_o$ and $i_p$ are marked with *txS* and *txE*. Support of the itemset $\{i_o, i_a, i_b, i_c, i_p\}$ is the height of the orange rectangle, which is $|i_o.txE - i_p.txS|$. Similarly, height of the green rectangle is the support of the itemset $\{i_a, i_b, i_c\}$.

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 $\lambda$ 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).

### 3.3.3 Finding Subspace Clusters

In the third and the last phase, one dimensional clusters are used as a base to find higher dimensional clusters. CLON conducts a deep-first search on dimensions to find out whether the one dimensional clusters are supersets of higher dimensional clusters. We exploit properties of ordered neighborhoods for this phase to speed-up the computation. Lastly, the recurrent object sets are output as clusters along with the dimensions that they are recurrent in.

Algorithm 5 shows the CLON algorithm that finds all the subspace clusters. It takes a numeric high dimensional database $\mathscr{DB}$, a cluster size $\lambda$, and a neighbor-

**Algorithm 4** 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: $e \leftarrow s+\lambda$

hood size $k$ as inputs and outputs all the subspace clusters in $\mathscr{DB}$. 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 $\mathscr{DB}$ 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 $\lambda$ as an argument to MaxiBand, we pass the maximum of $\lambda$ and $k \times 0.6$. Because the baseline property, Property 3.3, 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 $\lambda$ 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 3.2.3.

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.

Second part 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).

**Algorithm 5** CLON Algorithm

**Input:** $\mathscr{DB}$: Numeric high dimensional database, $\lambda$: Cluster size, $k$: Neighborhood size
**Output:** $\mathscr{C}$: Subspace clusters

```
1: C ← ∅
2: initialize empty stack S
3: A = {1, 2, ··· , |𝒜|}
4: T ← Convert DB to neighborhood DBs
5: for each ND^{A_i} ∈ T do
6:    F ← MaxiBand (I ∈ ND^{A_i}, max(λ, k × 0.6))
7:    for each f ∈ F do
8:       C ← C ∪ (f, {i})
9:       push (f, {i}, A − {i}) to S
10: repeat
11:    (f, D, P) ← pop S
12:    for each p ∈ P do
13:       I' ← get ordered items of f from ND^{A_p}
14:       F ← MaxiBand (I', λ)
15:       for each f' ∈ F do
16:          C ← C ∪ (f, D ∪ {p})
17:          push (f, D ∪ {p}, P − {p}) to S
18: until S is empty
```

### 3.3.4 Complexity Analysis

For a numeric $\mathscr{DB}$ that had $n$ objects and $d$ attributes, computational complexity of transforming it to the neighborhood DB is $O(d \times n \times \log(n) \times k)$. And, thanks to the consecutiveness properties we do not have to store the whole data, only the start and end of the neighborhoods are enough. Therefore, space need for the neighborhood DBs is $O(n \times d)$ which is equivalent to the storage need of the original $\mathscr{DB}$.

Exploiting the properties of the neighborhood DB allows us to find one dimensional clusters extremely efficiently, in $O(n)$. Therefore, complexity of finding all the clusters in all the projected databases is $O(d \times n)$.

Execution time for CLON to find all of the subspace clusters is highly dependent on the characteristics of the data. Section 3.5 shows the experiments that we conduct on various kinds of data to empirically evaluate the effectiveness of our method.

## 3.4 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 [61, 62, 97, 84] and comparative analysis articles [78] 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 [10].

PROCLUS [5] 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 [78].

FIRES [60] 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 $\epsilon$-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 [78] requires 9 different parameters, none of which is trivial to set.

CartiClus [10] uses a neighborhood transformation that is similar to that we use and employs frequent itemset mining (FIM) 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 [56] 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 [75] 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 3.5.

## 3.5 Empirical Evaluation

### 3.5.1 Experimental Setup

We evaluate our method on heterogeneous datasets, i.e., datasets with different scales, in terms of both 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 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 [10], FIRES [60], PROCLUS [5], STATPC [75], and SUBCLU [56], cf. Section 3.4.

Object cluster finding capabilities of the methods are evaluated by the supervised F1 Measure. The F1 score is the harmonic mean of *precision* and *recall*. *precision* between a found cluster $C_i$ and a true cluster $\mathbf{C}_j$ is $\frac{|C_i \cap \mathbf{C}_j|}{|C_i|}$, and the *recall* is $\frac{|C_i \cap \mathbf{C}_j|}{|\mathbf{C}_j|}$. Following the practice in literature [75, 78], we match found clusters to true clusters based on their intersection. A detailed explanation of the F1 Score is given in

| | # of Objects | # of Dimensions | Cluster Size | Dims per Cluster | Max/Min Dimension Range | Noise Ratio |
|---|---|---|---|---|---|---|
| DS-1 | 840 | 12 | 100 | 4–5 | 1 | 5% |
| DS-2 | 840 | 12 | 100 | 4–5 | 2 | 5% |
| DS-4 | 840 | 12 | 100 | 4–5 | 4 | 5% |
| DS-8 | 840 | 12 | 100 | 4–5 | 8 | 5% |
| DS-16 | 840 | 12 | 100 | 4–5 | 16 | 5% |
| DS-32 | 840 | 12 | 100 | 4–5 | 32 | 5% |
| DS-100 | 840 | 12 | 100 | 4–5 | 100 | 5% |
| DS-200 | 840 | 12 | 100 | 4–5 | 200 | 5% |

Table 3.1: Properties of the datasets with different scaling per dimension. For example, ranges of all of the dimensions of DS-1 are [0,500] while for DS-8, some of the ranges are [0,500] and some of them are [0,4000].

Section 2.5. We report both the *F1-Precision* (2.2) and *F1-Recall* (2.3)for better comparison and fairness. Where appropriate, we evaluate the subspace discovery capabilities by using the established E4SC score [78], cf. Section 2.5.

Runtime results are given for the performance versus scale experiments. For CartiClus, we use the implementation from the authors [10] and for the other methods we use the OpenSubspace package [78]. Our method is implemented in Java. All of the experiments are run on a computer with Intel i7 CPU and 8 GB of memory running with a GNU/Linux operating system. For the reproducibility of the experiments, a cross-platform implementation of the method and the information about the datasets are available on our web page.[2]

### 3.5.2 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 generated two sets of 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. Properties of these datasets are shown in Table 3.1. (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. Properties of these datasets are shown in Table 3.2.

Figures 3.8 and 3.9 respectively show the *F1-Recall* and *F1-Precision* scores of the methods on dimension scaling datasets. And Figures 3.12 and 3.12 respectively show *F1-Recall* and *F1-Precision* scores of the methods on cluster scaling datasets. 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-

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

| | # of Objects | # of Dimensions | Cluster Size | Dims per Cluster | Max/Min Cluster Radius | Noise Ratio |
|---|---|---|---|---|---|---|
| CS-1 | 840 | 9 | 100 | 2–3 | 1 | 5% |
| CS-2 | 840 | 9 | 100 | 2–3 | 2 | 5% |
| CS-4 | 840 | 9 | 100 | 2–3 | 4 | 5% |
| CS-8 | 840 | 9 | 100 | 2–3 | 8 | 5% |
| CS-16 | 840 | 9 | 100 | 2–3 | 16 | 5% |
| CS-32 | 840 | 9 | 100 | 2–3 | 32 | 5% |
| CS-100 | 840 | 9 | 100 | 2–3 | 100 | 5% |
| CS-200 | 840 | 9 | 100 | 2–3 | 200 | 5% |

Table 3.2: Properties of the datasets with different cluster spreads. For example, radiuses of the clusters in CS-1 are all approximately 10 units while in CS-8, radiuses of some of the clusters are 10 and some of them are 80 units.

based clustering. Radius-based neighborhood evaluation of FIRES and distance sensitive projections of PROCLUS cannot cope with scaling. STATPC produces high quality, although unstable, results because of its thorough search, which becomes a burden in terms of execution time, cf. Section 3.5.3. The results of SUBCLU are interesting: while *F1-Recall* results are very high, *F1-Precision* results are very low. This is because SUBCLU detects many clusters and although the known clusters are among them the majority of them are not in accordance with the known clusters.

Figure 3.10 and Figure 3.11 show the E4SC scores for the same sets of datasets. While some of the methods perform poorly on dimension detection, for most of the methods, dimension finding capabilities of the methods are in parallel with the object cluster finding capabilities.

### 3.5.3 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 [78, 10]. Properties of the datasets are given in Table 3.3.

Figure 3.16 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 3.17 shows the run times for different methods on datasets with an increasing number of dimensions. We conduct a test on 6 different datasets that have approximately 1500 objects in 5, 10, 15, 25, 50, and 75 dimensions. 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.

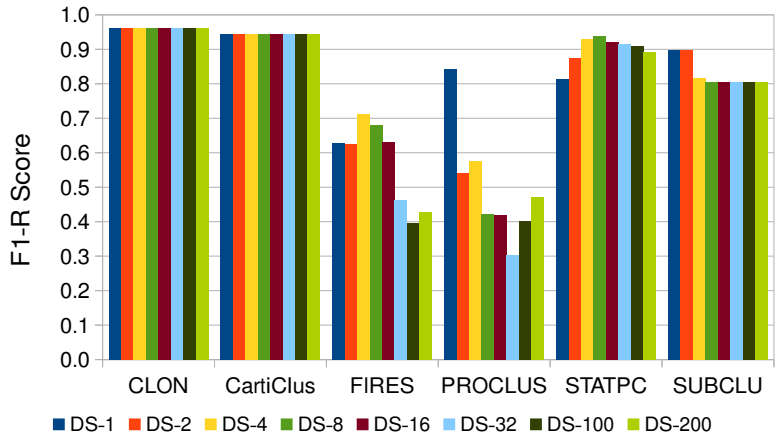



Figure 3.8: F1-Recall scores of the methods on datasets with dimension of various scales

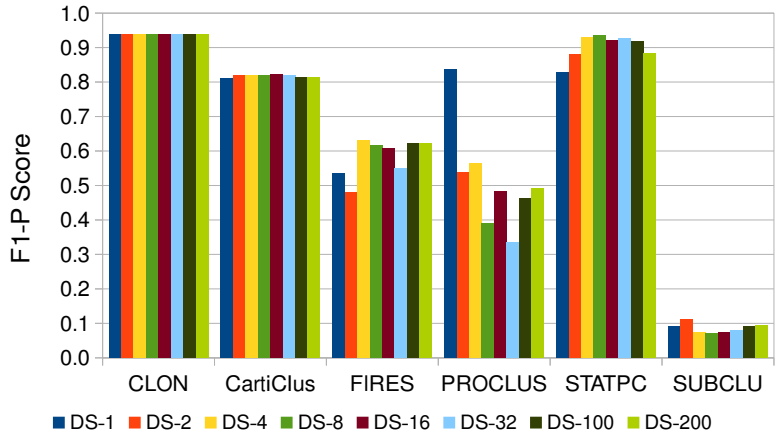



Figure 3.9: F1-Precision scores of the methods on datasets with dimension of various scales

F1 scores (2.4) for the experiments on scalability w.r.t. size and dimension are shown in Figures 3.14 and 3.15.

Figures 3.18 and 3.19 show the quality results for the datasets that contain 10%, 30%, 50%, and 70% noise. Our method and CartiClus can effectively discard noise and find the known clusters in noisy databases thanks to their recurrency and neighborhood-based foundations. In Figure 3.19, we see that CLON detects some cluster that are not in the known set of clusters. This is because CLON finds all of the clusters in all of the dimensions and it is possible that some of these noise objects

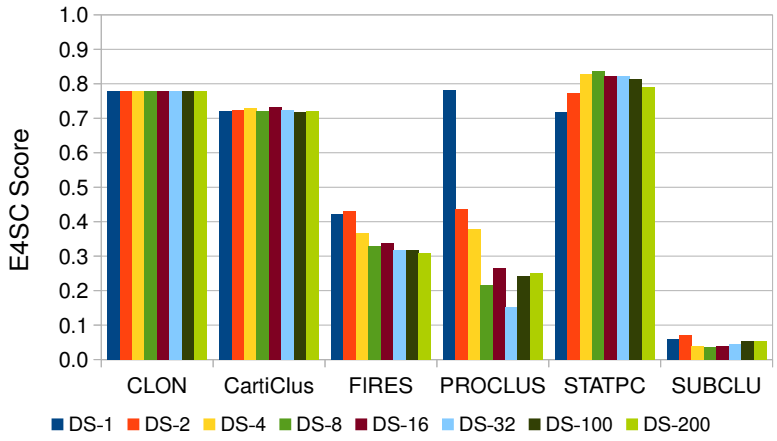



Figure 3.10: E4SC scores of the methods on datasets with dimension of various scales

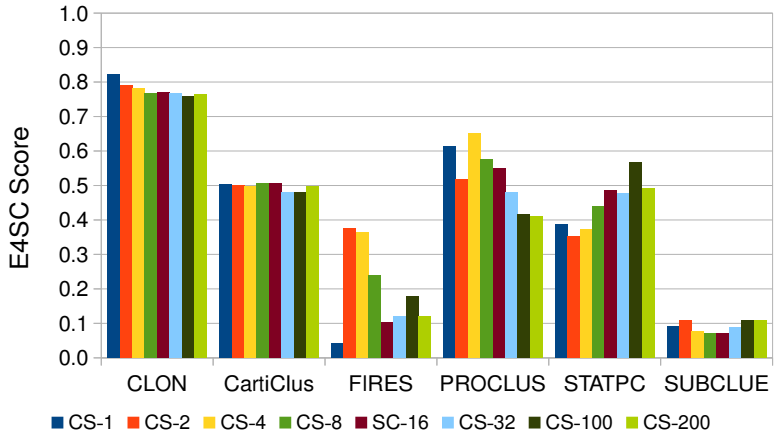



Figure 3.11: E4SC scores of methods on datasets that have clusters of various scales

form clusters. Overall, CLON is one of the best performers in noisy data along with CARTICLUS.

Figures 3.20 and 3.20 show the capability of irrelevant dimension detection. A database which has 10 clusters hidden in subsets of 10 dimension is created. Then, 10, 50, 100 and 200 uniformly randomly generated dimensions are added to the dataset. Here again, we see that CLON, CARTICLUS and SUBCLU can effectively find the known clusters. However, only CLON, CARTICLUS, and STATPC can discard irrelevant dimensions and report *only* the known clusters. STATPC and SUBCLU did not complete in meaningful time for some of the datasets, hence the missing values.

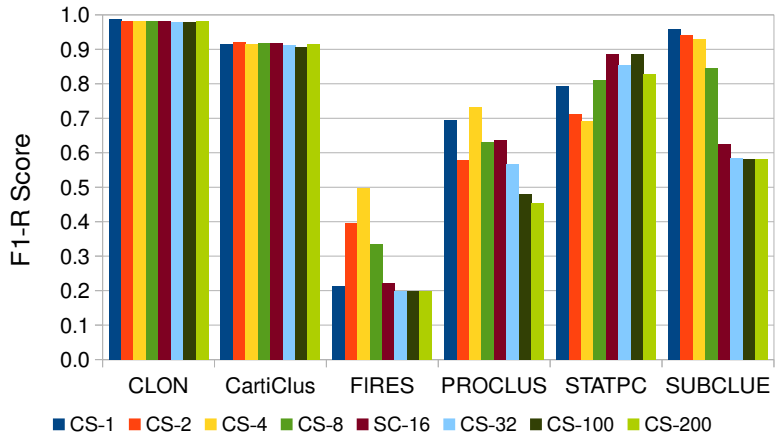



Figure 3.12: F1-Recall scores of methods on datasets that have clusters of various scales

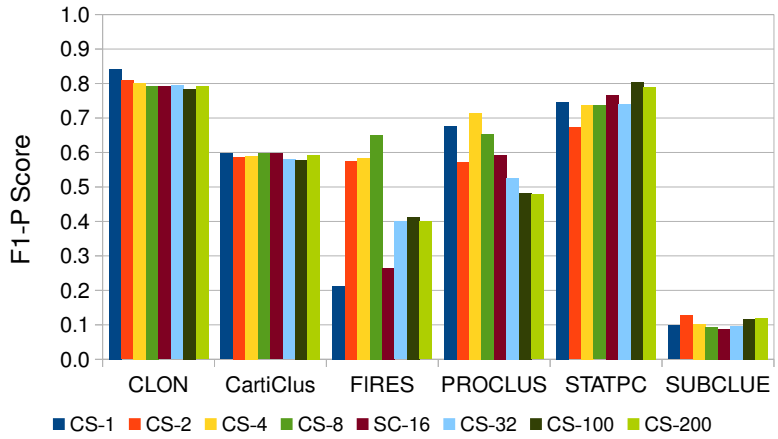



Figure 3.13: F1-Precision scores of methods on datasets that have clusters of various scales

### 3.5.4 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 [13]. *Nutt* dataset contains expressions of 1377 genes on 50 glioma tissue samples that are grouped into 4 different pathology categories [80]. High dimensional nature of both datasets make the clusteringchallenging even for subspace clustering methods. Table 3.4 shows the

| Name | # of Objects | # of Dimensions | Cluster Size | Dims per Cluster | Noise Ratio |
|---|---|---|---|---|---|
| S1500 | **1595** | 20 | **150** | 12–16 | 8.5% |
| S2500 | **2658** | 20 | **250** | 12–16 | 8.5% |
| S3500 | **3722** | 20 | **350** | 12–16 | 8.5% |
| S4500 | **4785** | 20 | **450** | 12–16 | 8.5% |
| S5500 | **5848** | 20 | **550** | 12–16 | 8.5% |
| D05 | 1595 | **5** | 150 | 3–4 | 9% |
| D10 | 1595 | **10** | 150 | 5–8 | 9% |
| D15 | 1595 | **15** | 150 | 9–12 | 9% |
| D25 | 1595 | **25** | 150 | 15–20 | 9% |
| D50 | 1595 | **50** | 150 | 30–40 | 9% |
| D75 | 1596 | **75** | 150 | 45–61 | 9% |
| N10 | **1611** | 20 | 150 | 12–16 | **10%** |
| N30 | **2071** | 20 | 150 | 12–16 | **30%** |
| N50 | **2900** | 20 | 150 | 12–16 | **50%** |
| N70 | **4833** | 20 | 150 | 12–16 | **70%** |
| ND10 | 1050 | **20** | 100 | 3–6 | 5% |
| ND50 | 1050 | **60** | 100 | 4–7 | 5% |
| ND100 | 1050 | **110** | 100 | 4–6 | 5% |
| ND200 | 1050 | **210** | 100 | 4–6 | 5% |

Table 3.3: Properties of the datasets for scalability experiments

F1 scores of the methods on these data sets. "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.[3] We use 10M movielens dataset which includes 10000054 ratings applied to 10681 movies by 71567 users from the website `http://movielens.org`. We use 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 3.22a.

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

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

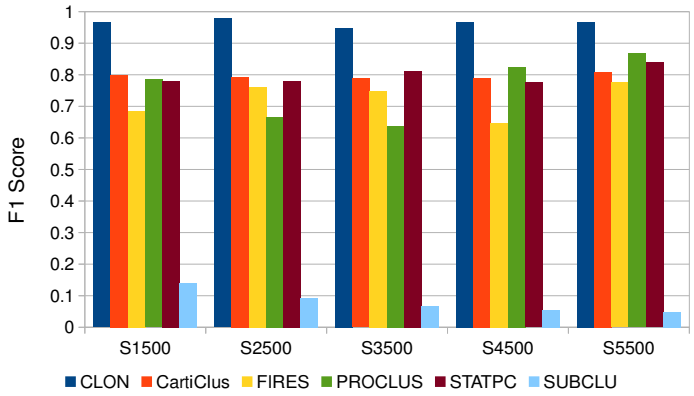



Figure 3.14: F1 Scores for the datasets with increasing number of objects

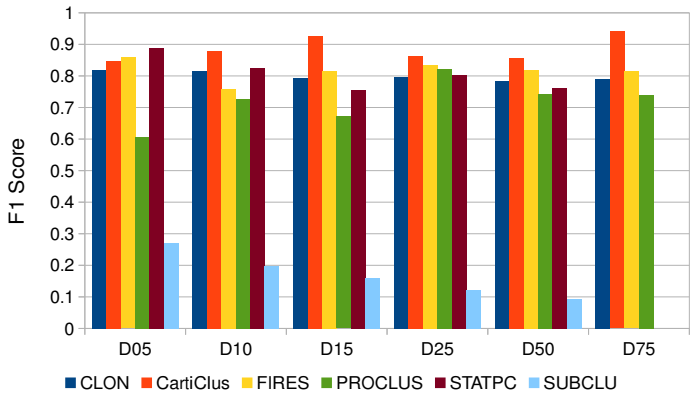



Figure 3.15: F1 Scores for the datasets with increasing number of dimensions

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 3.22b 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.

## 3.6 Conclusion

In this chapter, 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 recur-

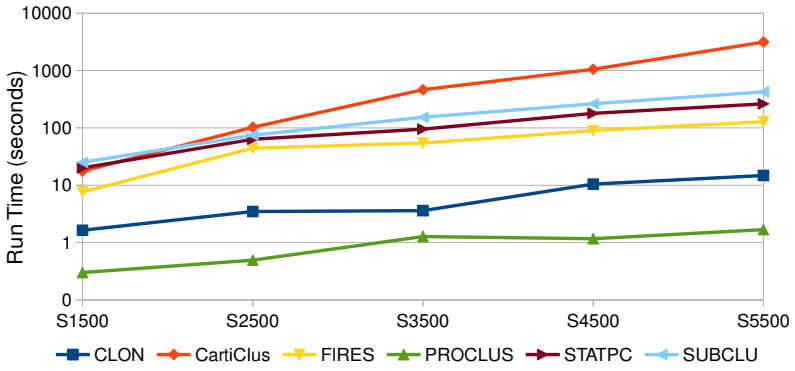



Figure 3.16: Execution times for datasets with increasing number of objects

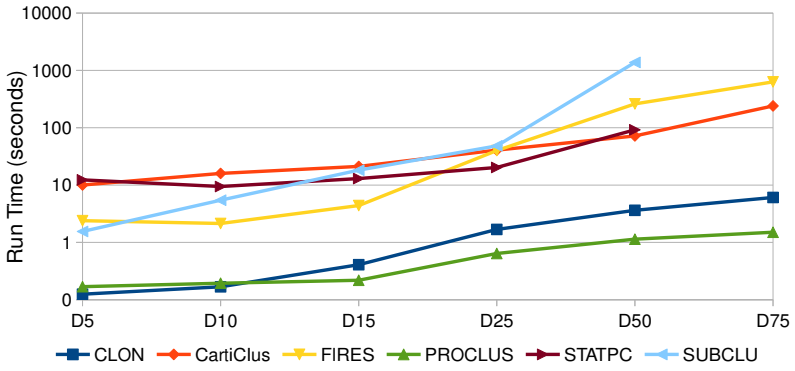



Figure 3.17: Execution times for datasets with increasing number or dimensions

rent 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.

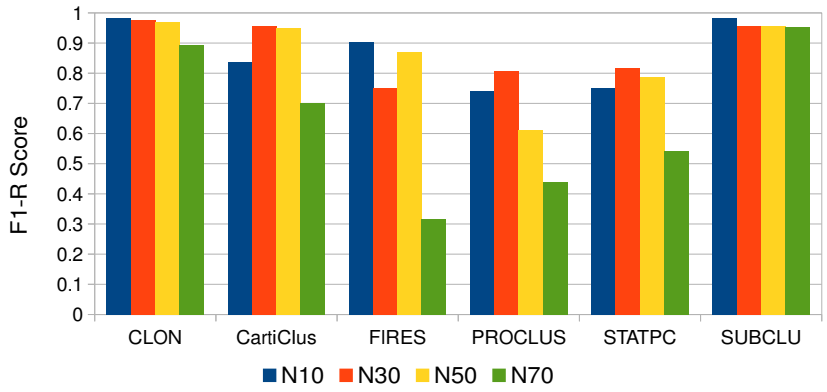



Figure 3.18: F1-Recall scores for datasets with increasing number of noise objects

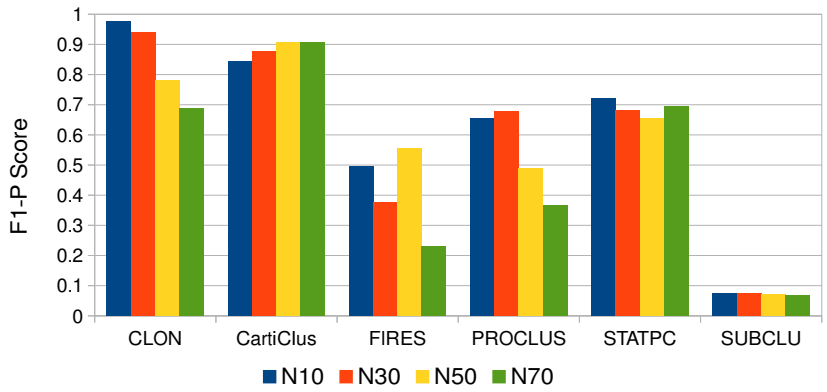



Figure 3.19: F1-Precision scores for datasets with increasing the number of noise objects

| | *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 3.4: F1 scores for gene expression datasets

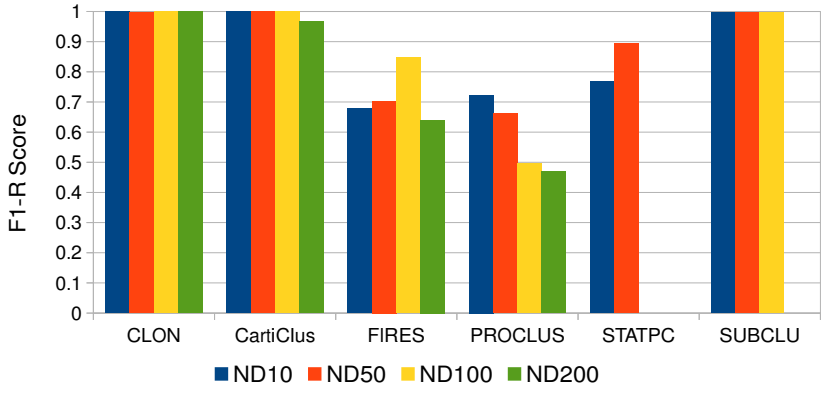



Figure 3.20: F1-Recall scores for datasets with increasing number of irrelevant dimensions

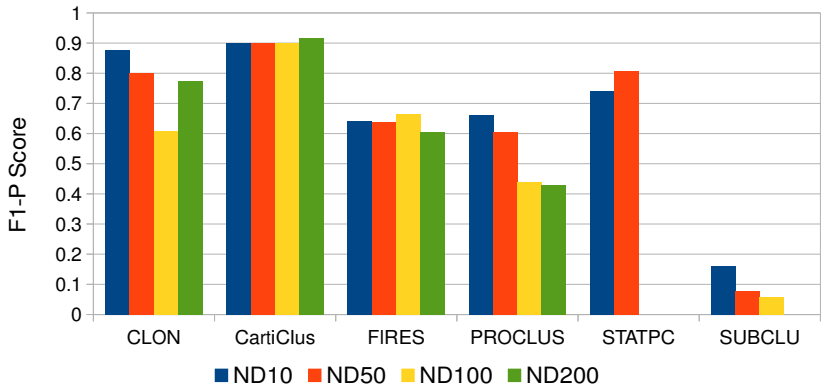



Figure 3.21: F1-Precision scores for datasets with increasing number of irrelevant dimensions

| |
|---|
| 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

Figure 3.22: Clusters of Movies

CHAPTER 4

# Exploiting the Order in Neighborhoods for Cluster Detection

*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 also was found to depend heavily on the setting of parameters. In other words, it is not robust to input parameters.*

*In this chapter,*[1] *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 investigate the properties of these ranked matrices and how cluster structures are represented in them. We propose an algorithm that exploits the properties of the ranked matrices to efficiently find cluster structures in combinations of dimensions. We demonstrate that this method is more robust than cartification in terms of cluster detection.*

[1] This chapter is based on work that will appear in ICDMW HDM'15 "Finding Subspace Clusters using Ranked Neighborhoods" by Emin Aksehirli, Siegfried Nijssen, Matthijs van Leeuwen, and Bart Goethals [12].

## 4.1 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, most points in the data become equidistant to each other, which makes it impossible to apply standard clustering techniques based on such distances.

One solution to this challenge is to perform *subspace clustering*. In subspace clustering, we do not cluster the data points based on all attributes, but only cluster the points 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 [10, 56, 5, 75]. 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 [78]. This makes it harder to get reliable results using these clustering techniques.

To address this challenge, the *cartification* approach to subspace clustering was developed. For one attribute, the cartification approach studied in Chapter 3 [11] proceeds as follows:

1. It calculates pairwise distances between all data points based on this attribute.

2. For each data point, it calculates 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 are set to the value 1.

3. It searches 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 is 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 data points within each of these clusters, cartification finds subspaces consisting of sets of attributes and sets of data points.

An advantage of the cartification approach is that in step 2, 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. Furthermore, infeasible computation of neighborhoods for each set of objects becomes unnecessary by computing the neighborhoods once and creating a neighborhood database.

Another advantage of cartification is its easy to determine 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 finding out the parameters such as density or distribution. Our earlier work showed that after parameter tuning of all methods, cartification performed better than other subspace clustering approaches, in the sense that the clusters it found are better in terms of an F1 score [10, 11].

On the other hand, the earlier work [11] also showed that the quality of the results of cartification is highly dependent on the right choice for the parameter $k$. Moreover, in some cases there is no right choice for $k$ because the dataset contains clusters of different sizes.

In this chapter, we address this challenge of setting good robust parameters. 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, this is the idea:

- 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 this 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 one other parameter $\theta$ 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 threshold all the ranks at $k$, we could easily create the binary matrix that was used in the original cartification approach.

Furthermore, experimentally we will show that the proposed method is less sensitive to the choice of the parameter $\theta$ and obtains results that are equally well as the original cartification method, or better.

The outline of this chapter is as follows: In Section 4.2, we give an overview of cartification in our terms; in Section III we introduce our proposed modification. We compare the binary cartification method and our algorithm in Section 4.4, and we conclude in Section 4.5.

## 4.2 Subspace Clustering using Cartification

This chapter improves the cartification approach for subspace clustering. We start this section by summarizing the most recent cartification algorithm CLON as introduced in Chapter 3 [11].

Cartification operates on relational databases. A *relational database* $\mathscr{DB}$ 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 attributes $\mathscr{A} = \{A_1, A_2, \cdots, A_m\}$. We are interested in finding subsets of attributes $S \subseteq \mathscr{A}$ and subsets of object identifiers $O \subseteq \{1, \ldots, n\}$ such that for each attribute $A \in S$, the values in the selected set of objects $O$ are similar.

We use $\delta(\mathbf{o}_i, \mathbf{o}_j)$ as the dissimilarity between objects $\mathbf{o}_i$ and $\mathbf{o}_j$. $|O|$ is the cardinality of set $O$.

The Cartification algorithm is an instance of Algorithm 6, where the algorithm is initialized with $S = \emptyset$ and $O = \{1, \ldots, n\}$. The algorithm iteratively considers all attributes and identifies clusters in the attributes by using a 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 recurses 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 $\mu$ and the exact implementation of function FINDCLUSTERS.

**Algorithm 6** Subspace Search ($S$, $O$)

```
1: for each d ∈ {1,...,m} do
2:   if A_d ∉ S then
3:     C := FINDCLUSTERS (O, a_d)
4:     for each O' ∈ C do
5:       if |O'| ≥ μ then
6:         Subspace Search (S ∪ {A_d}, O')
```

In the version of Cartification proposed by Aksehirli et al. [11], 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 dataset.

More formally, we can define the $k$-nearest neighborhood for one object as follows.

**Definition 4.1** ($k$-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 Table 4.1a is $6\text{-}NN(\mathbf{o}_1) = \{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_7, \mathbf{o}_8\}$.

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

**Definition 4.2** (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 dataset in Table 4.1a is shown in Table 4.1b.

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, 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, cf. Table 4.1b.

Consequently, in cartification an *itemset mining*-like approach is used to identify sets of frequently co-occurring objects. In [11], this is done by looking for frequent sets of objects in the neighborhood database. Each such frequent set identifies a cluster, and is returned by the FINDCLUSTERS function used in Algorithm 1.

One advantage of cartification is which clusters to be found are determined by the relative order of the objects; the original distances in the database 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

| | $A_1$ |
|---|---|
| $\mathbf{o}_1$ | 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

Table 4.1: An example database of 12 data objects with one attribute and its neighborhood database

other hand, we also limit the maximum size of clusters that can be found. Finding a value of $k$ that is neither too low nor too high is hence not easy. To address this problem, we propose a new approach in this chapter that operates on the ordered neighborhoods.

## 4.3 Ranked Cartification

In the approach studied in this chapter, we will exploit the relative similarity of objects while keeping the advantages of neighborhood approach. We achieve this by defining neighborhoods as ranked vectors instead of sets and hence keep the orders in the neighborhoods. We claim that clusters can be identified more robustly in these ranked neighborhoods than the binary ones.

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

### 4.3.1 Problem Definition

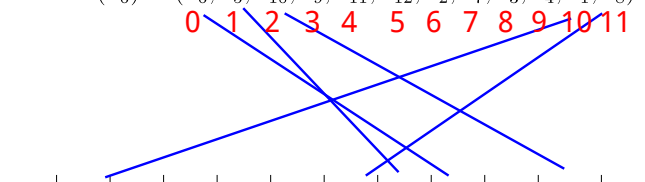
Following the transformation idea of cartification, we transform a relational database into a ranked neighborhood matrix, and then we use this matrix to detect subspace clusters.

First we extend the definition of neighborhood and take the order in the neighborhood into account.

**Definition 4.3** (Ordered Neighborhood)**.** Ordered neighborhood *of an object* $\mathbf{o}_i \in \mathcal{DB}$*,* $N(\mathbf{o}_i)$*, is a vector of objects that are ordered by their similarity to* $\mathbf{o}_i$*. For*

(a) Ordered neighborhoods of objects 1, 2, and 6.

$$N(\mathbf{o}_1) = (\mathbf{o}_1, \mathbf{o}_4, \mathbf{o}_8, \mathbf{o}_3, \mathbf{o}_7, \mathbf{o}_2, \mathbf{o}_{10}, \mathbf{o}_6, \mathbf{o}_5, \mathbf{o}_9, \mathbf{o}_{11}, \mathbf{o}_{12})$$
$$N(\mathbf{o}_2) = (\mathbf{o}_2, \mathbf{o}_7, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_1, \mathbf{o}_8, \mathbf{o}_{10}, \mathbf{o}_6, \mathbf{o}_5, \mathbf{o}_9, \mathbf{o}_{11}, \mathbf{o}_{12})$$
$$N(\mathbf{o}_6) = (\mathbf{o}_6, \mathbf{o}_5, \mathbf{o}_{10}, \mathbf{o}_9, \mathbf{o}_{11}, \mathbf{o}_{12}, \mathbf{o}_2, \mathbf{o}_7, \mathbf{o}_3, \mathbf{o}_4, \mathbf{o}_1, \mathbf{o}_8)$$




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

(b) Ranked neighborhood of $\mathbf{o}_6$, $\mathbf{N}(\mathbf{o}_i)$.

Figure 4.1: Example Neighborhoods

$\mathbf{o}_i, \mathbf{o}_j, \mathbf{o}_k \in \mathscr{DB}$ *and a dissimilarity* $\delta$,

$$N(\mathbf{o}_i) = (\cdots, \mathbf{o}_j, \cdots, \mathbf{o}_k, \cdots) \iff \delta(\mathbf{o}_i, \mathbf{o}_j) < \delta(\mathbf{o}_i, \mathbf{o}_k)$$

Ordered neighborhoods represent the relative similarities of pairs of objects in the dataset; while they are still unaffected by the scale. Figure 4.1a shows the *ordered neighborhoods* of $\mathbf{o}_1$, $\mathbf{o}_2$, and $\mathbf{o}_6$ in Table 4.1a. The closest data object to $\mathbf{o}_1$ is the $\mathbf{o}_1$ itself, therefore it is the first object in $N(\mathbf{o}_1)$, followed by $\mathbf{o}_4$ which is the second closest one. Note that the $k$-nearest neighborhood of $\mathbf{o}_i$ is the first $k$ elements of $N(\mathbf{o}_i)$.

**Definition 4.4** (Ranked Neighborhood)**.** *The* Ranked Neighborhood *of an object* $\mathbf{o}_i$ *is an* $n$*-dimensional vector, where* $n = |\mathscr{DB}|$*. The* $j^{th}$ *value of the vector is defined as*

$$\mathbf{N}(\mathbf{o}_i)_j = |\{\mathbf{o}_k \in \mathscr{DB} | \delta(\mathbf{o}_i, \mathbf{o}_k) < \delta(\mathbf{o}_i, \mathbf{o}_j)\}|,$$

*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.*

Figure 4.1b shows the *ranked neighborhood* of $\mathbf{o}_6$. The columns of the vector represent the objects 1 through 12. For example, since $\mathbf{o}_1$ is in the 10th position in $N(\mathbf{o}_6)$, the first value in the vector is 10.

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

**Definition 4.5** (Ranked Neighborhood Matrix)**.** *The* Ranked Neighborhood Matrix *is the matrix of ranked neighborhoods, i.e., it is the matrix*

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

| | $\mathbf{o}_1$ | $\mathbf{o}_2$ | $\mathbf{o}_3$ | $\mathbf{o}_4$ | $\mathbf{o}_5$ | $\mathbf{o}_6$ | $\mathbf{o}_7$ | $\mathbf{o}_8$ | $\mathbf{o}_9$ | $\mathbf{o}_{10}$ | $\mathbf{o}_{11}$ | $\mathbf{o}_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{o}_1$ | 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 |

Table 4.2: Ranked Neighborhood Matrix for Table 4.1a

Table 4.2 shows the *ranked neighborhood matrix* for the dataset in Table 4.1a. It has 12 rows and 12 columns representing the objects in the dataset. Each row is the ranked vector of the respective object: compare the row of $\mathbf{o}_6$ to Figure 4.1b.

The ranked matrix preserves the neighborhood relations in the original dataset. If a set of objects are close to each other, their respective *rank*s will be under a certain threshold. In Table 4.2, the mutual ranks of the 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 dataset, we can see that both of these object sets 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. Table 4.1b, have values below 6 at the first row of Table 4.2. Nevertheless, in contrast to the set-based approach of cartification's neighborhoods, ranked neighborhoods preserve the 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 4.6** (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. I.e., 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 $O$ as*

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

*where $\theta$ 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 $\theta$ in the corresponding tile.

| | $\mathbf{o}_{12}$ | $\mathbf{o}_{11}$ | $\mathbf{o}_9$ | $\mathbf{o}_5$ | $\mathbf{o}_6$ | $\mathbf{o}_{10}$ | $\mathbf{o}_2$ | $\mathbf{o}_7$ | $\mathbf{o}_3$ | $\mathbf{o}_4$ | $\mathbf{o}_1$ | $\mathbf{o}_8$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{o}_{12}$ | 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 |

Table 4.3: Ordered ranked neighborhood matrix for $a_1$

Note that in cartification, 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 $\theta$ 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.

### 4.3.2 Properties

The problem of ranked tiling was studied in earlier work by Le Van et al. [104]. However, given the origin of the ranked matrix studied in ranked cartification, we can use a more efficient algorithm in this particular setting. This algorithm relies on the particular properties of the sorted ranked neighborhood matrix.

**Definition 4.7** (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 value on attribute $A_d$.*

For example, if we sort the rows and columns of the ranked neighborhood matrix in Table 4.2 according to the object values in $A_1$, then we will have the matrix in Table 4.3.

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

**Property 4.1.** *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 4.2.** *In a row, minimum values are next to each other and this neighborhood involves the diagonal. Formally, for the values in the $i^{th}$ row, the following statements hold:*

1. *for $j$ and $k$ with $i < j < k$:* $\mathbf{M}_{ij} < \mathbf{M}_{ik}$*;*
2. *for $j$ and $k$ with $j < k < i$:* $\mathbf{M}_{ij} > \mathbf{M}_{ik}$.

*Proof.* By definitions 4.4 and 4.5, each row of $\mathbf{M}$ is a ranked neighborhood, i.e., $\mathbf{M}_{ij} = \mathbf{N}(\mathbf{o}_i)_j$. Since $\mathbf{M}$ is a *sorted* ranked neighborhood matrix, if the columns stand for the objects $\mathbf{o}_{a_0}, \mathbf{o}_{a_1}, \cdots, \mathbf{o}_{a_n}$, then $i < j \implies \mathbf{o}_{a_i} <= \mathbf{o}_{a_j}$. Consequently, for $\mathbf{o}_{a_i} < \mathbf{o}_{a_j} < \mathbf{o}_{a_k}$ and $\mathbf{o}_{a_x} \in \mathscr{DB}$,

$$|\{\mathbf{o}_{a_x} | \delta(\mathbf{o}_{a_i}, \mathbf{o}_{a_x}) < \delta(\mathbf{o}_{a_i}, \mathbf{o}_{a_j})\}| < |\{\mathbf{o}_{a_x} | \delta(\mathbf{o}_{a_i}, \mathbf{o}_{a_x}) < \delta(\mathbf{o}_{a_i}, \mathbf{o}_{a_k})\}|$$

and therefore $i < j < k \implies \mathbf{M}_{ij} < \mathbf{M}_{ik}$. Second statement of the property can be proven in a similar way.

□

We prove a similar property for the columns.

**Property 4.3.** *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 4.2, $i < j < k \implies \mathbf{o}_{a_i} < \mathbf{o}_{a_j} < \mathbf{o}_{a_k} \implies \mathbf{M}_{ji} \leq M_{ki}$. Here, the additional argument is that by moving lower down or higher up 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 farther away from the object $\mathbf{o}_i$. □

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

**Property 4.4.** *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 4.2 and 4.3. □

All of these properties can be observed in Table 4.3: (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 $\mathbf{o}_6$.

### 4.3.3 Algorithm

Property 4.4 allows us to develop an efficient search algorithm: 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 7 for finding a minimal tile. SQUARETILER takes the sorted ranked neighborhood matrix and the threshold $\theta$ 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 4.2 and 4.3.

**Algorithm 7** SQUARETILER: Tiling on Ordered Ranked Matrix

**Input:** $\mathbf{M}$: Rank Matrix, $\theta$
**Output:** $min\_t$: Minimum tile

$min\_t := (0,0)$ // *Minimum tile*
$min\_ts := -\theta$ // *Minimum tile sum*
**for** $i := 1$ **to** $n$ **do**
  $ts := \mathbf{M}_{ii} - \theta$ // *Tile sum*
  **for** $j := i+1$ **to** $n$ **do**
    $ns := (\mathbf{M}_{ij} - \theta) + (\mathbf{M}_{(i+1)j} - \theta) + \cdots + (\mathbf{M}_{jj} - \theta)$
      $+ (\mathbf{M}_{ji} - \theta) + (\mathbf{M}_{j(i+1)} - \theta) + \cdots + (\mathbf{M}_{j(j-1)} - \theta)$
    **if** $ns > 0$ **then**
      *Continue with the next i*
    $ts := ts + ns$
    **if** $ts < min\_ts$ **then**
      $min\_ts := ts$
      $min\_t := (i, j)$
**return** $min\_t$

## 4.4 Experiments

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.[2]

### 4.4.1 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 generate 6 synthetic datasets. Properties of these dataset are shown in Table 4.4. **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 run Cartification [11] and CARTIRANK with different neighborhood sizes, namely $\theta$ and $k$, and report the quality of the found clusters. Since we know the true clusters beforehand, we use the supervised F1 score to assess the quality of the found clusters. The F1 score is the harmonic mean of the *precision* and *recall* between 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

[2] `http://adrem.uantwerpen.be/cartirank`

| 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 |

Table 4.4: Datasets for robustness tests

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 known clusters are among the clusters found using CARTIRANK and Cartification. What we call F1 in this chapter is called *F1-Recall* in Section 2.5.

Figure 4.2 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 $\theta$ 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.

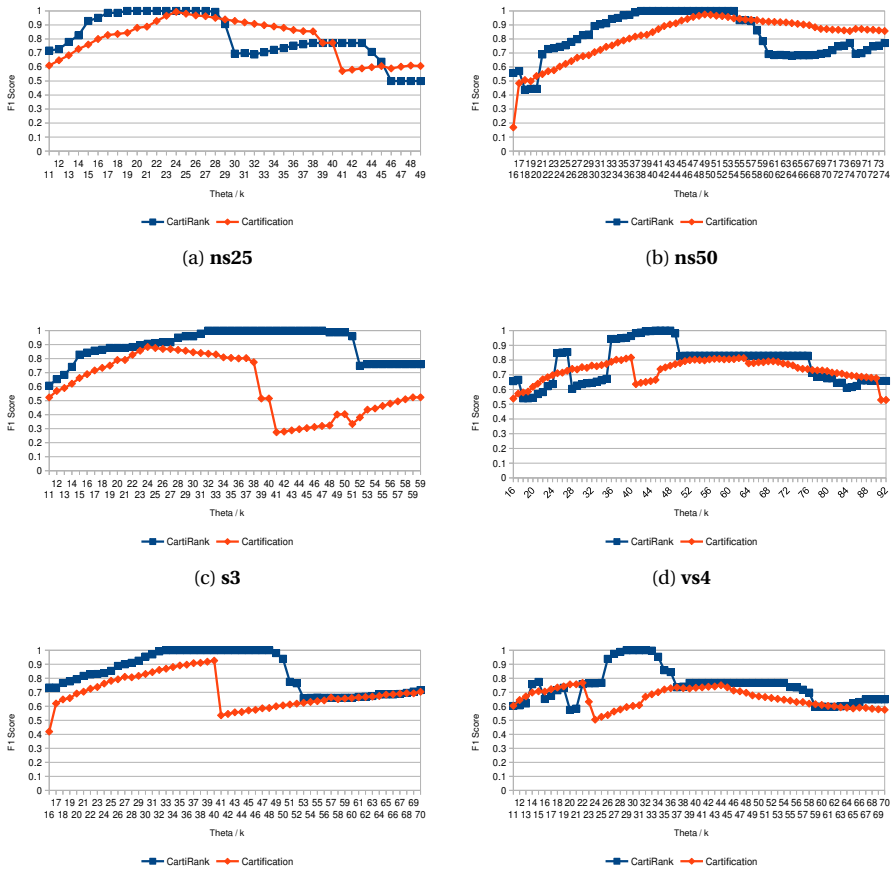
Our experiments with a wide range of values for the minimum length parameter $\mu$ show that the effect of this parameter is negligible. For the same $\theta$, in these experiments, the $\mu$ parameter does not change the outcome at all, or changes the F1 score of the found clusters not more than 0.05.

### 4.4.2 Subspace Clustering

To evaluate subspace cluster finding capabilities of the methods, we generated a set of datasets. Each dataset has 5 sets of overlapping clusters which are hidden in 5 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 4.4, 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 $A_k$.

We generate datasets with similar characteristics multiple times, run the al-




(a) **ns25** (b) **ns50**

(c) **s3** (d) **vs4**

(e) **vs5** (f) **vs6**

Figure 4.2: F1 Scores of the methods for a range of parameters

gorithms on them and report the average results. Generation parameters for the datasets are shown in Table 4.5. After generating datasets using these parameters, we add 2 redundant dimensions with random values and add 5% random noise to them.

The F1 scores of the found clusters for a range of parameters are shown in Figure 4.5. As in one-dimensional clustering, the 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. Here is the catch: Figure 4.6 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 into a good use in practical scenarios. Clusters found by CARTIRANK are orders of magnitude less redundant.

We compare the quality of CARTIRANK with two of the state-of-the-art subspace clustering algorithms. Figure 4.3 shows the F1 scores of CARTIRANK, Cartification,

| Name | # of clusters | Attributes/cluster | Cluster Sizes | # of Attributes |
|---|---|---|---|---|
| **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 |

Table 4.5: Generation parameters for subspace cluster datasets

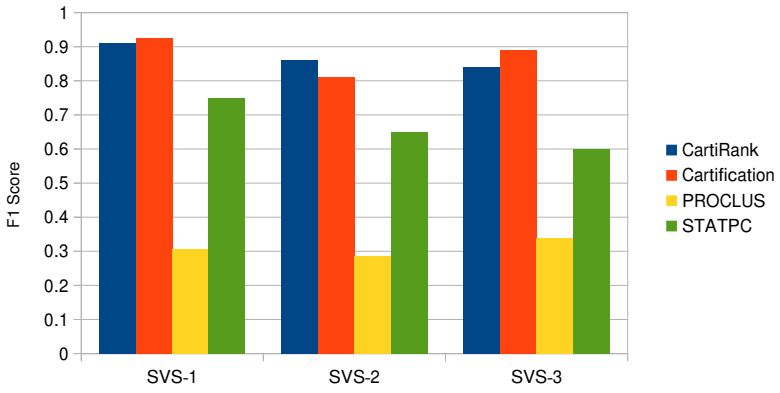



Figure 4.3: The best clusters found by the different algorithms

PROCLUS [5], and STATPC [75] 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 for our datasets it does not work well. Moreover, optimizing the complex parameters of STATPC is not trivial.

## 4.5 Conclusion

In this chapter, we tackle the problem of detecting subspace clusters. Following our previous work [10, 11], we exploited local neighborhoods by transforming a relational database into sorted ranked neighborhood matrices. We studied the properties of this matrix and show the relation between minimal tiles in the matrix and the cluster structures in the data. We proposed 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.

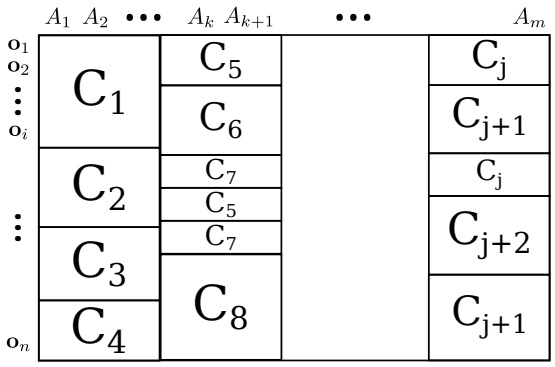
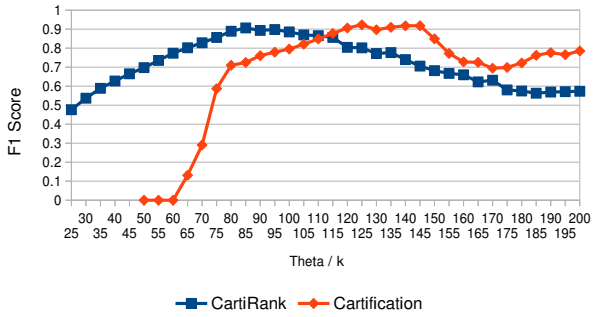
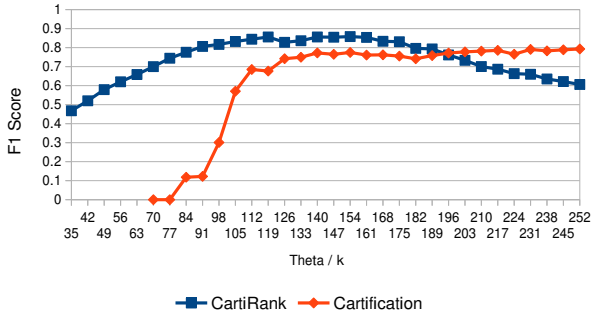


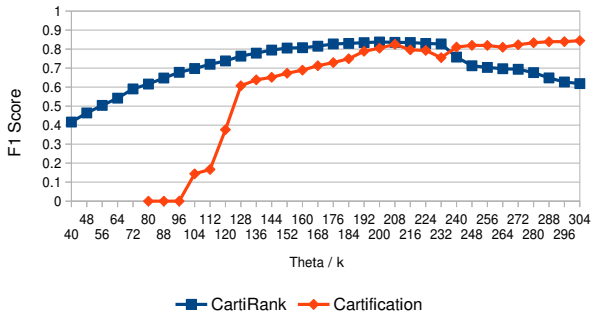

Figure 4.4: Subspace clusters in synthetic datasets




(a) SVS-1

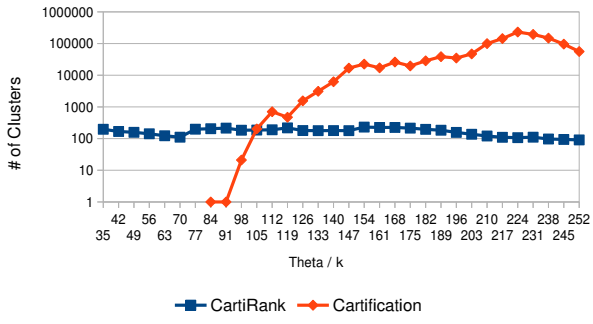



(b) SVS-2

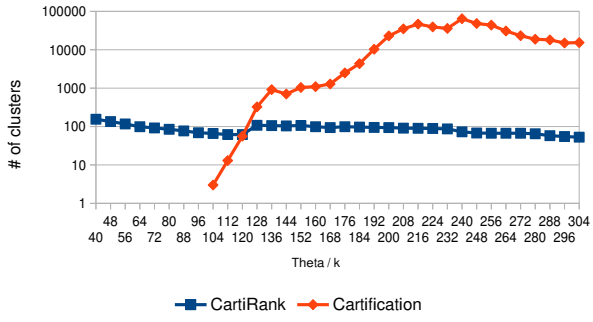



(c) SVS-3

Figure 4.5: Quality of the subspace clusters




(a) SVS-2




(b) SVS-3

Figure 4.6: The number of clusters found by the methods

CHAPTER 5

# Visual Interactive Neighborhood Mining on High Dimensional Data

*Cluster analysis is widely used for explorative data analysis, however, it is not trivial to select the right method and optimal parameters. Moreover, not all clustering methods can work with raw or dirty data. In this paper, we introduce an interactive data exploration tool, VINeM, which combines interactive mining with unsupervised tools by exploiting an intuitive neighborhood-based visualization technique. Local neighborhood-based visualization is useful not only for analyzing multiple (dis-)similarity measures but also for effectively discarding noise. VINeM works well with high dimensional data and can be used to find subspace clusters.*[1]

[1] This chapter is based on work published in KDD-IDEA 2015 as " Visual Interactive Neighborhood Mining on High Dimensional Data" by Emin Aksehirli, Bart Goethals, and Emmanuel Müller [74].

## 5.1 Introduction

Explorative data analysis is an important tool that is used in both academic and industrial research areas. In recent years computational discoveries have become more affordable than the actual experiments, thanks to Moore's law. Therefore, researchers try to infer as much as they can from a limited set of data and refer to experiments only for conclusive results [46].

In most data exploration scenarios, little is known about the data, which means that the data is not annotated, and hence, not a good fit for supervised learning tasks, e.g. classification. Unsupervised methods are more suitable for data exploration tasks since they can work with limited initial knowledge about the data.

Unsupervised methods have their own challenges, foremost of which is the selection of the method that best fits the data at hand. This is not easy considering the thousands of available clustering methods in the literature [52]. Moreover, the selection of the algorithm is the first step of the process, finding its optimal parameters is the next challenge which requires an understanding of the data.

To get a grasp of the data, one can rely on the most general and the least complicated techniques, such as descriptive statistics or visualizations. Descriptive statistics depend heavily on assumptions about the data. Not only determining them is not easy, but also, statistics can be misleading if those assumptions do not hold. Data visualization techniques, however, are easy to use by non-expert users and provide significant information. Therefore, data visualisation has always been an attractive research area [58]. Furthermore, using suitable visualizations, the user can be involved in the critical steps to improve the discovery process [40, 22]. On the other hand, existing tools either

1. are not capable of visualizing different views on the data [94],
2. do not integrate interactive and unsupervised mining methods [43, 78, 23],
3. are not aware of clusters that exist in subsets of the dimensions [95],
4. or, are not designed to find the clusters [101].

Data from observations are rarely usable as they are. Before starting the actual analysis, analysts should deal with additional data cleaning steps, such as removing noise and dealing with missing values. Methods that are robust enough to cope with dirty or unstructured data can significantly shorten this tedious process.

In this chapter, we introduce an interactive high dimensional data analysis tool that can visualise different views on the data in a unifying framework while seamlessly integrates unsupervised and interactive mining tasks. In summary, the contributions of this chapter are as follows:

- Neighborhood-based unifying data visualization,
- Micro cluster based relevant dimension detection,
- GUI application that combines interactive data exploration with automated tools,
- Use case scenarios for various data exploration tasks.

In Section 5.2, we discuss the properties of high-dimensional data space in terms of clustering. Then, we introduce a neighborhood-based data representation that can cope with high-dimensional and noisy data while being easily visualizable (Section 5.3). In Section 5.4, we present our software tool that exploits this representation to make the data more available to the user, both for understanding and for wrangling. We explore two use-case scenarios in Section 5.5 and conclude in Section 5.6.

## 5.2 Clustering High Dimensional Data

As a result of the advances in the data gathering and data storing technologies, we can associate many attributes with a single data object. Although more data may provide us with new insights, it may also hinder the discovery process by cluttering the interesting relations with redundant information. Furthermore, since the data objects become more and more alike with an increasing number of dimensions [20], the traditional definition of similarity becomes meaningless in high dimensional data, and hence, clustering methods that depend on the similarity between objects fail to cope with high dimensional data.

On the other hand, similarities of the objects according to a subset of attributes can still be meaningful. *For example, if a group of people have similar values for a specific bio-marker and they show a tendency for drug abuse, then other attributes such as the eye color, weight or sex are probably irrelevant.* Therefore, meaningful knowledge in high dimensional data is often extracted as a tuple of similar objects and the attributes that they are similar in. Such information is called a *subspace cluster*.

Formally, a data object $\mathbf{o}$ is defined as a vector over a set of attributes $\mathscr{A}$. A dataset $\mathscr{DB}$ is a collection of data objects. A cluster is a set of objects $C \subset \mathscr{DB}$. A subspace cluster is defined as a tuple of an object set and an attribute set, $SC = (C, A)$ where $A \subseteq \mathscr{A}$.

High dimensionality poses a problem during the visualization as well. Feature selection techniques, such as PCA [86] and MDS [26], are used in the literature to represent data in a lower dimensional space. However, feature selection is done on the whole dataset and can therefore easily miss the subspace clusters [78]. Reducing the dimensions while keeping the cluster structures is also proposed [90, 102, 68], but requires a computationally expensive pre-processing which also depends on the assumptions on the data. Therefore, they are not suitable for interactive data exploration settings.

Scatter plot matrices show a 2D matrix of scatter plots for each pair of dimensions. Although they are useful to get a grasp of the relatively lower dimensional data, it gets harder to interpret them for the high dimensional data because the number of charts is a combinatorial function of the number of dimensions. Parallel coordinates represents the relations of objects in different projections. They provide an interactive exploration environment but they cannot be used for non-univariate projections, i.e., they can represent only 1D projections [50].

## 5.3 Neighborhood Database

### 5.3.1 Object Neighborhoods

"Tell me who your friends are, and I will tell you who you are." The core concept of this famous phrase is successfully applied to many cases of data analysis. Neighborhoods, i.e. *friends*, of data objects provide robust assessment of the similarity, they can even be more accurate than the actual features in some cases [54]. Neighborhoods are a good estimator for determining class labels [41], or whether an object is an outlier [24]. As they are good at preserving the local relations, they are used to overcome the problems of high-dimensionality [10].

**Definition 5.1** (Neighborhood)**.** *The neighborhood of an object* $\mathbf{o}$ *is a set of objects that are similar to* $\mathbf{o}$ *and it is denoted by* $N(\mathbf{o})$.

**Definition 5.2** ($\varepsilon$-neighborhood)**.** *The radius-based neighborhood,* $\varepsilon$*-Neighborhood, of an object* $\mathbf{o}$ *is defined as the set of all objects that are more similar to* $\mathbf{o}$ *than a certain scalar value. Formally, let* $\delta : \mathscr{DB}^2 \rightarrow \mathbb{R}_+$ *be a dissimilarity measure,* $\varepsilon \in \mathbb{R}_+$ *and* $\mathbf{o}, \mathbf{p} \in \mathscr{DB}$*,*

$$\varepsilon\text{-}N(\mathbf{o}) = \{\mathbf{p} | \delta(\mathbf{o}, \mathbf{p}) < \varepsilon\}$$

**Definition 5.3** ($k$-Nearest Neighborhood)**.** *Let* $NN_k(\mathbf{o})$ *represent the* $k^{th}$ *closest object to* $\mathbf{o}$*, the* $k$*-nearest neighborhood of* $\mathbf{o}$ *is:*

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

Since the $k$-nearest neighborhood uses a relative similarity threshold, it is more robust for assessing the similarities in heterogeneous data than $\varepsilon$-neighborhood. Note that, we use the concept of generic *neighborhood* (Definition 5.1) if the type of the neighborhood is irrelevant.

**Definition 5.4** (Neighborhood Database)**.** *The Neighborhood Database (*$\mathscr{ND}$*) is the collection of neighborhoods of all objects. Figure 5.3b shows the neighborhood database of the dataset in Figure 5.3a.*

Since *a cluster is defined as a group of similar data objects*, there is a clear connection between the cluster structures in the data and the neighborhoods. Fig. 5.1 shows two clusters that are clearly separated, i.e., each object in a cluster is more similar to objects in the same cluster than to objects in the other clusters. For each object $\mathbf{o} \in C_i$, its $k$-nearest neighborhood with $k$ equal to the size of its cluster, $|C_i|$-$NN(\mathbf{o})$, is equal to the cluster itself. Formally, if cluster $C_i$ is clearly separated, then $|C_i|\text{-}NN(\mathbf{o}) = C_i, \forall \mathbf{o} \in C_i$. If the clusters are not separated, e.g., as shown in Figure 5.2, the most of the objects in $C_i$ still include the whole cluster, provided that the neighborhoods are large enough.

Neighborhoods of the objects that are in the same cluster are either the same or share a large set of objects. Therefore, we can find the cluster structures by finding the repetitive patterns in the neighborhoods [11, 10]. Although the repetitive patterns can be detected by unsupervised tools, selecting the correct parameters may not be trivial. On the other hand, through proper representation, a human can spot repetitive patterns even in noisy settings and provide the intuition that would substantially improve the accuracy and the speed.

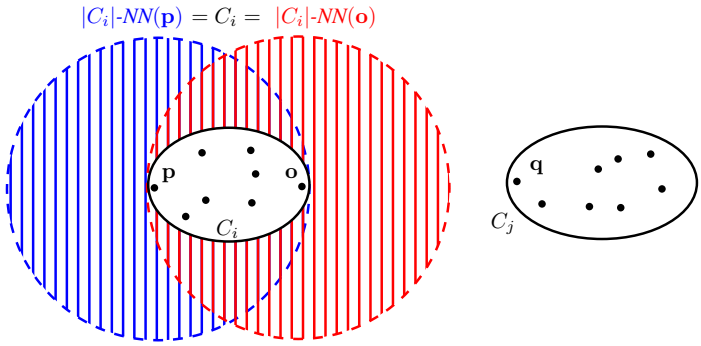



Figure 5.1: Case of separable clusters

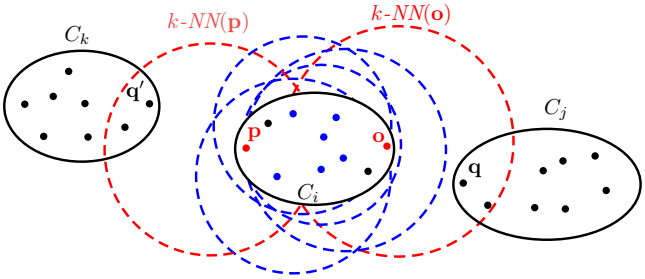



Figure 5.2: Case of non-separable clusters

### 5.3.2 Representation

**Definition 5.5** (Neighborhood Matrix)**.** *A neighborhood matrix is a binary adjacency matrix where columns and rows respectively represent objects and their neighborhoods. If the object in column $j$ is in the neighborhood of object $i$, then the corresponding cell, i.e. $i^{th}$ row and $j^{th}$ column, is 1 and 0 otherwise.*

Fig. 5.3a shows a small data set of 6 data objects with 2 attributes. We can identify two cluster structures: $\{\mathbf{o}_1, \mathbf{o}_4, \mathbf{o}_6\}$ and $\{\mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_5\}$. Fig. 5.4a shows the neighborhood matrix for the 3 nearest neighborhoods of the dataset. While the repetitive neighborhoods are present, it is hard to see and mine. On the other hand, by using the order in the data, we can create a better representation that is easier to interpret and compute. For example, Fig. 5.4c shows the same neighborhoods, where objects and neighborhoods are ordered according to $A_1$.

The neighborhood representation is compatible with the pixel-based visualization [58]. A *neighborhood matrix* can be represented graphically as an $n \times n$ square, where a pixel is white if the value of the corresponding cell is 1 and black otherwise. Figures 5.4b and 5.4d are the pixel-based representation of the neighborhood matrices in figures 5.4a and 5.4c, respectively. Cluster structures in the data are stunningly visible in Fig. 5.4d.

| | $A_1$ | $A_2$ |
|---|---|---|
| $\mathbf{o}_1$ | 4000 | 3200 |
| $\mathbf{o}_2$ | 5 | 6 |
| $\mathbf{o}_3$ | 7 | 6 |
| $\mathbf{o}_4$ | 3000 | 4000 |
| $\mathbf{o}_5$ | 6 | 8 |
| $\mathbf{o}_6$ | 5000 | 4200 |

(a) Data

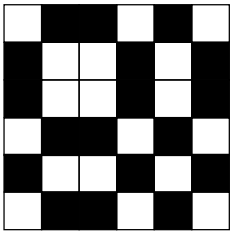
| | |
|---|---|
| 3-$NN(\mathbf{o}_1)$ | $\{\mathbf{o}_1, \mathbf{o}_4, \mathbf{o}_6\}$ |
| 3-$NN(\mathbf{o}_2)$ | $\{\mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_5\}$ |
| 3-$NN(\mathbf{o}_3)$ | $\{\mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_5\}$ |
| 3-$NN(\mathbf{o}_4)$ | $\{\mathbf{o}_1, \mathbf{o}_4, \mathbf{o}_6\}$ |
| 3-$NN(\mathbf{o}_5)$ | $\{\mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_5\}$ |
| 3-$NN(\mathbf{o}_6)$ | $\{\mathbf{o}_1, \mathbf{o}_4, \mathbf{o}_6\}$ |

(b) Neighborhood database

Figure 5.3: An example dataset and its neighborhood database

| | $\mathbf{o}_1$ | $\mathbf{o}_2$ | $\mathbf{o}_3$ | $\mathbf{o}_4$ | $\mathbf{o}_5$ | $\mathbf{o}_6$ |
|---|---|---|---|---|---|---|
| 3-$NN(\mathbf{o}_1)$ | 1 | 0 | 0 | 1 | 0 | 1 |
| 3-$NN(\mathbf{o}_2)$ | 0 | 1 | 1 | 0 | 1 | 0 |
| 3-$NN(\mathbf{o}_3)$ | 0 | 1 | 1 | 0 | 1 | 0 |
| 3-$NN(\mathbf{o}_4)$ | 1 | 0 | 0 | 1 | 0 | 1 |
| 3-$NN(\mathbf{o}_5)$ | 0 | 1 | 1 | 0 | 1 | 0 |
| 3-$NN(\mathbf{o}_6)$ | 1 | 0 | 0 | 1 | 0 | 1 |

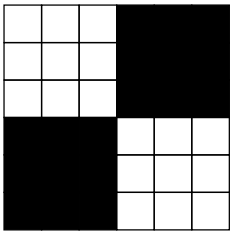
(a) Neighborhood matrix



(b) Visual Fig. 5.4a

| | $\mathbf{o}_2$ | $\mathbf{o}_5$ | $\mathbf{o}_3$ | $\mathbf{o}_4$ | $\mathbf{o}_1$ | $\mathbf{o}_6$ |
|---|---|---|---|---|---|---|
| 3-$NN(\mathbf{o}_2)$ | 1 | 1 | 1 | 0 | 0 | 0 |
| 3-$NN(\mathbf{o}_5)$ | 1 | 1 | 1 | 0 | 0 | 0 |
| 3-$NN(\mathbf{o}_3)$ | 1 | 1 | 1 | 0 | 0 | 0 |
| 3-$NN(\mathbf{o}_4)$ | 0 | 0 | 0 | 1 | 1 | 1 |
| 3-$NN(\mathbf{o}_1)$ | 0 | 0 | 0 | 1 | 1 | 1 |
| 3-$NN(\mathbf{o}_6)$ | 0 | 0 | 0 | 1 | 1 | 1 |

(c) Neighborhood matrix of $A_1$



(d) Visual Fig. 5.4c

Figure 5.4: Neighborhood matrices of the example dataset

Advantages of this representation include: (1) The representation is intuitive. (2) Individual objects are visible, i.e., they can be differentiated from their surroundings, regardless of the local density. (3) Since the repetitive structures stand out, cluster structures are visible. (4) It allows interactive mining since the individual objects are visible. (5) The relations in the representation are explainable because the original attributes are not distorted, e.g., the matrix is not translated as in PCA. (6) Creating the representation is not computationally expensive compared to dimensionality reduction techniques [68]. (7) Compared to graph representation, it provides a scalable and extendible solution, e.g., pixel sizes can be determined by the screen and the data size.

### 5.3.3 Mining the Neighborhoods

The evaluation of clusters is an important aspect when finding meaningful clusters. As we discussed in Section 5.3, objects in the same cluster repetitively co-occur in the neighborhoods of other objects. In this regard, we use the *support* of a cluster as a measure of repetitiveness of an object set in the neighborhood database. Therefore, clusters can be detected by finding the object sets that have a high *support*.

**Definition 5.6** (Support($\sigma$))**.** *The support of an object set, i.e. cluster, is the number of neighborhoods in which the objects occur together. Formally,*

$$\sigma(C) = |\{\mathbf{o} \mid C \subseteq N(\mathbf{o}), N(\mathbf{o}) \in \mathscr{ND}\}|$$

Support is a useful measure also for interactive mining. As shown in Section 5.4, supports of an object set in different neighborhood databases give an overview of the cluster formations. Moreover, this relation between the clusters and the support in the neighborhoods can directly be mapped to frequent itemset mining [9], so that the whole literature of frequent itemset mining methods becomes available for cluster analysis [10].

One clear advantage of using neighborhood databases is their unifying representation. All of the $\mathscr{ND}$s, regardless of the underlying (dis-)similarity measure, share the same properties, which means less context switches and more clarity for the user. For example, consider these measures: Euclidian distances on different subsets of attributes, cosine similarity on all numeric attributes, a measure for the boolean attributes, two different measures for the same categorical attribute. $\mathscr{ND}$s for each of them can be mined with the same set of tools because we are looking for the same kind of information, i.e., the objects that frequently co-occur in the neighborhoods.

As we discuss in Section 5.2, for high-dimensional data, local similarities are more meaningful than similarities in the whole data space. Therefore, bottom-up search is a widely used strategy to find subspace clusterings [75, 60]. In this regard, we propose to start the interactive analysis with the neighborhoods in one-dimensional projections and find the object sets that repetitively co-occur in the neighborhoods in different projections.

Our tool VINeM, cf. Section 5.4, includes two methods for unsupervised mining of neighborhood databases, both of which satisfy the time constraints of an interactive setting. ***Sampling Miner*** mines the dataset for a subset of all the frequently co-occurring object sets. It is based on a Monte Carlo process, and as such, it mines a predetermined number of maximally large object sets that has more support than a certain threshold. These object sets are counterparts of *maximal frequent itemset*s [18]. Although it is not guaranteed to be complete, it produces satisfactory results [10]. Details of *Sampling Miner* are discussed in Section 2.4. ***Fast Miner*** , which is an implementation of the CLON algorithm in Chapter 3, exploits the orders in an attribute to find the complete cluster structure [11]. It starts by mining the individual $\mathscr{ND}$s for a complete set of one-dimensional clusters. Then, each cluster is refined further by checking whether any of its subsets are clusters in other dimensions. It can only be used to mine neighborhoods of univariate measures.

Typically, subspace clusters overlap with each other both object-wise and attribute-wise. For example, a set of objects can form a cluster in dimensions 1 and 2

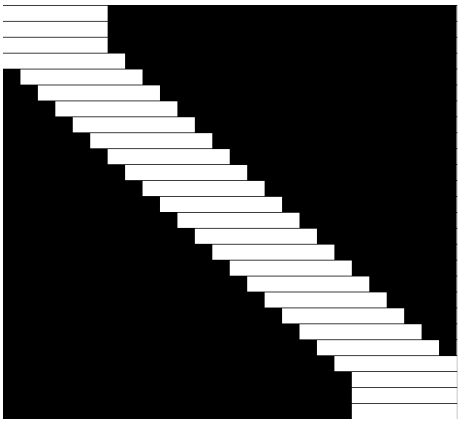


Figure 5.5: Neighborhoods of uniform data

while another, but not necessarily disjoint, set of objects can form a cluster in dimensions 2 and 3. Therefore, the relation between the attributes should be investigated by assessing localities instead of the whole domain. To find the similarities between attributes, we propose to use micro clusters, which have already been used to detect cluster structures [17, 49]. We mine a sample of micro clusters of size 5 in each attribute and the number of micro clusters shared by a pair of attributes becomes their similarity score. If a set of attributes are similar to each other, it can be worthwhile to investigate the neighborhoods of the combination of these dimensions. Even though the complete cluster structures are not visible in projections, micro clusters can effectively catch and aggregate local similarities, as we will show in Section 5.5.2.

Although sorted and non-sorted neighborhoods essentially contain the same information; in a visual setting, it is often easier to work with sorted neighborhoods. Unfortunately, sorted neighborhoods are possible only for univariate (one-dimensional) projections. For the interactive setting in VINeM, we approach the problem by *partially sorting* the $\mathscr{ND}$ by focusing on a subset of the objects. The partial sorting is done by selecting an object as a reference and sorting the remaining objects and neighborhoods according to their similarities to the reference object. As we show in Section 5.5.2, partial sorting provides enough visual information for the identification of cluster structures.

Even if there are no cluster structures in the data, there is a minimum amount of repetition in the neighborhoods. In the case of uniformly distributed data, each individual object appears in exactly $k$ neighborhoods, while the consecutive object sets of size $\frac{k}{2}$ appear in exactly $\frac{k}{2}$ neighborhoods. Neighborhoods for a uniform dataset are shown in Fig. 5.5. This observation is used as a key indicator to discard the projections that do not have cluster structures.

Outliers and noise objects exist in relatively sparse areas and do not occur frequently in neighborhoods of objects [98]. Therefore, noise can be easily identified in the neighborhood database as the objects with low support.

If a set of objects form cluster structures in multiple attributes, then their values are similar to each other in these attributes. Therefore, we can use the dispersion of

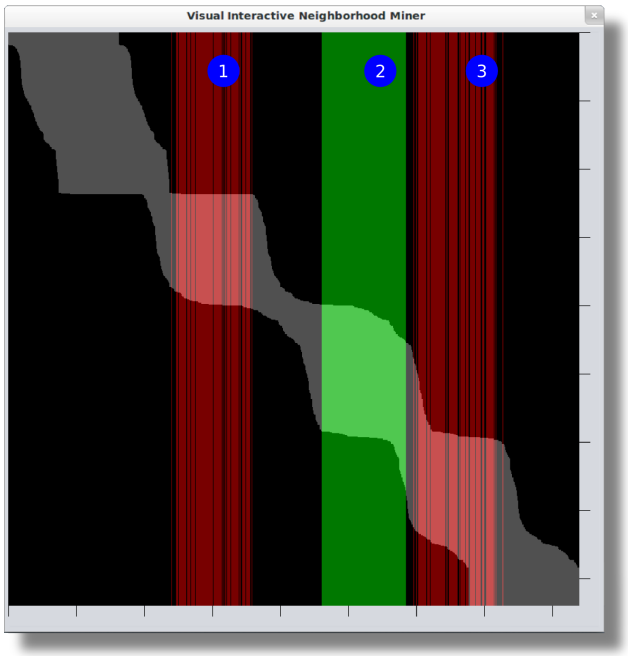



Figure 5.6: Neighborhood matrix

an object set in an attribute as an heuristic for cluster formation. Standard deviation and median absolute deviation (MAD) are widely used measures for dispersion. According to our observation, MAD gives more accurate results than the standard deviation.

## 5.4 Visualization

In this section we introduce VINeM, a data exploration tool that exploits the neighborhood database representation. An important design goal of VINeM is to provide a user friendly interface that allows a user to easily blend unsupervised tools with instant decisions. Therefore, all of the features are designed for human interaction while they can be manipulated by the unsupervised tools.

VINeM is implemented in Java using Swing as GUI toolkit. It is accessible along with a detailed user manual on the supplementary website.[2]

### 5.4.1 Neighborhood

The main interaction area of VINeM is the neighborhood panel where the neighborhood matrix is shown, cf. Figure 5.6. As discussed in Section 5.3.2, columns represent the objects and rows represent the neighborhoods.

[2] `http://adrem.uantwerpen.be/vinem`

Currently, two kinds of neighborhoods are supported: $\varepsilon$-neighborhood and $k$-nearest neighborhood. The *kNN* representation is the default because of its robustness. VINeM starts by showing the *kNN* neighborhood databases for each of the individual attributes. Initially, the matrix and the objects are sorted according to the shown attribute. The order of the objects can be observed in the selection list where the ids, or the names, of the objects are shown, cf. 4 in Figure 5.7.

The matrix representation can be manipulated using the following means:

*Dissimilarity measures for the projections.* In the initial setting, there is one neighborhood matrix per attribute, each of which represents the neighborhoods according to Euclidian distances per dimension. Which $\mathcal{ND}$ to show can be selected by using dropdown, cf. 10 in Figure 5.7.

*Object and neighborhood order.* The order of the objects are not necessarily dependent on the dissimilarity measure. For example, it is possible to sort the neighborhoods in attribute 1 according to attribute 2. On the other hand, the order and the measure are synchronized by default for convenience. The dimension that is used to order the objects can be selected using a slider, cf. 8 in Figure 5.7. If the synchronization is enabled, this will also change the dissimilarity measure.

*The type of the neighborhood assessment, $\varepsilon$ vs. kNN.* The parameter for neighborhood size ($k$) can be set for *kNN*, while the parameter for neighborhood radius ($\varepsilon$) is required for the $\varepsilon$-Neighborhood. For convenience, possible radius sizes are pre-computed per similarity measure and user selects among them. The type of the neighborhood and its parameters can be selected on the control panel, cf. 7 in Figure 5.7.

### 5.4.2 Interactivity

Interactivity of VINeM starts with parameter selection. The interface and the representation are updated instantly after each parameter change, allowing the user to experiment with parameters on a responsive interface.

The selection of objects is the first step of the interactive analysis. There are two intuitive ways to select objects: (1) Dragging the left mouse button on the neighborhood matrix highlights the columns in green and selects the corresponding objects, cf. 2 in Figure 5.6. (2) Selecting objects from the selection list combo box by using standard list selection techniques, cf. 4 in Figure 5.7. The main selection method is dragging on the matrix, the list selection is for fine tuning. There are three modes for the selection on matrix, cf. 5 in Figure 5.7. In “Select” mode, only the objects under the mouse are selected. The “And” mode selects the objects if they are already selected while the “Or” mode adds the new selection to the already selected objects.

A separate frame, cf. Figure 5.8, shows the information about the selected objects, such as the size of the selection, their support and dispersion in other $\mathcal{ND}$s. This information is used to determine the next $\mathcal{ND}$ to investigate. Selection of the objects is an essential part of the data analysis in VINeM. During the analysis, the selection is iteratively refined by removing the objects that do not belong to the cluster in the additional attributes, which results in a selection of similar objects according to some attributes, i.e., a subspace cluster.

Selected objects can be identified as a cluster by clicking the button “Cluster

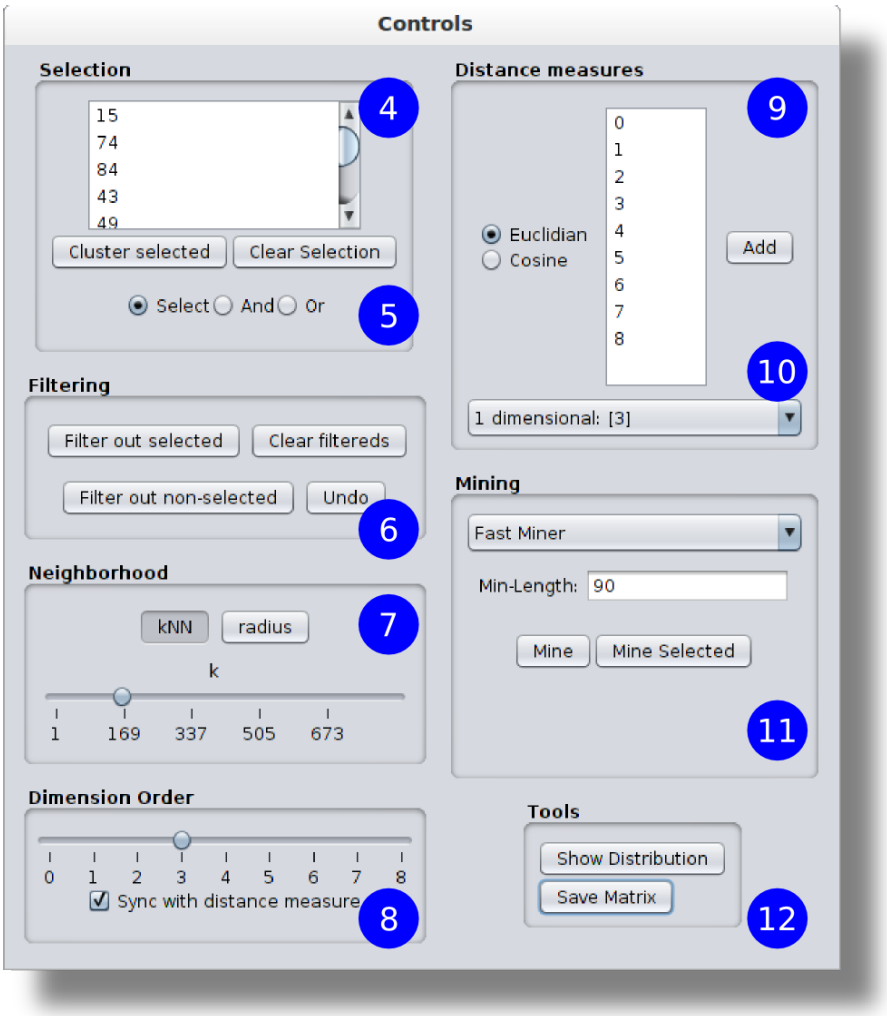



Figure 5.7: Control Panel

Selected", cf. 5 in Figure 5.7. The clusters that are identified either by the user or by an unsupervised tool are shown in the cluster list window, cf. Figure 5.9. Any of the identified clusters can be visualised on the neighborhood matrix along with the selected objects, so that the selection can be compared with the known clusters. Clusters are highlighted in red on the neighborhood matrix, cf. 1 and 3 in Figure 5.6. Clusters can be manipulated by adding or removing objects and if a substantial refinement is required, they can always be converted into selections. Any subset of clusters can be saved to a file for further study.

A set of objects can be filtered out, i.e. removed, from the view to focus on the objects under consideration, cf. 6 in Figure 5.7. Note that filtering removes the

objects and their neighborhoods just from the view, i.e., the neighborhoods are not re-computed. Therefore, it is fast and it does not change the data on the fly. Filtering comes in handy while performing a cross measure analysis since it hides the noise caused by the projection.

Neighborhood databases for new measures can be added by just a few mouse clicks using the GUI. To add an $\mathcal{ND}$, a similarity measure is selected along with the dimensions that it will be applied on, cf. 9 in Figure 5.7. After adding the new $\mathcal{ND}$, it can be viewed by selecting from the drop-down menu, cf. 10 in Figure 5.7.

$\mathcal{ND}$s of non-univariate similarity measures can be partially sorted. Right clicking on a column selects the corresponding object as the reference and sorts the other objects according to their similarities to this object, cf. Section 5.3.3.

### 5.4.3 Unsupervised Tools

VINeM is bundled with unsupervised tools to support the user during the data analysis, cf. 11 in Figure 5.7. They are seamlessly integrated with the interactive tools where applicable.

*Related dimension finder* mines the $\mathcal{ND}$ of each individual attribute for micro clusters as explained in Section 5.3.3. Parameters are updated with suggested values when the $\mathcal{ND}$ is modified, cf. Figure 5.11a. Relevancy scores between pairs of attributes, i.e. the number of shared micro clusters, are shown as a table. A thresholding slider is provided for visual assistance on detecting the highly related dimensions, cf. Figure 5.10.

The two miners that are introduced in the Section 5.3.3, namely *Fast Miner* and *Sampling Miner*, can be used for unsupervised mining. While *Fast Miner* requires

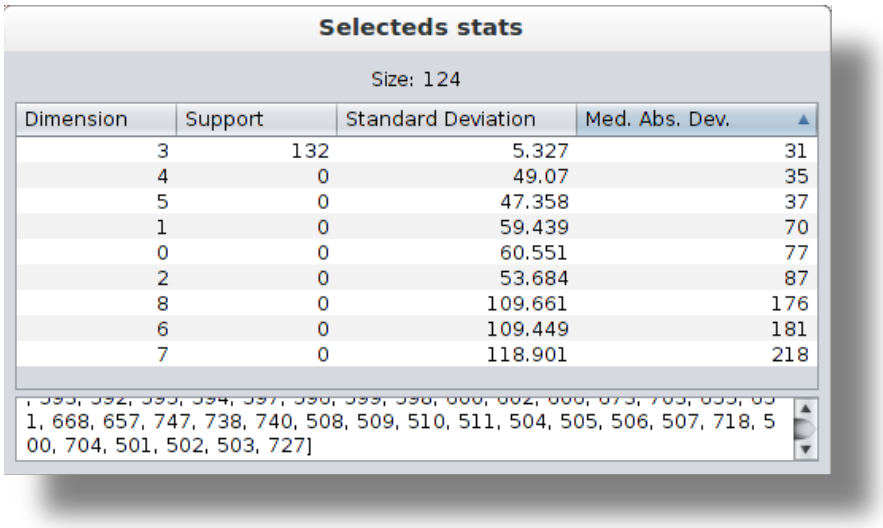



Figure 5.8: Basic information about the selection

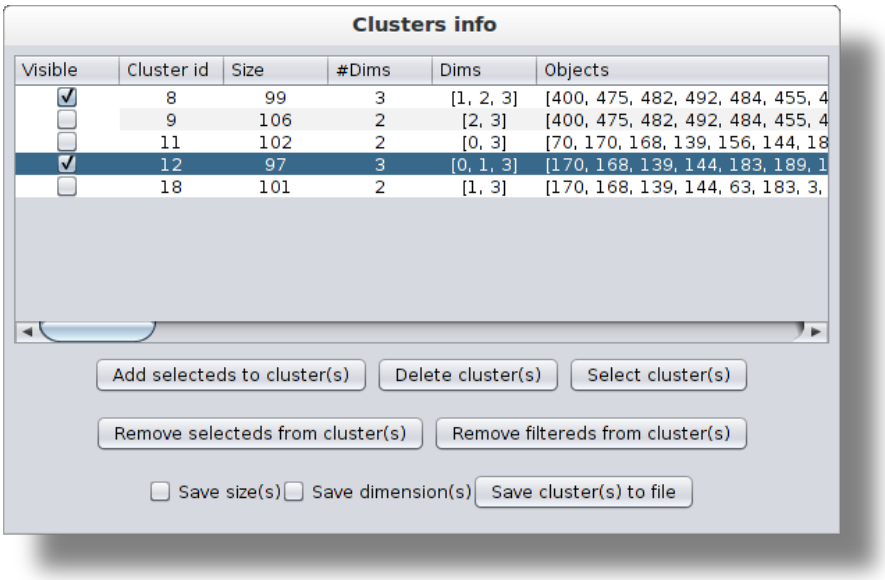



Figure 5.9: List of detected clusters

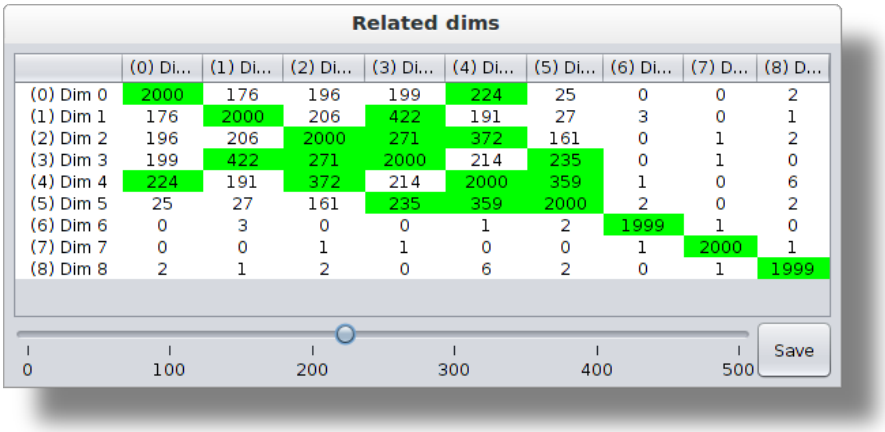
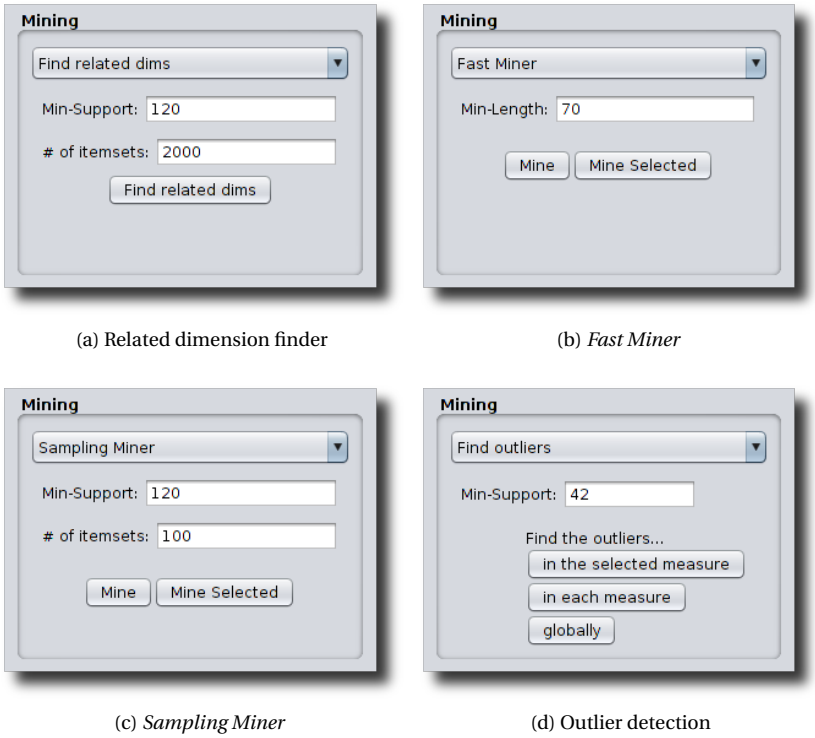



Figure 5.10: Relevancy score of dimensions

only one parameter which is the minimum length of a cluster, cf. Figure 5.11b; *Sampling Miner* requires two parameters: required minimum support of an object set to be identified as a cluster and number of samples, cf. Figure 5.11c. Both of the miners can be run either on the whole dataset or only on the selected objects, so that the objects that are not under consideration can be left out. The suggested values for the parameters are provided for a quick start. After the run, all of the found clusters




(a) Related dimension finder

(b) *Fast Miner*

(c) *Sampling Miner*

(d) Outlier detection

Figure 5.11: Parameters for miners

are added to the cluster list for further investigation.

Automated noise/outlier detection is also provided as a miner. The objects that have a support less than the threshold are identified as noise. The support of objects can be evaluated (1) *in the selected measure*, (2) *in each measure*, or (3) in *all measures*, cf. Figure 5.11d. Noise objects are added as a cluster. It is up to the user to filter them out of the view.

## 5.5 Application

### 5.5.1 Finding clusters in subspaces

Fig. 5.12 shows the steps of an interactive mining process. We start by finding a neighborhood size that makes the clusters visible in one of the $\mathcal{ND}$s. In this example, we can see the cluster structures in Dim(ension) 1 for a certain $k$ value (Step 1). We identify a one-dimensional cluster by selecting its objects (Step 2). The next step is to check other dimensions to find out whether any of the subsets forms a cluster there. When we switch to the $\mathcal{ND}$ of the Dim 2, objects in the cluster are still marked with red (Step 3).

Since we are looking for subsets of our cluster, we remove the remaining objects

by filtering them out (Step 4). We decide that the objects do not form cluster structures in Dims 2 and 3 because their filtered $\mathcal{ND}$s look like the $\mathcal{ND}$ of a uniform distribution, which is shown in Fig. 5.5 (Steps 4 and 5). Note that, although there are some repetitive neighborhoods in the filtered $\mathcal{ND}$ of Dim 2, they are too small to be identified as clusters. Filtered $\mathcal{ND}$ of Dim 4 looks interesting (Step 6), there is a large set of objects that co-occur in neighborhoods. We select these objects and identify them as a cluster in Dimensions 1 and 4.

### 5.5.2 Finding relevant dimensions

Fig. 5.13 shows the steps for an exploratory analysis on a 10-dimensional dataset. In this dataset, the clusters are not immediately visible in individual dimension projections. Step 1 of the figure shows 4 of the $\mathcal{ND}$s. There are signs of structures in Dims 0 and 2, which can probably be enhanced by modifying the neighborhoods size, while Dims 4 and 6 look like they lack any kind of structure. It is possible that the whole cluster structures are not visible in one-dimensional projections. Running the *related dimension finder* gives us the scores for each pair of dimensions. Then, we can interactively decide which of the dimensions are related to each other by examining the high scores (Step 2). It looks like the two sets of dimensions {0, 1, 2, 3} and {4, 5, 6, 7} share common structures.

With a few clicks, we add a new dissimilarity measure that assess the neighborhoods in the combination of dimensions 4, 5, 6, and 7 (Step 3). Although it is almost

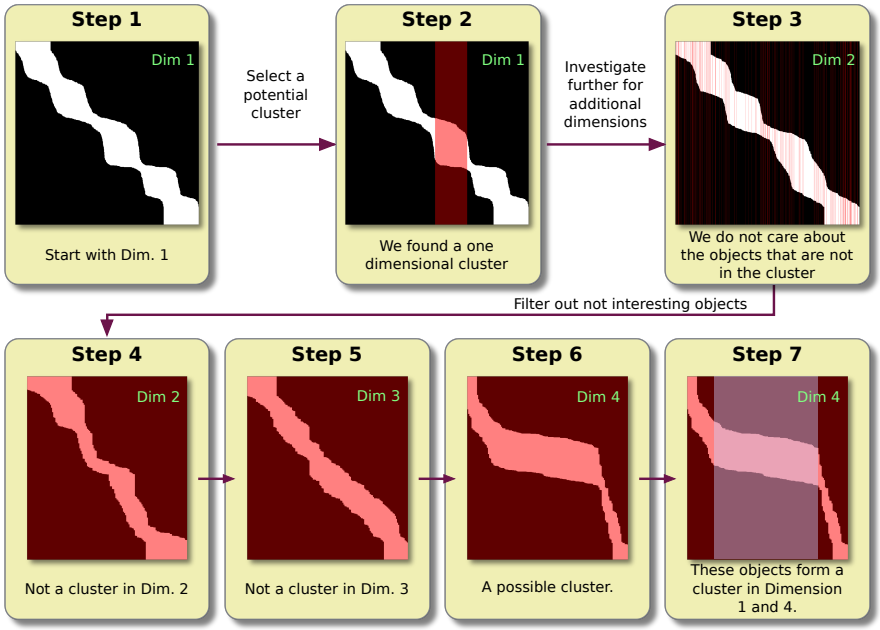



Figure 5.12: Subspace cluster detection

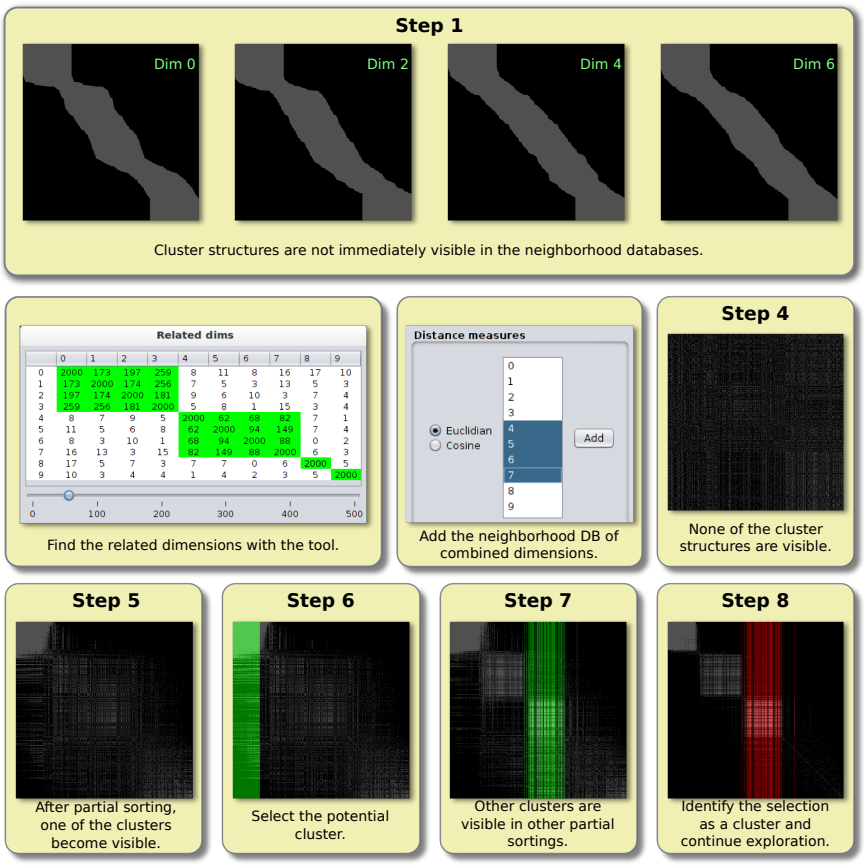



Figure 5.13: Finding relevant dimensions for cluster detection

impossible to spot the structures in the $\mathcal{ND}$ of the combined dimensions (Step 4), sorting the neighborhoods according to a reference object, i.e. partial sorting in Sec. 5.3.3, helps us to see the structure in the data (Step 5). The blob on the top left of the sorted $\mathcal{ND}$ represents a large set of objects that co-occur in the neighborhoods. We select these objects for further analysis (Step 6). Note that, since this sorting is according to only one object, some objects that are not inside the cluster can be in the blob by mistake, and they can be removed by using the partial orderings of the objects in the blob. We investigate further by sorting the $\mathcal{ND}$ according to an object that is not in the cluster candidate (Step 7), and now we can see some other structures in the $\mathcal{ND}$. We identify the selection as a cluster (Step 8), and then we continue our analysis with selecting a new cluster candidate. We can still modify the neighborhood parameters to improve the view. For example, cluster structures are more visible in Step 8 compared to Step 7, because of using $\varepsilon$-neighborhoods instead of *kNN*.

## 5.6 Conclusion

While visualisation and interactiveness are very important for exploratory data analysis, available tools fall short to satisfy all of its challenges. In this paper we show that local neighborhoods provide the means for both intuitive visualization and interactive mining of subspace clusters. We introduce VINeM, a platform-independent, visual and interactive data analysis tool that exploits the intuitive neighborhood-based representation to seamlessly combine a user friendly interactive interface with the unsupervised tools. We introduce a micro cluster based tool to find relevant dimensions and we provide example scenarios of exploratory data analysis to show the usefulness of our application.

CHAPTER 6

# Frequent Itemset Mining for BigData

*Frequent Itemset Mining (FIM) is one of the most well known techniques to extract knowledge from data. The combinatorial explosion of FIM methods become even more problematic when they are applied to BigData. Fortunately, recent improvements in the field of parallel programming already provide good tools to tackle this problem. However, these tools come with their own technical challenges, e.g. balanced data distribution and inter-communication costs. In this chapter*[1] *we investigate the applicability of FIM techniques on the MapReduce platform. We introduce two new methods for mining large datasets: Dist-Eclat focuses on speed while BigFIM is optimized to run on really large datasets. In our experiments we show the scalability of our methods.*

[1] This chapter is based on work published in IEEE BigData 2013 as "Frequent Itemset Mining for Big Data" by Sandy Moens, Emin Aksehirli, and Bart Goethals [74].

## 6.1 Introduction

Since recent developments (in technology, science, user habits, businesses, etc.) gave rise to production and storage of massive amounts of data, not surprisingly, the intelligent analysis of big data has become more important for both businesses and academics.

Already from the start, Frequent Itemset Mining (FIM) has been an essential part of data analysis and data mining. FIM tries to extract information from databases based on frequently occurring events, i.e., an event, or a set of events, is interesting if it occurs frequently in the data, according to a user given minimum frequency threshold. Many techniques have been invented to mine databases for frequent events [9, 18, 109]. These techniques work well in practice on typical datasets, but they are not suitable for truly BigData.

Applying frequent itemset mining to large databases is problematic. First of all, very large databases do not fit into main memory. In such cases, one solution is to use levelwise breadth first search based algorithms, such as the well known Apriori algorithm [9], where frequency counting is achieved by reading the dataset over and over again for each size of candidate itemsets. Unfortunately, the memory requirements for handling the complete set of candidate itemsets blows up fast and renders Apriori based schemes very inefficient to use on single machines. Secondly, current approaches tend to keep the output and runtime under control by increasing the minimum frequency threshold, automatically reducing the number of candidate and frequent itemsets. However, studies in recommendation systems have shown that itemsets with lower frequencies are more interesting [73]. Therefore, we still see a clear need for methods that can deal with low frequency thresholds in BigData.

Parallel programming is becoming a necessity to deal with the massive amounts of data, which is produced and consumed more and more everyday. Parallel programming architectures, and hence the algorithms, can be grouped into two major subcategories: shared memory and distributed (share nothing). On shared memory systems, all processing units can concurrently access a shared memory area. On the other hand, distributed systems are composed of processors that have their own internal memories and communicate with each other by passing messages [14]. It is easier to adapt algorithms to shared memory parallelism in general, but they are typically not scalable enough [14]. Distributed systems, in theory, allow quasi linear scalability for well adapted programs. However, it is not always easy to write or even adapt the programs for distributed systems, i.e., algorithmical solutions to common problems may have to be reinvented. Thanks to the scalability of distributed systems, not only in terms of technical availability but also cost, they are becoming more and more common.

While distributed systems are becoming more common, they are also becoming easier to use. In contrast, Message Passing Interface (MPI), one of the most common frameworks for scientific distributed computing, works efficiently only on low level programming languages, i.e., C and Fortran. It is known, however, that higher level languages are more popular in businesses [35]. Although they are not the most efficient in terms of computation or resources, they are easily accessible.

Fortunately, thanks to the MapReduce framework proposed by Google [28], which simplifies the programming for distributed data processing, and the Hadoop implementation by Apache Foundation [2], which makes the framework freely avail-

able for everyone, distributed programming is becoming more and more popular. Moreover, recent commercial and non-commercial systems and services improve the usability and availability for anyone. For example; the Mahout framework by Apache Foundation [3], which provides a straight forward usability for the most common machine learning techniques, can be set up and run on full-blown clusters on the cloud, having more than hundreds of processors in total, in less than 1 hour without prior experience with the system.

We believe that, although the initial design principles of the MapReduce framework do not fit well for the Frequent Itemset Mining problem, it is important to provide MapReduce with efficient and easy to use data mining methods, considering its availability and wide spread usage in industry.

In this chapter, we introduce two algorithms that exploit the MapReduce framework to deal with two aspects of the challenges of FIM on BigData: (1) Dist-Eclat is a MapReduce implementation of the well known Eclat algorithm [109], optimized for speed in case a specific encoding of the data fits into memory. (2) BigFIM is optimized to deal with truly BigData by using a hybrid algorithm, combining principles from both Apriori and Eclat, also on MapReduce. Implementations are freely available at `http://adrem.uantwerpen.be/bigfim`.

In this chapter, we assume familiarity with the most well-known FIM techniques [9, 109, 45] and with the MapReduce framework. In the next Section, we will shortly revise the basic notions, but we refer the reader to [39, 28] for good, short surveys on these topics.

## 6.2 Preliminaries

Let $\mathscr{I} = \{i_1, i_2, \ldots, i_n\}$ be a set items, a transaction is defined as $\mathscr{T} = (tid, X)$ where *tid*is a transaction identifier and $X$ is a set of items over $\mathscr{I}$. A transaction database $\mathscr{DB}$ is a set of transactions. Its vertical database $\mathscr{DB}'$ is a set of pairs that are composed of an item and the set of transactions $\mathscr{C}_j$ that contain the item:

$$\mathscr{DB}' = \{(i_j, \mathscr{C}_{i_j} = \{tid \mid i_j \in X, (tid, X) \in \mathscr{DB}\})\}.$$

$\mathscr{C}_{i_j}$ is also called the *cover* or *tid-list* of $i_j$.

The support of an itemset $Y$ is the number of transactions that contain the itemset. Formally,

$$supp(Y) = |\{tid \mid Y \subseteq X, (tid, X) \in \mathscr{DB}\}|$$

or, in vertical database format

$$supp(Y) = |\bigcap_{i_j \in Y} \mathscr{C}_{i_j}|.$$

An itemset is said to be *frequent* if its support is greater than a given threshold $\sigma$, which is called *minimum support* or *minsup* in short. Frequency is a monotonic property w.r.t. set inclusion, meaning that if an itemset is not frequent, none of its supersets are frequent. Similarly, if an itemset is frequent, all of its subsets are frequent.

Items in an itemset can be represented in a fixed order; without loss of generality let us assume that items in itemsets are always in the same order as they are in

$\mathscr{I}$, i.e., $Y = \{i_a, i_b, \ldots, i_c\} \Leftrightarrow a < b < c$. Then, the common $k$-*prefix* of two itemsets are the first $k$ elements in those sets that are the same, e.g., sets $Y = \{i_1, i_2, i_4\}$ and $X = \{i_1, i_2, i_5, i_6\}$ have a common 2-*prefix* of $\{i_1, i_2\}$. A *prefix tree* is a tree structure where each path represents an itemset, which is exactly the path from the root to the node, and sibling nodes share the same prefix. Note that all the possible itemsets in $\mathscr{I}$, i.e., the power set of $\mathscr{I}$, can be expressed as a prefix tree.

*Projected* or *conditional* database of an item $i_a$ is the set of transactions in $\mathscr{DB}$ that includes $i_a$.

Apriori [9] starts with finding the frequent items by making a pass over $\mathscr{DB}$ to count them. Then, it combines these frequent items to generate *candidate itemset*s of length 2, and counts their supports by making another pass over $\mathscr{DB}$, removing infrequent candidates. The algorithm iteratively continues to extend $k$-length candidates by one item and counts their supports by making another pass over $\mathscr{DB}$ to check whether they are frequent. Exploiting the monotonic property, Apriori prunes those candidates for which a subset is known to be infrequent. Depending on the minimum support threshold used, this greatly reduces the search space of candidate itemsets.

Eclat [109] traverses the prefix tree in a depth first manner to find the frequent itemsets. The monotonic property states that if an itemset, a path in the prefix tree, is infrequent then all of its subtrees are infrequent. Thus, if an itemset is found to be infrequent, its complete subtree is immediately pruned. If an itemset is frequent then it is treated as a prefix and extended by its immediate siblings to form new itemsets. This process continues until the complete tree has been traversed. Eclat uses a vertical database format for fast support computation. Therefore, it needs to store $\mathscr{DB}'$ in main memory.

MapReduce [28] is a parallel programming framework that provides a relatively simple programming interface together with a robust computation architecture. MapReduce programs are composed of two main phases. In the *map* phase, each mapper processes a distinct chunk of the data and produces key-value pairs. In the *reduce* phase, key-value pairs from different mappers are combined by the framework and fed to reducers as pairs of key and value lists. Reducers further process these intermediate parts of information and output the final results.

## 6.3 Related Work

Data mining literature employs parallel methods already since its very early days [8], and many novel parallel mining methods as well as proposals that parallelize existing sequential mining techniques exist. However, the number of algorithms that are adapted to the MapReduce framework is rather limited. In this section we will give an overview of the data mining algorithms on MapReduce. For an overview of parallel FIM methods in general, readers are kindly referred to [110, 65, 81, 82, 112].

Lin et al. propose three algorithms that are adaptations of Apriori on MapReduce [67]. These algorithms all distribute the dataset to mappers and do the counting step in parallel. *Single Pass Counting* (*SPC*) utilizes a MapReduce phase for each candidate generation and frequency counting steps. *Fixed Passes Combined-Counting* (*FPC*) starts to generate candidates with $n$ different lengths after $p$ phases and counts their frequencies in one database scan, where $n$ and $p$ are given as parameters. *Dy-*

*namic Passes Counting* (*DPC*) is similar to *FPC*, however $n$ and $p$ is determined dynamically at each phase by the number of generated candidates.

The *PApriori* algorithm by Li et al. [66] works very similar to *SPC*, although they differ on minor implementation details.

MRApriori [44] iteratively switches between vertical and horizontal database layouts to mine all frequent itemsets. At each iteration the database is partitioned and distributed across mappers for frequency counting.

The iterative, level-wise structure of Apriori based algorithms does not fit well into the MapReduce framework because of the high overhead of starting new MapReduce cycles. Furthermore, although Apriori can quickly produce short frequent itemsets thanks to breadth-first search, it can not handle long frequent itemsets efficiently because of the combinatorial explosion.

*Parallel FP-Growth* (*PFP*) [64] is a parallel version of the well-known pattern mining algorithm FP-Growth [45]. *PFP* groups the items and distributes their conditional databases to the mappers. Each mapper builds its corresponding FP-tree and mines it independently. Zhou et al. [113] propose to use frequencies of frequent items to balance the groups of PFP. The grouping strategy of PFP is not efficient neither in terms of memory nor speed. It is possible for some of the nodes to read almost the complete database into memory, which is very prohibitive in the field of BigData. Zhou et al. propose to balance distribution for faster execution using singletons, however as we discuss further in the paper, partitioning the search space using single items is not the most efficient way.

Malek and Kadima propose an aproximate FIM method that uses $k$-medoids to cluster transactions and uses the clusters' representative transactions as candidate itemsets [70]. The authors implemented a MapReduce version that parallelizes the support counting step.

The PARMA algorithm by Riondato et al. [87] finds approximate collections of frequent itemsets. The authors guarantee the quality of the frequent itemsets that are being found, through analytical results.

Some work exists that aims to improve the applicability of the MapReduce framework to data mining. For example, TWISTER [31] improves the performance between MapReduce cycles, or NIMBLE [38] provides better programming tools for data mining jobs. Unfortunately, none of these frameworks are as widely available as the original MapReduce. We therefore focus on an implementation that uses only the core MapReduce framework.

From a practical point of view, not many options are available to mine exact frequent itemsets on the MapReduce framework. To the best of our knowledge, PFP is the best, if not only, available implementation [3]. However, our experiments show that it has serious scalability problems. We discuss these issues in more detail in Section 6.6.2.

## 6.4 Search Space Distribution

The main challenge in adapting algorithms to the MapReduce framework is the limitation of the communication between tasks, which run as batches in parallel. All the tasks that have to be executed should be defined at start-up, such that nodes do not have to communicate with each other during task execution. Fortunately, the

prefix tree that is used by Eclat can be partitioned into independent groups. Each one of these independent groups can be mined separately on different machines. However, since the total execution time of an algorithm highly depends on the running time of the longest running sub task, balancing the running times is a crucial aspect for decreasing the total computation time [113].

In the case of Eclat, running time is a function of the number of frequent itemsets (FIs) in the partition. However, this can only be estimated with some boundaries [21, 37].

To estimate the computation time of a sub-tree, it has been proposed to use the total number of itemsets that can be generated from a prefix [109, 85], the logarithm of the order of the frequency of items [113], and the total frequency of items in a partition [8, 81] as estimators.

The total number of itemsets is not a good estimator for partitioning: since the monotonic property efficiently prunes the search space, large portions will never have to be generated or explored. On the other hand, assigning the prefixes to worker nodes using weights can provide a much better load balancing. However, the benefit of complicated techniques is not significant compared to a simple Round-Robin assignment.

Our experiments, shown in Section 6.6.1, suggest that the most important factor for obtaining a balanced partitioning is the length of the prefixes. In fact we can start Eclat at any depth $k$ of the prefix tree using all frequent $k$-FIs as seeds. These seeds can be partitioned among the available worker nodes. Partitioning the tree at lower depths (using longer FIs) provides better balancing. This behaviour is expected because longer FIs not only give more information about the data but they also avail the finer partitioning of the tree.

Our proposed algorithms exploit the inter-independence of sub-trees of a prefix tree and uses longer FIs as prefixes for a better load balancing. Both algorithms mine the frequent itemsets, in parallel, up to a certain length to find frequent $k$-length prefixes, $\mathscr{P}_k = \{p_1, p_2, \ldots, p_m\}$. Then, $\mathscr{P}_k$ is partitioned into $n$ groups $(\mathscr{P}_k^1, \mathscr{P}_k^2, \ldots, \mathscr{P}_k^n)$, where $n$ is the number of distributed workers.

Each group of prefixes, $\mathscr{P}_k^j$, is passed to a worker node and is used as a conditional database. Since each sub-tree for a distinct prefix is unique and not dependent on other sub-trees, each node can work independently without causing any overlaps. Frequent itemsets that are discovered by individual workers require no further post-processing.

Prefix trees are formed in a way that siblings are sorted by their individual frequency in ascending order. Formally, $I = (i_1, i_2, \ldots, i_n)$ where $supp(i_a) \leq supp(i_b) \Leftrightarrow a < b$. This ordering helps to prune the prefix tree at lower depths and provides shorter run times. The benefits of this ordering are discussed by Goethals [39].

Details of the algorithms are explained in the following section.

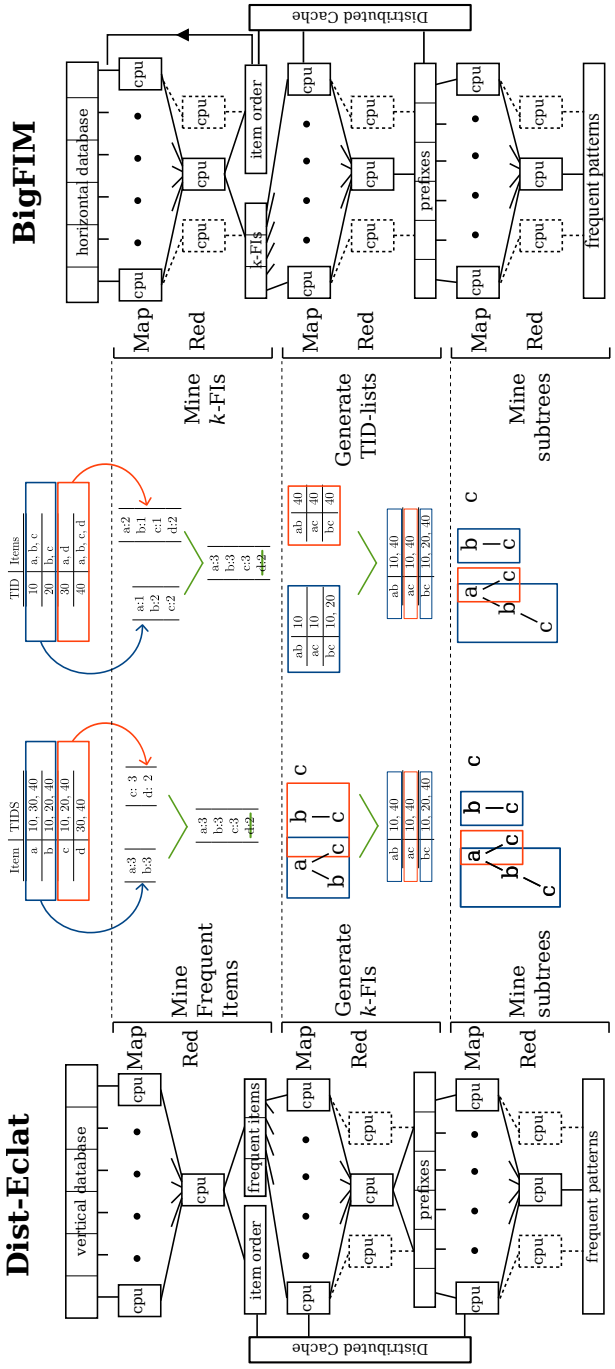



Figure 6.1: Eclat and BigFIM on MapReduce framework

## 6.5 Frequent Itemset Mining on MapReduce

We propose two new methods for mining frequent itemsets in parallel on the MapReduce framework that can handle low frequency thresholds. Our first method, called Dist-Eclat, is a pure Eclat method that distributes the search space as evenly as possible among mappers. This technique is able to mine large datasets, but can be prohibitive when dealing with massive amounts of data. Therefore, we introduce a second, hybrid method that first uses an Apriori based method to extract frequent itemsets of length $k$ and later on switches to Eclat when the projected databases fit in memory. We call this algorithm BigFIM.

### 6.5.1 Dist-Eclat

Our first method is a distributed version of Eclat that partitions the search space more evenly among different processing units. Current techniques often distribute the workload by partitioning the transaction database into equally sized sub databases, also called shards, e.g., the Partition algorithm [92]. Each of the sub databases can be mined separately and the results can then be combined. However, all local frequent itemsets should be combined and counted again to prune the globally infrequent ones, which is expensive.

First of all, this approach comes with a large communication cost, i.e., the number of sets to be mined can be very large, moreover, the number of sets that have to be recounted can also be very large. Implementing such a partitioning technique in Hadoop is therefore prohibitive. A possible solution for the recounting part, is to mine the sub databases with a lower threshold, hence, decreasing the number of itemsets that might have been missed. However, another problem then occurs: indeed, each shard defines a local sub database for which the local structure can be very different from the rest of the data. As a result, computation of the frequent itemsets can blow up tremendously for some shards, although many of the sets are actually local structures and far from interesting globally.

We argue that our method does not have to deal with such problems, since we are dividing the search space rather than the data space. Therefore, no extra communication between mappers is necessary and no checking of overlapping mining results has to be accounted for. We also claim that Eclat, more specifically Eclat using diffsets [111], memory-wise is the best fit for mining large datasets.

First of all, Eclat uses the depth-first approach, such that only a limited number of candidates have to be kept in memory even while finding long frequent patterns. In contrast, Apriori, the breadth-first approach, has to keep all $k$-sized frequent sets in memory when computing $k$+1-sized candidates. Secondly, using diffsets limits the memory overhead approximately to the size of the original tid-list root of the current branch [39]. This is easy to see: a diffset represents the tids that have to be subtracted from the tid-list of the parent to obtain the tid-list for this node, so, at most the tids that can be subtracted on a complete branch extension is the size of the original tid-list.

Dist-Eclat operates in a three step approach as shown on the left side of Figure 6.1. Each of the steps can be distributed among multiple mappers to maximally benefit from the cluster environment. For this implementation we do not start with a typical transaction database, rather we utilize immediately the vertical database format.

1) **Finding the Frequent Items:** During the first step, the vertical database is divided into equally sized blocks (shards) and distributed to available mappers. Each mapper extracts the frequent singletons from its shard. In the reduce phase, all frequent items are gathered without further processing.

2) **$k$-FIs Generation:** In this second step, $\mathscr{P}_k$, the set of frequent itemsets of size $k$, is generated. First, frequent singletons are distributed across $m$ mappers. Each of the mappers finds the frequent $k$-sized supersets of the items by running Eclat to level $k$. Finally, a reducer assigns $\mathscr{P}_k$ to a new batch of $m$ mappers. Distribution is done using Round-Robin.

3) **Subtree Mining:** The last step consists of mining the prefix tree starting at a prefix from the assigned batch using Eclat. Each mapper can complete this step independently since sub-trees do not require mutual information.

### 6.5.2 BigFIM

Our second method overcomes two problems inherent to Dist-Eclat. First, mining for $k$-FIs can already be infeasible. Indeed, in the worst case, one mapper needs the complete dataset to construct all 2-FIs pairs. Considering BigData, the tid-list of even a single item may not fit into memory. Secondly, most of the mappers require the whole dataset in memory in order to mine the sub-trees (cf. Section 6.6.1). Therefore the complete dataset has to be communicated to different mappers, which can be prohibitive for the given network infrastructure. Flow of BigFIM is shown on the right side of Figure 6.1.

1) **Generating $k$-FIs:** BigFIM covers the problem of large tid-lists by generating $k$-FIs using the breadth-first method. This can be achieved by adapting the Word Counting problem for documents [28], i.e., each mapper receives part of the database (a document) and reports the items/itemsets (the words) for which we want to know the support (the count). A reducer combines all local frequencies and reports only the globally frequent items/itemsets. These frequent itemsets can be redistributed to all mappers to act as candidates for the next step of breadth-first search. These steps can be repeated $k$ times, to obtain the set of $k$-FIs.

Computing the $k$-FIs in a level-wise fashion is the most logical choice: we do not have to keep large tid-lists in memory, rather we need only the itemsets that have to be counted. In Apriori, for the first few levels, keeping the candidates in memory is still possible—as opposed to continuing this process to greater depths. Alternatively, when a set of candidates does not fit into memory, partitioning the set of candidates across mappers can resolve the problem.

2) **Finding Potential Extensions:** After computing the prefixes, the next step is computing the possible extensions, i.e., obtaining tid-lists for $(k+1)$-FIs. This can be done similar to Word Counting, however, now instead of local support counts we report the local tid-lists. A reducer combines the local tid-lists from all mappers to a single global tid-list and assigns complete prefix groups to different mappers.

3) **Subtree Mining:** Finally, the mappers work on individual prefix groups. A prefix group defines a conditional database that completely fits into memory. The mining part then utilizes diffsets to mine the conditional database for frequent

itemsets using depth-first search.

Using 3-FIs as prefixes generally yields well-balanced distributions (cf. Section 6.6.1). Unfortunately, when dealing with large datasets, a set of 3-FIs extensions can still be too large to fit into memory. In such cases we can continue the iterative process until we reach a set of $k$-FIs that are small enough. To make an estimate on the size of the prefix extensions, the following heuristic can be used. Given a prefix $p$ with support $s$ and order $r$ out of $n$ items, the size of the conditional database described by $p$ is at most $s*(n-(r+1))$. Recall that when a set of candidates does not fit in to memory, the set of candidates can also be distributed and then the diffsets does not incur an exponential blow up in memory usage.

### 6.5.3 Implementation details

In our methods frequent itemsets are mined in step 3 by the mappers and then communicated to the reducer. To reduce network traffic, we encoded the mined itemsets using a compressed trie string representation for each batch of patterns, similar to the representation introduced by Zaki [108]. Basically, delimiters indicate if the tree is traversed downwards or upwards, and if a support is specified. As an example:

| | | |
|---|---|---|
| a (300)<br>a b c (100)<br>a b d (200)<br>b d (50) | → | a(300)\|b\|c(100)$d(200)$$$b\|d(50) |

As a second remark, we point out that our algorithm computes a superset of the closed itemsets found in a transaction dataset. We can easily do so by letting the individual mappers report only the closed sets in their subtree. Although it is perfectly possible to mine the correct set of closed itemsets, we omitted this post-processing step in our method.

## 6.6 Experiments

### 6.6.1 Load Balancing

We examine the load balancing in two ways: the relation between the workload and (1) the length of the distributed prefixes, (2) the assignment scheme.

Let the set of $k$-prefixes $\mathscr{P}_k = \{p_1, p_2, \ldots, p_m\}$ be partitioned to $n$ workers, $\mathscr{P}_k^1$, $\mathscr{P}_k^2, \ldots, \mathscr{P}_k^n$, where $\mathscr{P}_k^j$ is the set of prefixes assigned to worker $j$, then the prefixes are assigned to worker nodes using the following methods:

- **Round-Robin:** $p_i$ is assigned to the worker $\mathscr{P}_k^{(i \bmod n)}$.
- **Equal Weight:** When $p_i$ is assigned to a worker, $supp(p_i)$ is added the score of that worker. $p_{i+1}$ is assigned to a worker with the lowest score. Assignment is order dependent.

| Dataset | *minsup* | Total | 1-FIs | 2-FIs | 3-FIs |
|---|---|---|---|---|---|
| Abstracts | 5 | 388,631 | 1,393 | 37,363 | 171,553 |
| T10I4D100K | 100 | 27,169 | 797 | 9,627 | 16,742 |
| Mushroom | 812 | 11,242 | 56 | 664 | 2,816 |
| Pumsb | 24,523 | 22,402,411 | 52 | 968 | 9,416 |

Table 6.1: Number of total and 1, 2 and 3 length frequent itemsets for datasets.

- **Block partitioning:** Assignments are done by intervals, i.e., $\{p_1, \ldots, p_{\lceil m/n \rceil}\}$ are assigned to $\mathscr{P}_k^1$, $\{p_{(\lceil m/n \rceil+1)}, \ldots, p_{(2\times\lceil m/n \rceil)}\}$ are assigned to $\mathscr{P}_k^2$, and so on.
- **Random:** Each $p_i$ is assigned to a random worker.

For this set of experiments we use 4 different datasets: Abstracts provided by De Bie [27], T10I4D100K, Mushroom and Pumsb are from FIMI repository [1]. Properties of these datasets are given in Table 6.2.

We first mine the datasets to find prefixes of length 1, 2, and 3, then we distribute these prefixes to 128 workers using the assignment methods above. The number of FIs that are generated by each worker is used to estimate the amount of work done by that worker. Because of the high pruning ratio at lower depths, this is an accurate estimate of total work of a worker. Note that, since the number of FIs that are mined in the first step increases with the prefix length, the amount of distributed load decreases.

Table 6.4 shows the standard deviation (StDev), average (Avg) and the difference between the maximum and the minimum (Max-Min) of the workloads. Since we are trying to decrease the total running time by balancing the workloads, the most important indicator of the balancing is the Max-Min.

Figure 6.2 shows the number of frequent itemsets generated by 8 workers when the prefixes of length 1, 2, 3 are assigned using Round-Robin.

Our experiments suggest independence of the assignment method: using longer prefixes results in a better balancing of the load.

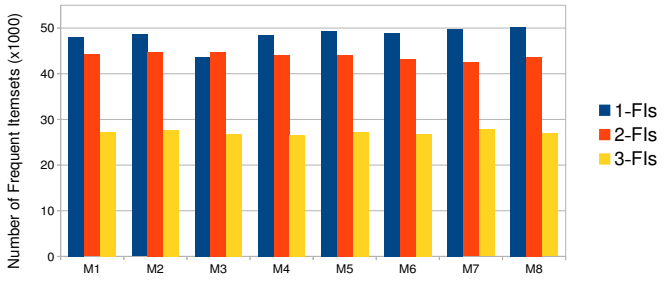
Generating the longer prefixes requires additional initial computation. However, for large databases this computation is negligible compared to the entire mining process. Table 6.1 shows the 1, 2 and 3 length frequent itemsets along with the total number of frequent itemsets in different databases. For example in Abstracts, a small dataset, finding 3-FIs is half of the total work, on the other hand; in Pumsb, a larger dataset, the number of 3-FIs is negligible to the total number of FIs, thus, it is still viable to distribute the remaining work after finding long prefixes.

| Dataset | Number of Items | Number of Transactions |
|---|---|---|
| Abstracts | 4,976 | 859 |
| T10I4D100K | 870 | 100,000 |
| Mushroom | 119 | 8,124 |
| Pumsb | 2,113 | 49,046 |
| Tag | 45,446,863 | 6,201,207 |

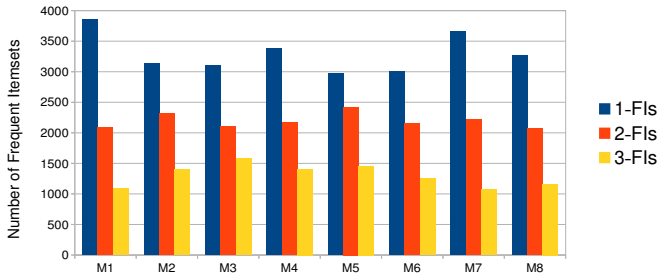
Table 6.2: Properties of datasets for our experiments

| | FIs | Avg | Max | Min | StDev |
|---|---|---|---|---|---|
| Abstracts | 1 | 0.29 | 0.66 | 0.20 | 0.09 |
| | 2 | 0.91 | 0.93 | 0.86 | 0.01 |
| | 3 | 0.95 | 0.96 | 0.93 | 0.01 |
| Pumsb | 1 | 0.78 | 0.99 | 0.51 | 0.17 |
| | 2 | 0.99 | 1.00 | 0.99 | 0.00 |
| | 3 | 1.00 | 1.00 | 1.00 | 0.00 |
| Mushroom | 1 | 0.38 | 1.00 | 0.10 | 0.23 |
| | 2 | 0.82 | 0.99 | 0.57 | 0.09 |
| | 3 | 0.99 | 1.00 | 0.97 | 0.01 |

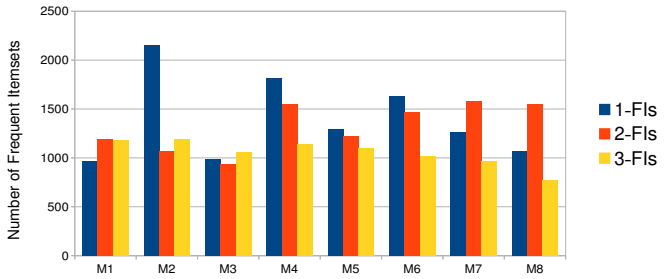
Table 6.3: Data ratio needed for different computation units



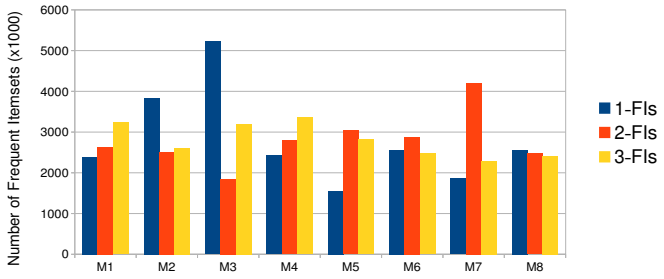

(a) Abstracts




(b) T10I4D100K




(c) Mushroom




(d) Pumsb

Figure 6.2: Number of Frequent Itemsets generated by partitions

| | | Round-Robin | | | Equal Weight | | | Block | | | Random | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\mathscr{P}_1$ | $\mathscr{P}_2$ | $\mathscr{P}_3$ | $\mathscr{P}_1$ | $\mathscr{P}_2$ | $\mathscr{P}_3$ | $\mathscr{P}_1$ | $\mathscr{P}_2$ | $\mathscr{P}_3$ | $\mathscr{P}_1$ | $\mathscr{P}_2$ | $\mathscr{P}_3$ |
| Abstracts | StDev | 1071 | 311 | 107 | 1071 | 371 | 118 | 7235 | 4850 | 1835 | 2260 | 432 | 150 |
| | Average | 3025 | 2744 | 1696 | 3025 | 2744 | 1696 | 3025 | 2744 | 1696 | 3025 | 2744 | 1696 |
| | Max-Min | 4488 | 1695 | 482 | 4488 | 2223 | 675 | 35314 | 29606 | 9840 | 11540 | 2756 | 686 |
| T10I4D100K | StDev | 103 | 57 | 34 | 103 | 64 | 32 | 138 | 137 | 68 | 130 | 62 | 32 |
| | Average | 213 | 142 | 85 | 213 | 142 | 85 | 215 | 142 | 87 | 213 | 140 | 85 |
| | Max-Min | 666 | 384 | 183 | 666 | 499 | 179 | 723 | 699 | 471 | 1002 | 333 | 239 |
| Mushroom | StDev | 13287 | 6449 | 4096 | 13287 | 6052 | 3596 | 13287 | 10026 | 7518 | 13287 | 5736 | 3679 |
| | Average | 4488 | 4482 | 4446 | 4488 | 4482 | 4446 | 4488 | 4482 | 4446 | 4488 | 4482 | 4446 |
| | Max-Min | 98303 | 36348 | 23814 | 98303 | 33453 | 24711 | 98303 | 69626 | 40406 | 98303 | 30991 | 24830 |
| Pumsb | StDev | 3897683 | 1955503 | 1112237 | 3897683 | 2077845 | 1003724 | 3897683 | 3442159 | 2598825 | 3897683 | 2065441 | 1191945 |
| | Average | 1296121 | 1296113 | 1296037 | 1296121 | 1296113 | 1296037 | 1296121 | 1296113 | 1296037 | 1296121 | 1296113 | 1296037 |
| | Max-Min | 21342943 | 9809612 | 6167970 | 21342943 | 10089432 | 4815391 | 21342943 | 20059553 | 14507567 | 21342943 | 10534931 | 5827088 |

Table 6.4: Prefix assignments with different methods and prefix lengths.

The allocation schemes that have been used in our experiments (c.f. Section 6.6.1) are able to distribute the search space well among different computation units w.r.t. the workload balancing. However, we found that such schemes rendered almost all nodes to be depending on the complete dataset. Hence, the complete data should also be communicated to all nodes. Table 6.3 shows statistics on the percentage of the data that is required by a node when using Equal Weight assignment. The table shows that for 3-FIs, the average ratio is almost 1, meaning that almost all nodes need the full dataset. This property can be infeasible when dealing with BigData.

### 6.6.2 Execution Time

For our runtime experiments we used a scrape of the delicious dataset provided by DAI-Labor [107]. The original dataset contains tagging content between 2004 and 2007. The content is identified by a timestamp, a user, a url and a given tagname. We created a transaction dataset where transactions represent tags. Properties of the dataset is shown in Table 6.2. For the first few experiments we used our local cluster consisting of 2 machines. Each machine containing 32 Intel(R) Xeon(R) processing units and 32GB of RAM. However, we restricted each machine to use up to 6 mappers each. Both machines are running on Ubuntu 12.04 and Hadoop 1.1.2.

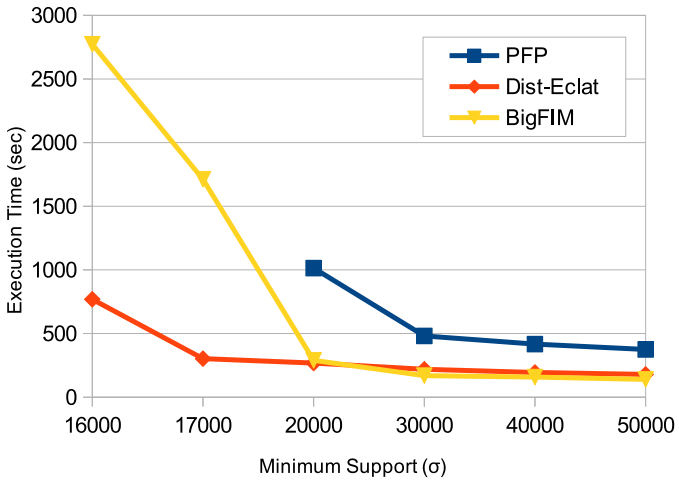



Figure 6.3: Timing comparison for different methods on Tag

We compared running times for our Hadoop implementations of Dist-Eclat and BigFIM and the PFP implementation provided by the Mahout library [3]. Note that PFP is designed as a top-k miner. Therefore, we forced the algorithm to output the same number of patterns found by our implementations to obtain a more fair comparison. The result is shown in Figure 6.3, the x-axis gives the *minsup* threshold for different runs and the y-axis shows the computation time in seconds. As expected,

Dist-Eclat is the fastest method and BigFIM is second. However, for lower *minsup* values it is an order of magnitude slower. Remember, however, that the goal of BigFIM is to be able to mine enormous databases. More importantly, PFP is always much slower than our methods and while decreasing the minimum support, we observed that PFP either takes an infeasible amount of time (we stopped execution after approximately one week) or runs out of memory.

The rest of our experiments are conducted without generating the frequent sets as output. We focus on mining speed rather than on shipping of data.

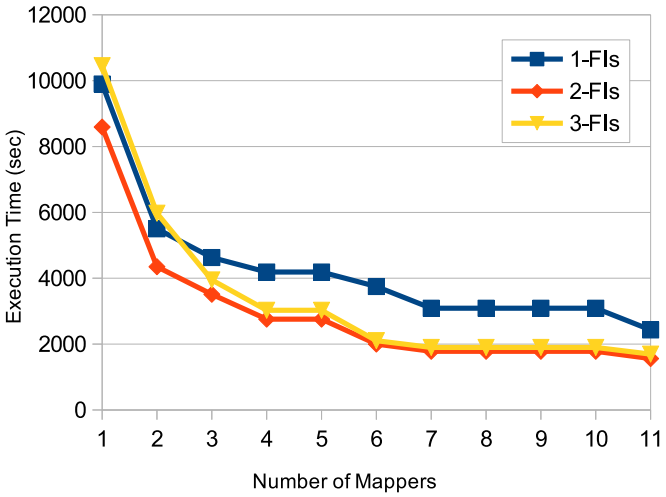



Figure 6.4: Total execution time on Tag with $\sigma = 15.5K$

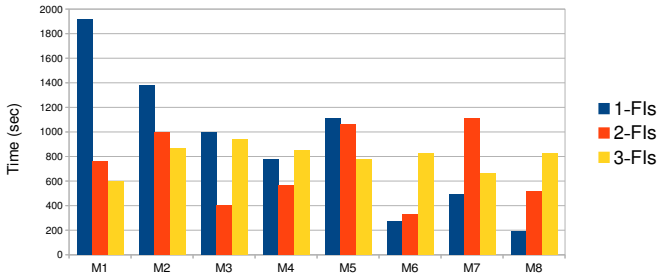



Figure 6.5: Execution time of mappers for Tag with $\sigma = 15.5K$

We analysed the speed of our algorithm using seeds of length 1, 2 and 3. Figure 6.4 shows timing results when mining the Tag dataset with a *minsup* of 15.5K. The x-axis

shows the number of mappers used for each run and the y-axis shows the time in seconds. The figure shows that using 2- and 3-FIs give lowest running times and that both of them are in fact similar for this dataset in terms of complete running time. Figure 6.5 gives an in depth view on the time distribution for individual mappers for Tag. The x-axis shows 8 mappers with unique IDs and the y-axis shows the computation time in seconds. We see a clear inbalance for the 1-FIs: mapper M8 finishes 10 times faster than M1. 2-FIs is much better due to a better load balancing, however, the variance is still large. Using 3-FIs yields equally divided mining times.

| #Mappers | Avg | Max | Min | StDev | Avg FIs |
|---|---|---|---|---|---|
| 1 | 36,546.00 | 36,546 | 36,546 | 0.00 | 18,885 |
| 5 | 7,775.00 | 8,147 | 7,042 | 400.17 | 3,777 |
| 10 | 3,810.90 | 4,616 | 2,591 | 710.14 | 1,889 |
| 15 | 2,516.47 | 3,726 | 1,733 | 616.26 | 1,259 |
| 20 | 1,912.85 | 3,533 | 875 | 754.05 | 944 |
| 40 | 943.63 | 2,016 | 278 | 456.56 | 472 |
| 60 | 629.80 | 1,579 | 146 | 344.30 | 315 |
| 80 | 472.35 | 1,422 | 69 | 326.37 | 236 |
| 100 | 376.45 | 2,177 | 60 | 313.57 | 189 |
| 120 | 316.15 | 1,542 | 29 | 264.72 | 157 |

Table 6.5: Mining time stats Pumsb with 3-FIs and $\sigma = 12K$

| #Mappers | Avg | Max | Min | StDev | Avg FIs |
|---|---|---|---|---|---|
| 20 | 25,851 | 42,900 | 11,160 | 8,724.88 | 844 |
| 40 | 13,596 | 28,620 | 4,320 | 6,439.29 | 422 |
| 60 | 8,508 | 24,240 | 1,440 | 5,195.64 | 281 |
| 80 | 6,548 | 23,760 | 1,320 | 4,066.11 | 211 |

Table 6.6: Mining time stats Tag with 3-FIs and $\sigma = 15K$

In another experiment we tested the Pumsb dataset with a minimum support of 12K. We used only 3-FIs as seeds, but varied the number of mappers between 1 and 120. Statistics for this experiment are shown in Table 6.5. We see that not only the average runtime per node decreases drastically, also the maximum mining time decreases almost linearly. The latter, in the end, influences the running time the most. One peculiarity in the result is the sudden increase in time when using 100 mappers. This behaviour is due to coincidence; indeed, some mappers may get many large seeds because of our simple allocation scheme, resulting in an increased runtime. Figure 6.6 shows the decrease in running time compared to the average number of prefixes per node. The red line in fact shows the perfect scalability behaviour. Its shows that the scalability is finite in the sense that after a while adding only a few nodes is not productive. Because the number of seeds to be divided

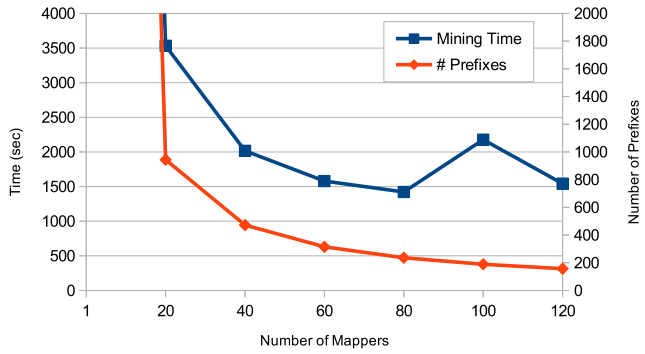



Figure 6.6: Timing results Pumsb, $\sigma = 12K$

reaches a saturation point. We also have to remark that during initialisation of the mappers, the data has to be distributed to all of them, resulting in a larger total computation time, in contrast to using less nodes.

At last, we also conducted the previous experiment for Tag on a real cluster, using Amazon Elastic MapReduce[2]. We used clusters of sizes between 20 and 80 mappers, using the *m1.xlarge* high memory and high I/O performance instances. Each instance consists of multiple cores performing one map task. The instances are running Red Hat Enterprise Linux and the Amazon Distribution of MapReduce. Table 6.6 shows the statistics for this experiment. We see that the scaling up is less drastic compared to Pumsb. One reason is the blow up of the subtree for some 3-FIs which can not be distributed any longer. The only solution is to increase the length of the prefixes.

## 6.7 Conclusion

In this chapter we studied and implemented two frequent itemset mining algorithms for MapReduce. Dist-Eclat focuses on speed by using a simple load balancing scheme based on $k$-FIs. A second algorithm, BigFIM, focuses on mining very large databases by utilizing a hybrid approach. $k$-FIs are mined by an Apriori variant and then the found frequent itemsets are distributed to the mappers. At the mappers frequent itemsets are mined using Eclat.

We studied several techniques for balancing the load of the $k$-FIs. Our results show that using 3-FIs in combination with even a basic Round-Robin allocation scheme results in a good workload distribution. Progressing further down the prefix tree results in even better workload distributions, however the number of intermediate frequent itemsets blow up.

Assignment methods that will result in a better workload distribution is a further research issue.

[2] http://aws.amazon.com/elasticmapreduce

At last, the experiments show that our methods outperform state-of-the-art FIM methods on BigData using MapReduce.

CHAPTER 7

# Conclusions

An important leap on data utilisation changed the world forever. Technical advancements and paradigm changes that led to a critical break point of how we use data, can be referred by the umbrella term *BigData*. The world we are living in is now much more data-oriented and considering its benefits and how it changes our lives, importance of data will not diminish soon.

## 7.1 Main Contributions

In this thesis, we investigate some of the challenges of the increasing the amount of data to analyse. We propose new algorithms and provide software tools to better understand the data in a more efficient and effective manner.

In Chapter 2, we proposed *cartification*, a novel method that transforms high dimensional relational databases into a transaction database. Similarity measures and the traditional clustering methods that rely on them are disturbed in high-dimensional databases by the effects of so-called *curse of dimensionality*. Cartification transformation preserves the local neighborhoods and similarity information. Therefore, the local cluster structures in the data are preserved and they form repetitive patterns in the transformed database.

Mining repetitive and frequent patterns in transaction databases is well-studied in the literature. Our transformation allows the application of itemset mining techniques to mine patterns in relational databases. We showed the applicability of this approach by proposing the CARTICLUS method, which finds subspace clusters in high-dimensional relational databases using frequent itemset mining methods on the transformed databases.

In Chapter 3, we investigated the special cases for cartification. In particular, we studied the properties of neighborhoods in univariate projections. We showed that when a total ordering of the data objects is possible, the neighborhood databases have some intrinsic properties. We proposed CLON, a method that exploits these properties to efficiently perform cartification transformation and find all of the clusters in the database. With exhaustive experiments, we showed that CLON is

orders of magnitude faster than the state-of-the-art methods including CARTICLUS, while the quality of the detected clusters are better or comparable. Experiments on a movie rating database showed that our method is a good fit for exploratory data analysis too.

Both the CARTICLUS and CLON use binary transaction databases. In Chapter 4, we experimented with a non-binary cartification transformation. We convert high-dimensional relational databases into multiple rank matrices, preserving the relative neighborhood information for the whole database. Since objects in a cluster often co-occur in each other's neighborhoods, *minimal tiles* in the ranked matrices correspond to clusters in the data. Exploiting the relative similarity of the objects in the neighborhoods provides robustness on parameter selection. Moreover, the rank matrices of univariate projections have some intrinsic properties. Using these properties, we developed CARTIRANK, an efficient algorithm that finds minimal tiles and, hence, the clusters. We experimentally showed that the cluster finding capabilities of CARTIRANK is less dependent on the input parameters, and therefore it can effectively find clusters of different sizes. The high dimensional search strategy of CARTIRANK produces less redundant clusters than CARTICLUS and CLON.

We gather up the ideas about the cartification and implemented VINeM, a software tool for Visual Interactive Neighborhood Mining. VINeM can effectively visualize the neighborhoods, so that the cluster structures become visible. A user can *play* with the data by selecting interesting clusters, filtering out uninteresting objects, introducing new similarity measures, or applying different projections. Unsupervised methods are also implemented in the application and the user can seamlessly switch between interactive analysis and unsupervised mining.

There are soft time constraints in interactive application settings. Although CLON can improve the efficiency for some particular datasets, on other datasets we still have to employ frequent itemset mining methods. However, FIM methods are computationally expensive and may overburden the client, failing to satisfy time constraints. Fortunately, computationally expensive mining can be performed remotely and the resources of the client can be allocated for interactive analysis. In chapter 6, we investigate the applicability of frequent itemset mining methods on sate-of-the-art technologies. We focus on Hadoop as it is one of the most widely-used platforms for distributed computing. We studied the limitations of the platform along with the requirements of frequent itemset mining algorithms. We found out that the overall performance depends on the balanced distribution of the work load over the computation nodes, and proposed a heuristic to partition the prefix tree. Our implementation performs much better that the publicly available implementations, and it is itself publicly available under a free software license.

## 7.2 Outlook

Local neighborhoods hold essential similarity information. In this thesis, we studied the local neighborhoods and possible applications that exploit their properties in the domain of pattern mining. However, we barely scratched the surface of what is possible by exploiting the local neighborhoods. Before looking forward, let us look on what we have already improved.

Three chapters of the thesis deal with the theoretical properties of local neigh-

| | CARTICLUS | CLON | CARTIRANK |
|---|---|---|---|
| Memory Requirement | $O(n \times d \times \|C\|)$ | $O(n \times d)$ | $O(n^2 \times d)$ |
| Advantage | Can work with any kind of data | Fast | Improved accuracy |
| Advantage | Noise aware | Noise aware | Less redundant output |
| Limitation | Slow | Can only work with univariate data | Slow |
| Limitation | Redundant clusters | Redundant clusters | High memory requirement |

Table 7.1: A short summary of the advantages and limitations of the methods. $n$ is the number of objects, $|C|$ is the cluster size, and $d$ is the number of attributes.

borhoods, or in other words, the *cartification* transformation: Chapter 2 defines the general framework, Chapter 3 improves the run time performance for continuous-value databases and Chapter 4 improves the accuracy by allocating more resources. Both of the latter chapters investigate and improve the original method for some particular use cases. Core features of these three methods are summarized in Table 7.1.

One of the advantages of cartification is its usage of $k$-nearest neighborhoods. When we disregard the actual distances and work with the relative similarities, we avoid the pitfalls of different scales in the database. On the other hand, $k$-nearest neighborhoods impose fixed-size neighborhoods, which have its own advantages and disadvantages. On the plus side, $k$ parameter operates as a zooming factor, i.e., the user can select between large or small clusters by configuring the $k$ parameter. This can be desired for the exploratory data analysis scenarios. On the negative side, fixed-size neighborhoods can hinder the detection of clusters of different sizes.

CARTIRANK tackles the problem of fixed size neighborhoods by using rank matrices instead of binary neighborhoods. However, not only storage requirements for rank matrices are high, but also the tile mining is computationally more expensive than frequent itemset mining. More balanced approaches that will relax the memory requirements should be possible and considering the quality improvement of CARTIRANK, worth researching.

Recurrency-based mining of our techniques are effective both at finding clusters and discarding noise. However, computing the recurrencies is computationally expensive and it is not efficient enough to depend on frequent itemset mining methods. CARTICLUS uses a sampling-based frequent itemset mining algorithm to improve efficiency and can still produce very good results since the FIM methods are known to produce redundant results. CLON, on the other hand, focuses on continuous-value databases and exploit their intrinsic properties to speed-up the computation. This shows us that studying the neighborhood databases can yield to discoveries of some other intrinsic properties that can be exploited to invent more

efficient algorithms.

The cartification methods in this thesis focus on univariate projections of the data. In Chapter 3 and Chapter 4, we showed that this focus improves the efficiency of the algorithms. However, in some databases clusters are not separable on one-dimensional projections. Combined with the limitations of fixed-size neighborhoods, mining only one-dimensional projections hinder the cluster detection on some real world datasets. Of course, neighborhoods can still be computed and mined for redundancy in multi-dimensional projections, but finding and selecting the projection dimensions is not trivial and, as a matter of fact, it is a well-known research problem.

Small frequent itemsets in a cartified database represent the micro-clusters in the original database. By comparing micro-clusters of different projections, one can discover the relations between these projections. Since the micro-clusters are localized, they can be more effective than feature selection techniques on high dimensional data. We already provide a tool in VINeM that uses them for finding relevant dimensions, cf. Chapter 5. Further research on more sophisticated techniques can yield much more interesting results.

While the transformation emphasizes some information in the database, such as localities, it disregards some other information, such as the actual distances. By combining with recurrency-based mining methods, we can extract valuable knowledge from the transformed data. In this spirit, it will be interesting to investigate the amount and the impact of the information loss, or gain in some cases, caused by the transformation. Can it be measured theoretically or empirically? How should we evaluate non-linear transformations in terms of information gain? For example, non-binary transformation of neighborhoods, such as the one we have done in Chapter 4, improves the robustness of the methods.

We studied one example of non-binary neighborhood transformation, and shows that new approaches can address some issues. Rank matrices for neighborhoods, and other approaches for neighborhoods transformation, should be studied in more detail to find out whether they can address some other issues. As cartification put the frequent itemset mining algorithms at the disposal of relational databases, non-binary transformation can be a bridge to the rich literature on mining ranked matrices.

Pixel-based representation of our GUI tool VINeM provides a novel and intuitive way to get a better understanding of the data. However, pixels are limited with the screen resolutions and therefore not suitable for large databases. Considering the convenience of visual data mining, we should investigate more to discover new techniques that allows the representation and interactive analysis of larger databases.

Main motivations behind implementing the frequent itemset mining algorithms for a popular framework were to improve their accessibility and to evaluate their adaptivity to new architectures. Since we released the code for the Hadoop framework, there have been many changes in the field: Mahout dropped support for MapReduce and Apache Spark [4] framework became a strong alternative for pure MapReduce implementations. Applicability of our algorithms to the new platforms should be investigated.

# Bibliography


[1] Frequent itemset mining dataset repository. http://fimi.ua.ac.be/data, 2004.

[2] Apache hadoop. http://hadoop.apache.org/, 2015.

[3] Apache mahout. http://mahout.apache.org/, 2015.

[4] Apache spark. http://spark.apache.org/, 2015.

[5] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. Fast algorithms for projected clustering. *SIGMOD Rec.*, 28(2):61–72, June 1999.

[6] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Rec.*, 27(2):94–105, June 1998.

[7] R. Agrawal, T. Imielinksi, and A. Swami. Mining association rules between sets of items in large databases. pages 207–216, 1993.

[8] R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE Trans. Knowl. Data Eng.*, pages 962–969, 1996.

[9] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994.

[10] E. Aksehirli, B. Goethals, E. Müller, and J. Vreeken. Cartification: A neighborhood preserving transformation for mining high dimensional data. In *2013 IEEE 13th International Conference on Data Mining (ICDM)*, pages 937–942, Dec. 2013.

[11] E. Aksehirli, B. Goethals, and E. MÃijller. Efficient cluster detection by ordered neighborhoods. In S. Madria and T. Hara, editors, *Big Data Analytics and Knowledge Discovery*, volume 9263 of *Lecture Notes in Computer Science*, pages 15–27. Springer International Publishing, 2015.

[12] E. Aksehirli, S. Nijssen, M. van Leeuwen, and B. Goethals. Finding subspace clusters using ranked neighborhoods. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on (to appear)*. IEEE, 2015.

[13] U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, June 1999.

[14] G. A. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 2000.

[15] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *SODA*, pages 1027–1035, 2007.

[16] I. Assent, R. Krieger, E. Müller, and T. Seidl. INSCY: Indexing subspace clusters with in-process-removal of redundancy. In *Eighth IEEE International Conference on Data Mining, 2008. ICDM '08*, pages 719–724, Dec. 2008.

[17] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, 6(6):937–965, 2005.

[18] R. J. Bayardo, Jr. Efficiently mining long patterns from databases. *SIGMOD Rec.*, 27(2):85–93, June 1998.

[19] K. P. Bennett, U. Fayyad, and D. Geiger. Density-based indexing for approximate nearest-neighbor queries. In *KDD*, pages 233–243, 1999.

[20] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Database Theory–ICDT'99*, pages 217–235. Springer, 1999.

[21] M. Boley and H. Grosskreutz. Approximating the number of frequent sets in dense data. *Knowl. Inf. Syst.*, pages 65–89, 2009.

[22] M. Boley, M. Mampaey, B. Kang, P. Tokmakov, and S. Wrobel. One click mining: Interactive local pattern discovery through implicit preference and performance learning. In *Proceedings of the ACM SIGKDD*, IDEA '13, pages 27–35, New York, NY, USA, 2013. ACM.

[23] S. Bremm, T. von Landesberger, M. He\s s, T. Schreck, P. Weil, and K. Hamacherk. Interactive visual comparison of multiple trees. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pages 31–40. IEEE, 2011.

[24] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *ACM Sigmod Record*, volume 29, pages 93–104. ACM, 2000.

[25] C.-H. Cheng, A. W. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *KDD*, pages 84–93, 1999.

[26] T. F. Cox and M. A. Cox. *Multidimensional scaling*. CRC Press, 2010.

[27] T. De Bie. An information theoretic framework for data mining. In *Proc. ACM SIGKDD*, pages 564–572, 2011.

[28] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. OSDI*. USENIX Association, 2004.

[29] G. Dong and J. Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 43–52, New York, NY, USA, 1999. ACM.

[30] J. G. Dy and C. E. Brodley. Feature selection for unsupervised learning. 5(8):909–921, 2004.

[31] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: A runtime for iterative MapReduce. In *Proc. HPDC*, pages 810–818. ACM, 2010.

[32] T. Emrich, H.-P. Kriegel, N. Mamoulis, J. Niedermayer, M. Renz, and A. ZÃijfle. Reverse-nearest neighbor queries on uncertain moving object trajectories. volume 8422, pages 92–107. 2014.

[33] L. Ertöz, M. Steinbach, and V. Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. In *Clust. Inf. Ret.*, pages 83–104. 2003.

[34] X. Z. Fern and C. E. Brodley. Solving cluster ensemble problems by bipartite graph partitioning. In *ICML '04*, pages 36–. ACM, 2004.

[35] K. Finley. 5 ways to tell which programming languages are most popular ReadWrite. http://readwrite.com/2012/06/05/5-ways-to-tell-which-programming-lanugages-are-most-popular, 2012.

[36] A. Fred and A. Jain. Combining multiple clusterings using evidence accumulation. 27(6):835–850, 2005.

[37] F. Geerts, B. Goethals, and J. V. D. Bussche. Tight upper bounds on the number of candidate patterns. *ACM Trans. Database Syst.*, pages 333–363, 2005.

[38] A. Ghoting, P. Kambadur, E. Pednault, and R. Kannan. NIMBLE: a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce. In *Proc. ACM SIGKDD*, pages 334–342. ACM, 2011.

[39] B. Goethals. Survey on frequent pattern mining. *Univ. of Helsinki*, 2003.

[40] B. Goethals, S. Moens, and J. Vreeken. MIME: a framework for interactive visual pattern mining. In *Proceedings of the 17th ACM SIGKDD*, pages 757–760. ACM, 2011.

[41] M. Goldstein. $k_n$-nearest neighbor classification. *Information Theory, IEEE Transactions on*, 18(5):627–630, 1972.

[42] S. Günnemann, I. Färber, E. Müller, I. Assent, and T. Seidl. External evaluation measures for subspace clustering. pages 1363–1372. ACM, 2011.

[43] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[44] S. Hammoud. *MapReduce Network Enabled Algorithms for Classification Based on Association Rules*. Thesis, 2011.

[45] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, pages 1–12, 2000.

[46] T. Hey, S. Tansley, and K. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Wash., 1 edition edition, Oct. 2009.

[47] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *VLDB*, pages 506–515, 2000.

[48] M. E. Houle. Navigating massive data sets via local clustering. In *KDD*, pages 547–552, 2003.

[49] P. Hu, C. Vens, B. Verstrynge, and H. Blockeel. Generalizing from Example Clusters. In J. Fürnkranz, E. Hüllermeier, and T. Higuchi, editors, *Discovery Science*, number 8140 in Lecture Notes in Computer Science, pages 64–78. Springer Berlin Heidelberg, Jan. 2013.

[50] A. Inselberg and B. Dimsdale. Parallel coordinates. In *Human-Machine Interactive Systems*, pages 199–233. Springer, 1991.

[51] A. Jain, M. Murty, and P. Flynn. Data clustering: a review. 31(3):264–323, 1999.

[52] A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, June 2010.

[53] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[54] R. Jarvis and E. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, C-22(11):1025 – 1034, Nov. 1973.

[55] R. A. Jarvis and E. A. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Trans. Comput.*, 22(11):1025–1034, 1973.

[56] K. Kailing, H.-P. Kriegel, and P. Kröger. Density-connected subspace clustering for high-dimensional data. In *Proc. SDM*, volume 4. SIAM, 2004.

[57] K. Kailing, H.-P. Kriegel, P. Kröger, and S. Wanka. Ranking interesting subspaces for clustering high dimensional data. In *PKDD*, pages 241–252, 2003.

[58] D. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, Jan. 2002.

[59] F. Keller, E. Müller, and K. Böhm. HiCS: High contrast subspaces for density-based outlier ranking. In *ICDE*, 2012.

[60] H.-P. Kriegel, P. Kröger, M. Renz, and S. Wurst. A generic framework for efficient subspace clustering of high-dimensional data. In *Fifth IEEE International Conference on Data Mining*, pages 8 pp.–, Nov. 2005.

[61] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1âĂŞ1:58, Mar. 2009.

[62] H.-P. Kriegel, P. KrÃűger, and A. Zimek. Subspace clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(4):351âĂŞ364, 2012.

[63] J. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction*. Springer, New York, 2007.

[64] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang. Pfp: Parallel fp-growth for query recommendation. In *Proc. RecSys*, pages 107–114, 2008.

[65] J. Li, Y. Liu, W.-k. Liao, and A. Choudhary. Parallel data mining algorithms for association rules and clustering. In *Intl. Conf. on Management of Data*, 2008.

[66] N. Li, L. Zeng, Q. He, and Z. Shi. Parallel implementation of apriori algorithm based on MapReduce. In *Proc. SNPD*, pages 236–241, 2012.

[67] M.-Y. Lin, P.-Y. Lee, and S.-C. Hsueh. Apriori-based frequent itemset mining algorithms on MapReduce. In *Proc. ICUIMC*, pages 26–30. ACM, 2012.

[68] L. Maaten. Learning a parametric embedding by preserving local structure. In *International Conference on Artificial Intelligence and Statistics*, pages 384–391, 2009.

[69] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14, 1967.

[70] M. Malek and H. Kadima. Searching frequent itemsets by clustering data: Towards a parallel approach using mapreduce. In *Proc. WISE 2011 and 2012 Workshops*, pages 251–258. Springer Berlin Heidelberg, 2013.

[71] M. Mampaey, J. Vreeken, and N. Tatti. Summarizing data succinctly with the most informative itemsets. *TKDD*, 6(4):16, 2012.

[72] S. McCann and D. Lowe. Local naive bayes nearest neighbor for image classification. pages 3650–3656, June 2012.

[73] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In *Proc. WIDM*, pages 9–15. ACM, 2001.

[74] S. Moens, E. Aksehirli, and B. Goethals. Frequent itemset mining for big data. In *Big Data, 2013 IEEE International Conference on*, pages 111–118, Oct 2013.

[75] 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 SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 533–541, New York, NY, USA, 2008. ACM.

[76] G. Moise, J. Sander, and M. Ester. P3C: A robust projected clustering algorithm. In *ICDM*, pages 414–425, 2006.

[77] E. Müller, I. Assent, S. Günnemann, R. Krieger, and T. Seidl. Relevant subspace clustering: Mining the most interesting non-redundant concepts in high dimensional data. In *Ninth IEEE International Conference on Data Mining, 2009. ICDM '09*, pages 377–386, Dec. 2009.

[78] E. Müller, S. Günnemann, I. Assent, and T. Seidl. Evaluating clustering in subspace projections of high dimensional data. *Proceedings of the VLDB Endowment*, 2(1):1270–1281, 2009.

[79] H. V. Nguyen, E. Müller, J. Vreeken, F. Keller, and K. Böhm. CMI: An information-theoretic contrast measure for enhancing subspace cluster and outlier detection. In *SDM*, pages 198–206, 2013.

[80] C. L. Nutt, D. R. Mani, R. A. Betensky, P. Tamayo, J. G. Cairncross, C. Ladd, U. Pohl, C. Hartmann, M. E. McLaughlin, T. T. Batchelor, P. M. Black, A. v. Deimling, S. L. Pomeroy, T. R. Golub, and D. N. Louis. Gene expression-based classification of malignant gliomas correlates better with survival than histological classification. *Cancer Res*, 63(7):1602–1607, Apr. 2003.

[81] E. Ozkural, B. Ucar, and C. Aykanat. Parallel frequent item set mining with selective item replication. *IEEE Trans. Parallel Distrib. Syst.*, pages 1632–1640, 2011.

[82] B.-H. Park and H. Kargupta. Distributed data mining: Algorithms, systems, and applications. 2002.

[83] Y. Park, S. Park, W. Jung, and S. goo Lee. Reversed cf: A fast collaborative filtering algorithm using a k-nearest neighbor graph. 42(8):4022 – 4028, 2015.

[84] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explor. Newsl.*, 6(1):90–105, June 2004.

[85] S. Parthasarathy, M. J. Zaki, M. Ogihara, and W. Li. Parallel data mining for association rules on shared-memory systems. *Knowl. Inf. Syst.*, pages 1–29, 2001.

[86] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572, 1901.

[87] M. Riondato, J. A. DeBrabant, R. Fonseca, and E. Upfal. PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce. In *Proc. CIKM*, pages 85–94. ACM, 2012.

[88] A. Rodriguez and A. Laio. Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496, June 2014.

[89] V. Roth and T. Lange. Feature selection in clustering problems. In *NIPS*, pages 141–152, 2003.

[90] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, Dec. 2000. PMID: 11125150.

[91] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[92] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. VLDB*, pages 432–444, 1995.

[93] J. Schneider and M. Vlachos. Fast parameterless density-based clustering via random projections. In *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM '13, pages 861–866, New York, NY, USA, 2013. ACM.

[94] J. Seo and B. Shneiderman. Interactively exploring hierarchical clustering results [gene identification]. *Computer*, 35(7):80–86, July 2002.

[95] J. Seo and B. Shneiderman. A rank-by-feature framework for unsupervised multidimensional data exploration using low dimensional projections. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, pages 65–72. IEEE, 2004.

[96] K. Sequeira and M. Zaki. SCHISM: A new approach for interesting subspace mining. In *Data Mining, IEEE International Conference on*, volume 0, pages 186–193, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[97] K. Sim, V. Gopalkrishnan, A. Zimek, and G. Cong. A survey on enhanced subspace clustering. *Data Min Knowl Disc*, 26(2):332–397, Mar. 2013.

[98] B. Sluban, M. Juršič, B. Cestnik, and N. Lavrač. Exploring the power of outliers for cross-domain literature mining. In *Bisociative Knowledge Discovery*, pages 325–337. Springer, 2012.

[99] A. Strehl and J. Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. 3:583–617, Dec. 2002.

[100] A. Tanay, R. Sharan, and R. Shamir. Biclustering algorithms: A survey. *Handbook of computational molecular biology*, 9:26–1, 2005.

[101] A. Tatu, F. Maas, I. Farber, E. Bertini, T. Schreck, T. Seidl, and D. Keim. Subspace search and visualization to make sense of alternative clusterings in high-dimensional data. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 63–72. IEEE, 2012.

[102] J. B. Tenenbaum, V. d. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, Dec. 2000. PMID: 11125149.

[103] S. Vadapalli, S. R. Valluri, and K. Karlapalem. A simple yet effective data clustering algorithm. In *Proceedings of the International Conference on Data Mining (ICDM'06)*, pages 1108–1112, 2006.

[104] T. L. Van, M. v. Leeuwen, S. Nijssen, A. C. Fierro, K. Marchal, and L. D. Raedt. Ranked Tiling. In T. Calders, F. Esposito, E. HÃijllermeier, and R. Meo, editors, *Machine Learning and Knowledge Discovery in Databases*, number 8725 in Lecture Notes in Computer Science, pages 98–113. Springer Berlin Heidelberg, Sept. 2014.

[105] G. I. Webb. Self-sufficient itemsets: An approach to screening potentially interesting associations between items. *ACM TKDD*, 4(1):3:1–3:20, Jan. 2010.

[106] K. Weinberger and L. Saul. Unsupervised learning of image manifolds by semidefinite programming. 70(1):77–90, 2006.

[107] R. Wetzker, C. Zimmermann, and C. Bauckhage. Analyzing Social Bookmarking Systems: A del.icio.us Cookbook. In *Proc. ECAI MSoDa*, pages 26–30, 2008.

[108] M. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Trans. Knowl. Data Eng.*, pages 1021–1035, 2005.

[109] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithms for discovery of association rules. *Data Min. and Knowl. Disc.*, pages 343–373, 1997.

[110] M. J. Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency*, pages 14–25, 1999.

[111] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *Proc. ACM SIGKDD*, pages 326–335, 2003.

[112] L. Zeng, L. Li, L. Duan, K. Lu, Z. Shi, M. Wang, W. Wu, and P. Luo. Distributed data mining: a survey. *Information Technology and Management*, pages 403–409, 2012.

[113] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Huang, and S. Feng. Balanced parallel FP-Growth with MapReduce. In *Proc. YC-ICT*, pages 243–246, 2010.

# Samenvatting

Data is al een paar jaar belangrijker dan ooit tevoren. Gezien hoe het de wereld om ons heen vormt, is het duidelijk dat het belang van data niet snel zal afnemen. Organisaties van over de hele wereld hebben het grote belang van data begrepen en geprobeerd alle informatie waartoe ze toegang hebben op te slaan.

Omdat het niet triviaal is grote hoeveelheden data te annoteren, zijn unsupervised methodes zeer geschikt om grote hoeveelheden data te analyseren. Terwijl cluster analyse kan gebruikt worden als een pre-processing stap om gegevens samen te vatten en voor te bereiden voor supervised technieken, kunnen *frequent pattern mining* methoden gebruikt worden om interessante verbanden te vinden in de data.

Een van de kenmerken van de BigData is de hoeveelheid data die gekoppeld is aan een enkel dataobject. Hoewel meer data voor een beter en nauwkeuriger begrip zorgt, is het ook mogelijk dat de redundantie in de data het ontdekken van interessante relaties hindert. Bovendien, is het mathematisch aangetoond dat wanneer het aantal dimensies verhoogt, dataobjecten steeds meer op elkaar lijken. Dit fenomeen staat bekend als de *curse of dimensionality*.

Omdat het concept gelijkaardigheid vervormd is in hoog dimensionale ruimtes, kunnen traditionele clustering methoden, die sterk afhankelijk zijn van similarity measures, niet effectief zijn. Anderzijds, is de gelijkaardigheid aan de hand van subsets van attributen nog steeds zinvol en kan dit worden gebruikt om interessante verbanden te ontdekken. Subspace clustering algoritmen pakken het probleem van hoge dimensionaliteit aan door een cluster te definiëren als als een paar van sets: de dataobjecten die op elkaar lijken en de dimensies waarin zij vergelijkbaar zijn. Bijgevolg degradeert het beperken van de relevante dimensies de gevolgen van de *curse of dimensionality* en benadrukt de gelijkenis van objecten door het produceren van compacte clusters. Maar het zoeken naar de relevante dimensies maakt het *NP-hard* probleem van cluster analyse nog ingewikkelder.

In deze thesis hebben we het probleem van hogedimensionaliteit benaderd vanuit een “neighborhood-oriented” oogpunt. We onderzoeken verschillende manieren om een relationele database te transformeren, zodat we meer instrumenten hebben, die we kunnen gebruiken om kennis te extraheren. Na het toepassen van de cartification transformatie kunnen *frequent itemset mining* methoden gebruikt worden op relationele databases en dankzij de neighborhood-based vertegenwoordiging zijn er nieuwe mogelijkheden voor visualisatie en interactiviteit.

De thesis is als volgt georganiseerd:

**Hoofdstuk 2** introduceert het hoofdthema van deze thesis: de *Cartification transformatie*. Cartification transformeert een relationele database naar een transactie database door het behoud van localities van individuele data-objecten als neighborhoods. Door het transformeren van de gegevensruimte, maakt cartification het gehele domein van frequent itemset mining methoden toepasbaar op het gebied van relationele gegevensanalyse. We laten zien dat de *curse of dimensionality* kan verminderd worden door het gebruik van de localities en dat zelfs het toepassen van de meest elementaire FIM methoden op gecartificeerde gegevens resultaten oplevert die de state-of-the-art cluster analysemethoden verslaan.

**Hoofdstuk 3** introduceert CLON algoritme. Cartification is een algemene transformatie die kan toegepast worden op elk type data zolang een similariteitsmaat tussen data objecten is gedefinieerd. Wanneer een totale ordening van de objecten mogelijk is, d.w.z het bestudeerde attribute is univariaat, dan krijgt de getransformeerde database sommige intrinsieke eigenschappen. Deze eigenschappen kunnen gebruikt worden om het computationele duur proces van *frequente itemset mining* te elimineren. CLON is ordegroottes sneller dan de originele cartification en geeft betere resultaten op synthetische en echte datasets.

**Hoofdstuk 4** benadert cartification vanuit een verfijnder perspectief. In plaats van een binaire transactie database te creëren, creëert CARTIRANK *ranked neighborhood matrices*. We bestuderen de eigenschappen van deze ranked neighborhood matrices en stellen een algoritme voor dat deze eigenschappen uitbuit om efficiënt tiles, die clusters vertegenwoordigen, te vinden. We laten zien dat het behoud van de werkelijke volgorde van de objecten meer robuuste resultaten oplevert.

**Hoofdstuk 5** gaat over onze tool VINeM, die het vinden van neighborhoods beschikbaar voor een breder publiek. VINeM, waarvan de naam staat voor *Visual Interactive Neighborhood Miner*, biedt een gebruiksvriendelijke interface die een intuÃŕtieve voorstelling van neighborhoods met unsupervised algoritmes combineert. Belangrijkste focus van VINeM is om de relaties tussen objecten onder verschillende attributen te tonen, zodat een gebruiker snel greep op de data kan krijgen. Dankzij de interactiviteit kan de gebruiker fundamentele manipulaties van de gegevens uitvoeren en verschillende oogpunten op de data creëren.

**Hoofdstuk 6** introduceert twee frequent itemset mining algoritmes die worden uitgevoerd op het Hadoop framework. Een van de belangrijkste beperkingen van interactieve data exploratie is de reactietijd. Gelukkig, kunnen zware mining algoritmes *uitbesteed* worden aan cloud services, zodat de beperkte resources van de klant beter kunnen benut worden. Op deze manier, zijn Dist-Eclat en BigFIM twee parallelle algoritmen die kunnen uitgevoerd worden op cloud services. We tonen aan dat de prestaties van de mining algoritmes in een MapReduce paradigma afhankelijk zijn van een evenwichtige verdeling van de data tussen de berekeningsnodes. We deden exhaustieve experimenten en onderzochtten manieren om de data en de computationele belasting optimaal te verdelen.

**Hoofdstuk 7** geeft een overzicht van onze bijdragen en concludeert de thesis met een vooruitzicht naar toekomstige mogelijkheden.