TECHNISCHE UNIVERSITÄT DRESDEN

Feedback-Driven Data Clustering

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Dipl.Medien-Inf. Martin Hahmann
geboren am 8. Juli 1980 in Lübben

Gutachter:

Prof. Dr.-Ing. Wolfgang Lehner
Technische Universität Dresden
Fakultät Informatik, Institut für Systemarchitektur
Lehrstuhl für Datenbanken
01062 Dresden

Prof. Dr. rer. nat. Thomas Seidl
RWTH Aachen
Fakultät Mathematik, Informatik, Naturwissenschaften
Lehrstuhl für Informatik 9
52056 Aachen

Tag der Verteidigung: 28. Oktober 2013

Dresden, im Februar 2014

ABSTRACT

The acquisition of data and its analysis has become a common yet critical task in many areas of modern economy and research. Unfortunately, the ever-increasing scale of datasets has long outgrown the capacities and abilities humans can muster to extract information from them and gain new knowledge. For this reason, research areas like data mining and knowledge discovery steadily gain importance. The algorithms they provide for the extraction of knowledge are mandatory prerequisites that enable people to analyze large amounts of information. Among the approaches offered by these areas, clustering is one of the most fundamental. By finding groups of similar objects inside the data, it aims to identify meaningful structures that constitute new knowledge. Clustering results are also often used as input for other analysis techniques like classification or forecasting.

As clustering extracts new and unknown knowledge, it obviously has no access to any form of ground truth. For this reason, clustering results have a hypothetical character and must be interpreted with respect to the application domain. This makes clustering very challenging and leads to an extensive and diverse landscape of available algorithms. Most of these are expert tools that are tailored to a single narrowly defined application scenario. Over the years, this specialization has become a major trend that arose to counter the inherent uncertainty of clustering by including as much domain specifics as possible into algorithms. While customized methods often improve result quality, they become more and more complicated to handle and lose versatility. This creates a dilemma especially for amateur users whose numbers are increasing as clustering is applied in more and more domains. While an abundance of tools is offered, guidance is severely lacking and users are left alone with critical tasks like algorithm selection, parameter configuration and the interpretation and adjustment of results.

This thesis aims to solve this dilemma by structuring and integrating the necessary steps of clustering into a guided and feedback-driven process. In doing so, users are provided with a default modus operandi for the application of clustering. Two main components constitute the core of said process: the algorithm management and the visual-interactive interface. Algorithm management handles all aspects of actual clustering creation and the involved methods. It employs a modular approach for algorithm description that allows users to understand, design, and compare clustering techniques with the help of building blocks. In addition, algorithm management offers facilities for the integration of multiple clusterings of the same dataset into an improved solution. New approaches based on ensemble clustering not only allow the utilization of different clustering techniques, but also ease their application by acting as an abstraction layer that unifies individual parameters. Finally, this component provides a multi-level interface that structures all available control options and provides the docking points for user interaction.

The visual-interactive interface supports users during result interpretation and adjustment. For this, the defining characteristics of a clustering are communicated via a hybrid visualization. In contrast to traditional data-driven visualizations that tend to become overloaded and unusable with increasing volume/dimensionality of data, this novel approach communicates the abstract aspects of cluster composition and relations between clusters. This aspect orientation allows the use of easy-to-understand visual components and makes the visualization immune to scale related effects of the underlying data. This visual communication is attuned to a compact and universally valid set of high-level feedback that allows the modification of clustering results. Instead of technical parameters that indirectly cause changes in the whole clustering by influencing its creation process, users can employ simple commands like *merge* or *split* to directly adjust clusters.

The orchestrated cooperation of these two main components creates a modus operandi, in which clusterings are no longer created and disposed as a whole until a satisfying result is obtained. Instead, users apply the feedback-driven process to iteratively refine an initial solution. Performance and usability of the proposed approach were evaluated with a user study. Its results show that the feedback-driven process enabled amateur users to easily create satisfying clustering results even from different and not optimal starting situations.

ACKNOWLEDGMENTS

First and foremost I'd like to thank Professor Wolfgang Lehner for being my supervisor and giving me the chance to realize this dissertation project. During the last years, he contributed lots of feedback and many ideas to my work. Often highlighting new directions and ideas for research that helped to advance my work. Despite my lack of database-enthusiasm he always had faith in my skills and gave me the freedom and encouragement to pursue my, sometimes unusual, ideas. In my opinion he presents the model of a supervisor. Due to his busy schedule he was not around most of the time and left me to roam unattended, but when I needed advice or motivation, he always found some time for me. I also want to thank Dirk Habich, who supported and guided my work during the past years. His suggestions and ideas helped me to solve many problems and always were a valuable addition to my work. Furthermore, he introduced me to the art of short-term paper writing, which has become one of my most-used skills. A big thank you goes to Markus Dumat who dealt with a lot of tedious programming issues and was an invaluable help during the implementation of my prototypes. Finally I want to thank Professor Thomas Seidl for co-refereeing this thesis.

Of course, this thesis would not have been possible without a supporting environment. Therefore, I'd like to thank the great people that make the database research group the most awesome group in the whole faculty. Everyday they created a fun and creative atmosphere that included discussions, collaboration, and scientific work as well as cake, knitting and occasional firefights. All in all, there were only a few days when I did not enjoy working with them. In particular, I want to thank the 'exciting' Maik Thiele for always showing me how 'not' to do things, the adorable Ulrike Fischer for always borrowing money from me, hipster Hannes Voigt for introducing me to baking and repertoire cinema, and the shaggy Benjamin Schlegel for essential discussions about video games and cats. Special thanks go to my pleasant office mates Steffen Preissler, Tim Kiefer, and as of late Julian Eberius. I'm also much obliged to Katrin Braunschweig for providing last-minute proofreading. Further thanks go to Bernd, Claudio, Elena, Frank, Gunnar, Ines, Kai, Lars, Matthias, Robert, Thomas, Tobias, the other Tomas, and Till for being great colleagues.

Finally but probably most importantly, I am deeply grateful for the constant backing from my family and friends. My parents always had confidence in me and supported me in every possible way. I'd also like to thank Anna, Bernhard, Claudia, David, Doreen and Rene for being great friends and for cheering me up the whole time, especially during my moody episodes in the latter stages of this thesis.

Martin Hahmann
September 12, 2013

CONTENTS

1 Introduction 9

1.1 Contribution . . . 11

1.2 Outline . . . 13

2 Contemporary Clustering: An Assessment 15

2.1 Traditional Clustering . . . 16

2.1.1 Partitional Clustering . . . 17

2.1.2 Density-based Clustering . . . 19

2.1.3 Hierarchical Clustering . . . 21

2.1.4 Assessment . . . 22

2.2 Multi-Solution Clustering . . . 23

2.2.1 Alternative Clustering . . . 23

2.2.2 Subspace Clustering . . . 24

2.2.3 Assessment . . . 26

2.3 Ensemble Clustering . . . 26

2.3.1 Pairwise-Similarities . . . 26

2.3.2 Cluster-Labels . . . 27

2.3.3 Assessment . . . 28

2.4 Clustering Interpretation . . . 28

2.4.1 Quality Measures . . . 29

2.4.2 Visualization . . . 29

2.4.3 Adjustments . . . 32

2.4.4 Assessment . . . 32

2.5 Consequences . . . 32

3 Principles of a Versatile Clustering Process 35

3.1 Handling of Algorithms . . . 36

3.2 Interacting with the Clustering . . . 37

3.3 Iterative Refinement . . . 38

3.4 Summary . . . 38

4 Algorithm Management 41

4.1 Algorithm Description . . . 42
4.1.1 Extracting the Essentials of Data Clustering . . . 43
4.1.2 Formal Description . . . 45
4.1.3 Transcription of Algorithms . . . 50
4.1.4 Summary . . . 66

4.2 Integration . . . 67
4.2.1 Flexible Clustering Aggregation . . . 67
4.2.2 Frequent Groupings . . . 75
4.2.3 Summary . . . 87

4.3 Control Interface . . . 87

4.4 Summary . . . 90

5 Visual-Interactive Interface 91

5.1 High-Level Feedback . . . 93
5.1.1 Mapping to the Flexible Clustering Aggregation . . . 95
5.1.2 Mapping to Frequent Groupings . . . 95

5.2 Hybrid Visualization Concept . . . 97

5.3 Visualization I: Large Screen . . . 99
5.3.1 Overview . . . 100
5.3.2 Cluster Description . . . 101
5.3.3 Attribute View . . . 103
5.3.4 Summary . . . 104

5.4 Visualization II: Small Screen . . . 104
5.4.1 Process View . . . 105
5.4.2 Clustering View . . . 106
5.4.3 Cluster View . . . 107
5.4.4 Attribute View . . . 110

5.5 Summary . . . 113

6 Process Application 115

6.1 A Guided Tour to Iterative Refinement . . . 116

6.2 Learning by Doing - Process Adaptation . . . 123

6.3 Summary . . . 125

7 Evaluation 127
7.1 User Study Setup . . . 128
7.2 Application Results . . . 129
7.2.1 Group A . . . 131
7.2.2 Group B . . . 132
7.2.3 Group C . . . 134
7.3 Usability Results . . . 135
7.4 Summary . . . 136
8 Conclusion and Outlook 139

INTRODUCTION

1.1 Contribution

1.2 Outline

We live in the age of information. Of course this is hardly news, as it has been around for several decades. While it is debatable, whether it started with the invention of the transistor, the introduction of the home computer or the emergence of the Internet, it is a fact that it permeates and defines every part of modern life. Due to constant progress in science and engineering, the agents of this age have become legion. Computers, smart phones, sensors, and embedded systems are ubiquitous and found their way into schools, homes, industry, and everywhere else. There, they fuel an ongoing massive trend for digitalization and data collection that has become one of the most essential characteristics of this age. Nearly every imaginable aspect can and is measured, stored and made available as data. With this development, data and information have become resources for today's work that are as crucial as coal, steel, and oil in the prior age of industrialization. In contrast to those diminishing natural materials, the deposits of data are constantly growing.

While data itself is already valuable, it is only capitalized to the full extent by obtaining knowledge from it. Although human capabilities for data analysis and knowledge extraction are remarkable, they simply cannot match the vast scale of available data. As science and industry already start to prepare for Exascale computing, the average human cannot even process datasets of several kilobytes, e.g. a table with 10000 rows and 10 columns. To solve this problem, research areas like machine learning, data mining, and knowledge discovery have been established and provide tools that can process amounts of data that cannot be analyzed manually. The methods and algorithms developed in this field are designed to automate, emulate, and augment the different ways of human reasoning in order to allow the extraction of useful knowledge from data. Along with these tools, data scientists and analysts emerged as a new class of professionals skilled in their application.

One of the most basic approaches that humans use to gain understanding is grouping. When facing a complex and unknown situation that includes many elements, humans typically observe it and try to find similarities in their appearance and/or behavior. Based on this, elements are arranged into groups. From these, new knowledge can be derived, based on the assumption that similarity of elements indicates a semantic connection or functional dependency. With this approach, even complex issues can be partitioned, explored, and comprehended. Due to these characteristics, this approach is also used for data analysis in which it is represented by the technique of *clustering*. It describes the problem of partitioning a set of objects into groups, called clusters, so that objects in the same cluster are similar, while objects in different clusters are dissimilar.

Although this problem seems to be clearly defined, its solution is extremely challenging due to two main reasons. The first one is the variable and subjective character of similarity. While there exists a multitude of functions to determine this feature, each one is based on a different notion of similarity. For this reason, the similarity between objects depends on the employed measurement function. Furthermore, simply obtaining a value for the similarity between objects is not enough to decide whether they are similar enough to be grouped together or not. For this decision, a threshold must be defined. Unfortunately, the perception of similarity is very individual and depends on the application domain, experience, and subjective attitude. Obviously, it is hard to model such a frame for decision making in an algorithm.

The second big challenge of clustering is working with the unknown. As mentioned earlier, we want to extract new and useful knowledge from the data. While every identified cluster structure must be considered as new knowledge, it is not guaranteed that it is actually useful. Since clustering aims to acquire new knowledge, there is no ground truth available that could be used to validate its findings. Therefore, clustering results can be considered as proposals that must be validated in the application domain from which their underlying data originates.

Over the years, a multitude of clustering algorithms has been proposed [40]. All of them share the characteristic of being not versatile or robust, meaning that certain algorithms and parameterizations

only suit certain datasets and will yield poor results otherwise. In order to tackle the mentioned challenges to a certain degree, specialization has become a trend in clustering. More and more algorithms are designed specifically for a narrow and clearly defined application scenario. By introducing domain experts and background knowledge, similarity measurement and thresholds can be tailored to the task and even result validation is possible to a certain degree. On the downside, this trend towards customization is very costly and the reusability becomes nearly non-existent.

In addition to these traditional clustering algorithms, higher-order approaches that work with multiple clusterings have been proposed. *Alternative clustering* creates multiple clustering solutions for a dataset and thus can provide several views on complex data. *Subspace clustering* addresses high-dimensional data and identifies subspaces that are interesting for clustering. In doing so, it tackles two problems. The first one is also known as curse of dimensionality and manifests in distance measures becoming less meaningful with an increasing number of dimensions. The second one is that it becomes more and more unlikely that a single meaningful clustering solution can be found in a space defined by a large number of attributes. The third main approach based on multiple clusterings is *ensemble clustering*, which tackles the challenge of robustness and versatile application by integrating a set of multiple traditional clusterings into a single consensus clustering result.

In summary, contemporary clustering offers a wide selection of specialized tools that are clearly aimed at experts. This is an unfavorable situation, as large amounts of data are accumulated in more and more application domains, by which clustering becomes a necessary technique for analyzing this data. With the evolution from a niche application in research to a widespread analysis technique, new users come into contact with clustering. Thus, formerly neglected topics like usability and applicability become important issues. At present, users that want to apply clustering have two choices: either they hire experts and set up a multi-year project to develop a customized clustering solution, or they embark on a journey of trial-and-error by trying to build a fitting solution from existing techniques. This thesis focuses on changing this current state by evolving clustering into a feedback-driven process.

1.1 CONTRIBUTION

In general we can state that clustering suffers from an abundance of tools and a lack of guidance on how to use them. In order to create a clustering, users must choose a suitable algorithm and execute it with fitting parameters. Both actions have crucial impact on the result and can lead to useless clusterings if not performed correctly. After creation, the user has to examine the obtained result and decide whether its clusters are feasible or not. If the outcome is not satisfying, parameters or algorithm must be changed to create an improved result. Again, a certain level of understanding is necessary to effectively carry out these tasks. Current clustering practice is mainly focused on clustering creation and considers all of the mentioned actions as individual tasks. Due to this loose coupling of actions and the arbitrariness of their execution, clustering is a hardly accessible technique for amateur users.

The main contribution of this thesis is to present a way out of this dilemma by dismissing current clustering practice and replacing it with a novel integrative structure that reorganizes the clustering procedure as a whole. The design of said structure strongly emphasizes the user and aims to relieve him from the complex and technical aspects of clustering, to allow a focus on working with the clustering. To achieve this, the necessary actions for clustering are no longer considered as autonomous steps, but as parts of a bigger picture. Thus, they are tightly coupled and integrated into a structured process that provides users with a default procedure to follow. Our process is not meant to be a fixed clustering algorithm that can handle all possible datasets, but defines a tool that allows the easy and versatile application of existing techniques. We will use abstraction and simplification to create our

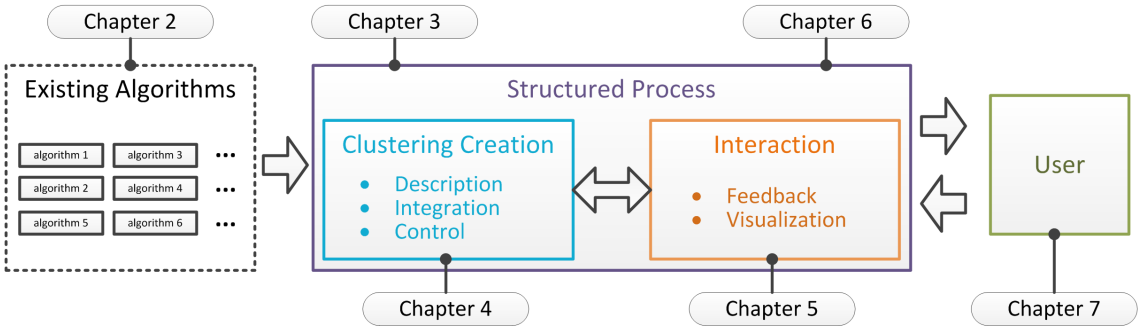


Figure 1.1: Outline of a feedback-driven clustering process.

process as a template, providing certain core characteristics that can be realized in different ways to ensure versatility. To reach our goals, we develop a new form of interaction that allows users to understand the obtained clustering results and improve them via natural feedback instead of technical parameters. Such an interaction also requires new approaches for actual clustering creation. These must be able to handle the variety of existing clustering algorithms and furthermore must provide an interface that can translate user feedback and implement it in the clustering result. Subsequently, we give a summary of the contributions made in this thesis.

- We give a detailed overview and assessment of the existing techniques that make up contemporary clustering practice. We concentrate on the main classes of clustering algorithms and introduce some of their representatives to convey an impression of the existing diversity. In addition, we describe existing ways for the evaluation of results. All introduced techniques are assessed in terms of applicability and usability. This includes aspects of versatility, robustness, configuration, complexity, and support for users. From our assessment we derive the main consequences users face in contemporary clustering.

- We define a template for a *versatile clustering process*. This includes the description of the main tasks and requirements that are necessary to realize a holistic and user-friendly application of clustering. The process provides two main functions. The first one is to handle clustering creation in its entirety which includes algorithm description and specification, options for the integration and management of existing clustering techniques, and the development of a control interface that accepts and implements feedback coming from the user. The second main function is the provision of an interaction interface which communicates the characteristics of the clustering result to the user and offers direct feedback actions that allow the improvement of the current result. Our process tightly couples and attunes both functions in order to allow a seamless application.

- We propose a *modular approach* for the description and design of clustering algorithms. The general concept of clustering algorithms is broken down into its basic components, from which building blocks are derived. This modular structuring eases understanding and allows the comparison of algorithms on a block-to-block level. Furthermore, it enables users to construct new clustering algorithms or modify existing ones by simply assembling/switching sets of blocks.

- We propose new approaches for the *integration* of existing clustering techniques into our process. For this, the concept of ensemble clustering is used as a starting point because it allows the combination of multiple results into a single final solution, which in addition often improves overall quality. We expand this concept and propose techniques for the controllable integration of multiple results. Due to our proposed control options, these integration methods also act as an abstraction layer for the management of multiple different clustering algorithms.

- We introduce a *multi-level control interface* that collects and structures the control options offered by our modular algorithm description and our proposed integration techniques. Via these levels, the interface creates docking points for user interaction.

- We define a compact set of universally valid *feedback options* for the adjustment of a clustering. For this we take the users point of view, abstract the most basic adjustments for a clustering, and convert them into feedback. While traditional adjustments are made by changing the cause of a clustering, i.e. the parameters used during its creation, our feedback directly represents the effects a user wants to establish in the result. In addition to this direct character, usability is improved by keeping feedback fixed and independent from the underlying algorithms.

- As existing approaches for clustering interpretation either provide too much or too little information, we propose a novel *hybrid visualization concept* that represents a middle way between these extremes. Our visualization presents not the whole dataset, but concentrates on the basic aspects that are necessary for the interpretation of a clustering's characteristics. The displayed information is organized over different views that allow a systematic examination of the result without overburdening the user. Communication as a whole is designed with our feedback in mind and supports the user in deriving appropriate actions for clustering adjustments. Due to the focus on certain aspects and the distribution of information over several views, our proposed concept does not suffer from high-dimensional or high-volume data and can be realized with simple and familiar visual components.

- To assess the improvements that our clustering process provides, we conduct a user study. Performance is evaluated by letting users carry out different clustering tasks, while usability is rated with the System Usability Scale questionnaire.

1.2 OUTLINE

The outline of this thesis and the main parts of our desired feedback-driven clustering process are depicted in Figure 1.1. It starts with Chapter 2, in which we give an overview on the current clustering practice and assess existing techniques. In the following Chapter 3, our process and its main tasks are defined. After this, Chapter 4 covers the realization of the necessary methods for clustering creation. This includes the introduction of our approaches for modular algorithm description and clustering integration, as well as the specification of the control interface. During Chapter 5, we focus on the design of user interaction which includes the description of feedback, our visualization concept, and their coupling.

In Chapters 4 and 5 all algorithms and methods that are necessary to fulfill the tasks and requirements defined in Chapter 3 are proposed. This allows the fully operational realization of our process in Chapter 6. An exemplary application of it is described in a step-by-step fashion, using a small example scenario. In addition, we present further ways for the adaptation of our process to different scenarios. Since the contributions of this thesis are focused on the user, a small user study is conducted for the purpose of evaluation. Chapter 7 describes the setting of this study and its results regarding performance and usability. At last, this thesis is concluded in Chapter 8.

CONTEMPORARY CLUSTERING: AN ASSESSMENT

2.1 Traditional Clustering

2.2 Multi-Solution Clustering

2.3 Ensemble Clustering

2.4 Clustering Interpretation

2.5 Consequences

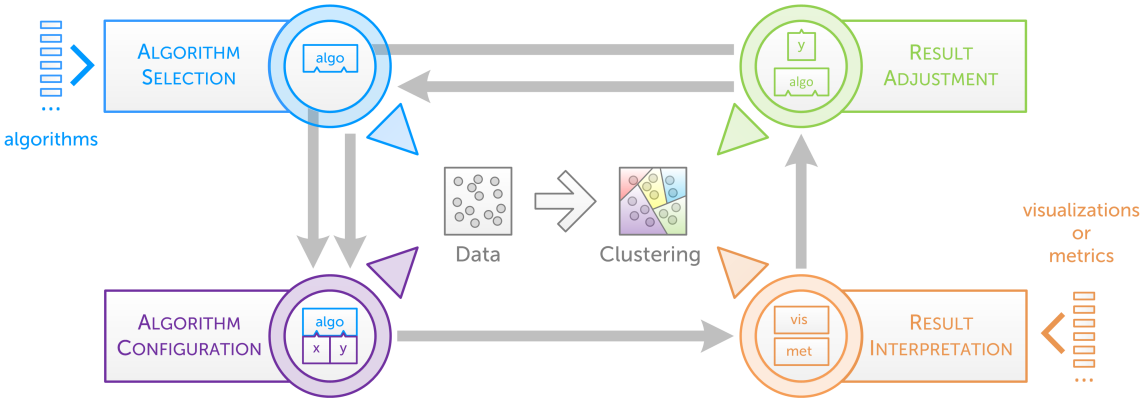


Figure 2.1: Steps of Contemporary Clustering.

The contemporary clustering practice usually requires the completion of the four steps shown in Figure 2.1. Each one of them requires the user to make different decisions that have a strong influence on the final result. To provide a general definition, we describe these steps as: *Selection* of an algorithm. *Configuration* of the algorithm and subsequent execution. *Interpretation* of the obtained result. *Adjustment* of algorithm and parameters to improve result quality.

All of these steps are demanding, even for experts with experience in clustering. Therefore, they form a substantial challenge for inexperienced amateur users. All problems that occur during their completion, typically result from the enormous amount of possible choices and the general lack of background-knowledge. Let us regard algorithm selection as an example, in which the user is confronted with a multitude of available algorithms. To choose the optimal algorithm for the particular task, the user must be aware of all existing clustering techniques and their functionality. Furthermore, the user must know which one of them is the best fit for the dataset that should be analyzed. Basically, these problems recur during algorithm configuration, when similarity measures and parameters must be chosen and also during result interpretation when the user must select a method for cluster validation.

Contemporary clustering practice provides different approaches for the creation of a clustering, in which one or more of the mentioned steps are addressed in different ways and with different priority. In the course of this chapter, we introduce several of them and describe some of their associated algorithms. Our descriptions are complemented with an assessment of the handling of each step and the resulting implications for the user. Research in clustering is mostly focused on the development of new clustering methods and their enhancement. Therefore, the first three sections of this chapter will mainly address clustering creation and thus deal with the algorithm selection and configuration step. Our descriptions cover traditional clustering, multi-solution clustering, and ensemble clustering. In the fourth section, we discuss the more user-centered steps of result interpretation and adjustment.

2.1 TRADITIONAL CLUSTERING

We use the term traditional clustering to subsume all clustering techniques which comply with the following mode of application: *one* algorithm is executed using *one* set of parameters, generating *one* clustering solution for the given data. Later in this chapter, we introduce approaches that differ from this application mode in some points, which is why the term 'traditional' is used for differentiation. The area of traditional clustering contains the bulk of available clustering algorithms. In order to structure it, we divide it into three main classes: *partitional*, *density-based* and *hierarchical* methods. Please note that this division is neither fixed nor universally valid, as the diversity of the field allows for many other.

2.1.1 Partitional Clustering

Members of this class use the following basic approach to group a set of *n* objects into k clusters: Each of the k clusters is represented by a so-called prototype. To assign each object to a cluster, the similarities between objects and prototypes are determined by employing a similarity/distance measure like the *euclidean distance*. In doing so, a $n \times k$ similarity matrix is constructed. Following this similarity evaluation, each object is assigned to the prototype with the highest similarity and its corresponding cluster. The initialization of prototypes often incorporates random factors and thus is most likely not optimal. Therefore, most partitional algorithms iteratively optimize prototypes with regard to a specific objective function.

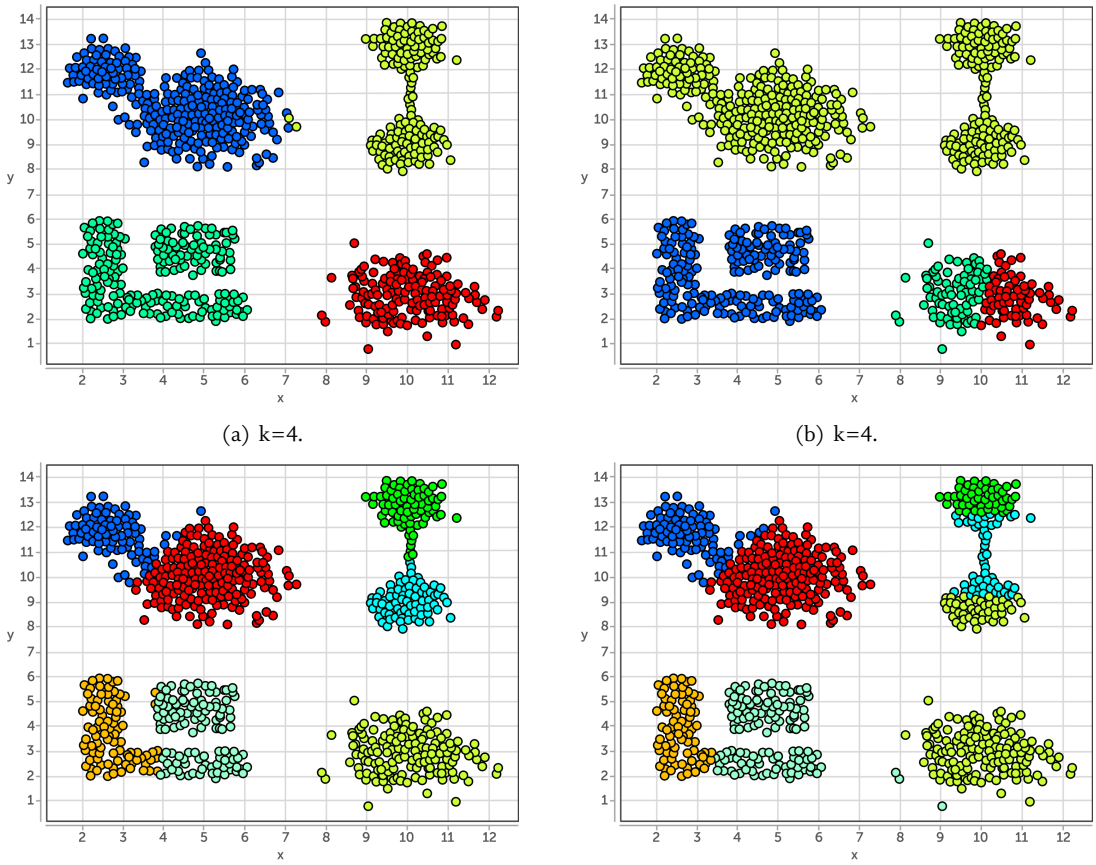
The best known member of this class of algorithms is *k-means*. Although it was developed more than fifty years ago by [61], [53], and [50] it is still the most popular and most used clustering algorithm due to its simplicity and ease of implementation. K-means works as follows: assuming a set of d-dimensional objects $P = p_i, i = 1, \ldots, n$, k-means partitions P into k clusters $C = c_j, j = 1, \ldots, k$ so that the squared error between a cluster's prototype and its assigned objects is minimized. The prototype of a cluster c_j is defined as the mean μ_j of its objects therefore the squared error is defined as:

$$J(c_j) = \sum_{p_i \in c_j} \|x_i - \mu_j\|^2 .$$

In order to minimize the sum of squared errors over all clusters, k-means works in an iterative fashion. After generating an initial set of k prototypes, the following steps are repeated: (i) assign each object to the most similar prototype/cluster, (ii) update prototypes as mean of their assigned objects. These two steps are repeated until the clusters stabilize, i.e. their prototypes do not change during update or a pre-specified number of iterations is reached.

To execute the k-means algorithm, the user needs to configure several parameters: the number of clusters k, an initialization for the prototypes, and a stopping criterion for the iteration. Furthermore, the utilized distance metric can be considered as an additional parameter. All of these configuration choices have a critical impact on the clustering result. To illustrate this impact we prepared a toy dataset and some example clusterings that are shown in Figure 2.2. The first two clusterings were generated with the same value for k but different prototype initializations—in this case: seed values for a random generator drawing the initial prototype positions—and show very different results. While the clustering in Figure 2.2(a) identifies a cluster in each quadrant—an adequate yet not optimal solution—the result shown in Figure 2.2(b) incorrectly groups all objects of the upper quadrant into the same cluster and furthermore splits the single cluster located in the lower right quadrant. Even if we use the correct number of clusters—which is 7 in our example—to set k, an improper initialization can still lead to a non-satisfying result like the one depicted in Figure 2.2(c), where the clusters in the lower left quadrant are not separated correctly. Structures with arbitrary shape are a general problem for algorithms of the partitional class that are inherently designed to detect convex spherical clusters. This characteristic makes it ultimately impossible for k-means to correctly identify the 'L-shaped' cluster in our example. To conclude, we illustrate the influence of the stopping criterion in Figure 2.2(d). So far, a fixed number of 4 iterations was used as a stopping criterion for all our clusterings. Regarding our small example, this is sufficient as the centroids stabilize early. For the clustering in Figure 2.2(d) we used the same configuration as in Figure 2.2(c) but stopped the optimization process after just 2 iterations, resulting in a clustering with considerable differences to the former result.

What makes the configuration of k-means even more challenging, is the fact that there exists no reliable analytical way to determine optimal or even feasible choices for its parameters. In practice,



(a) k=4. (b) k=4.

(c) k=7. (d) k=7.

Figure 2.2: K-means Clustering Results for Different k and Cluster Initializations.

different configurations are tested by running k-means repeatedly while varying one or more parameters. From the results obtained during this 'trial and error' procedure the one that appears to be most meaningful to a domain expert is chosen.

These shortcomings have been addressed by several variations of the k-means algorithm. One of the earliest approaches that tackles the problem of finding the right value for k is *ISODATA*[8]. This method is able to adjust the number of identified clusters after each iteration of the algorithm. The adjustment is made by either: deleting too small clusters, combining similar clusters or dividing heterogeneous clusters. Each of these options is selected with regard to pre-specified thresholds, thus the extended functionality comes at the cost of multiple additional parameters. Another approach to this issue is the *x-means* algorithm [57] that increases the number of centroids in an iterative fashion. This is done by running conventional k-means until convergence in a first step. After that, the algorithm splits the existing centroids and runs k-means again. The newly added centroids are rated using the *Bayesian Information Criterion (BIC)* or the *Akaike Information Criterion (AIC)* and are only kept if this score is improved. Further approaches in this direction are *g-means* [36] and *pg-means* [22] which use gaussian mixtures to model the data/clusters and statistical tests like the Kolmogorov-Smirnov test for scoring added centroids.

In addition to algorithms which use centroids as prototypes, there is a sizable group of methods which employ medoids as prototypes. A centroid is defined as the average of a set of objects and thus exists as a virtual point in the datasets feature space. In contrast, a medoid is an actual object of the dataset which means that medoids are not calculated, but selected. Apart from that, medoid-based algorithms also try to minimize an objective function. The best known representative of this group of algorithms

is *k-medoids* [45] and its most common realization *Partitioning Around Medoids (PAM)* [46]. Further variants of this algorithm are *CLARA* [46] and *CLARANS* [55] both of which combine the idea of PAM with sampling-techniques to allow the fast clustering of large datasets.

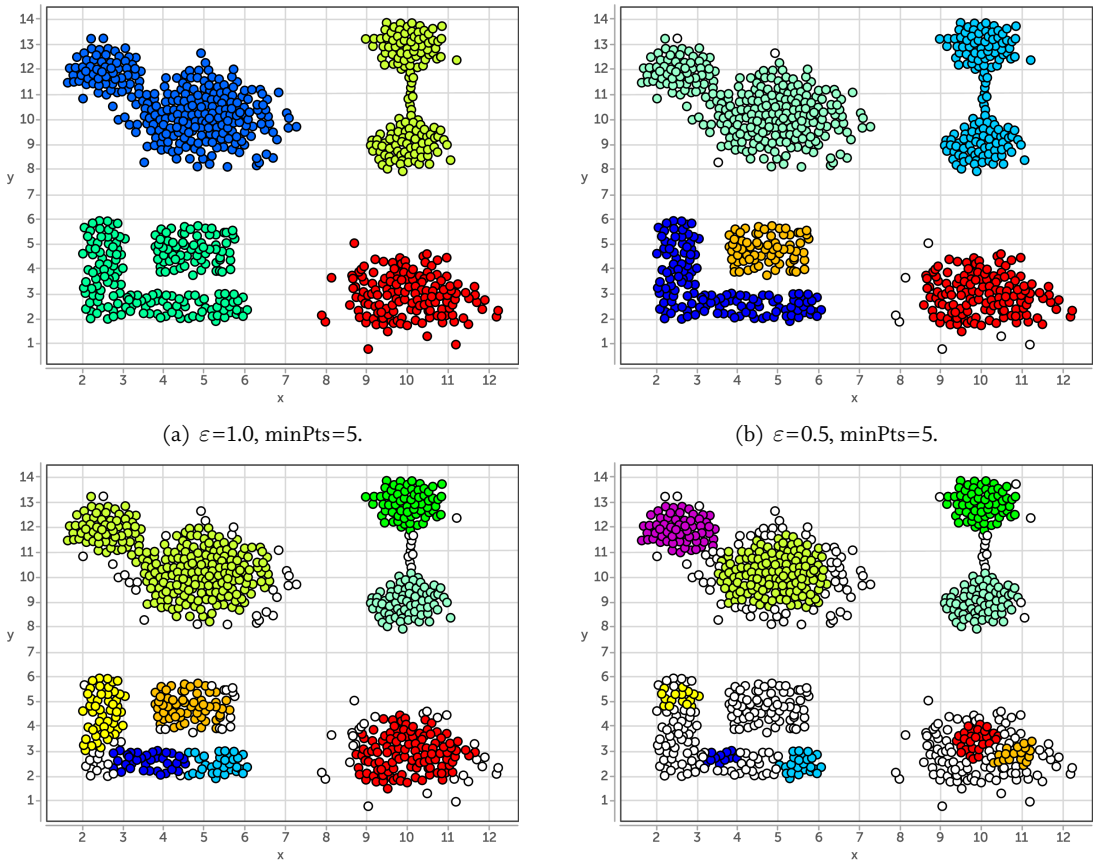
2.1.2 Density-based Clustering

Algorithms belonging to this class define clusters as areas of high density in d-dimensional space, which are separated by areas of lower density. Based on this definition, density-based clustering methods try to identify dense regions inside a dataset and connect them to clusters. To take care of this task, different algorithms utilize different ways to model density and connections.

As an example, we introduce *DBSCAN* by Kriegel et al.[21]. This algorithm models density by using two parameters ε and *minPts*, with ε defining the size of a neighborhood around each data object and *minPts* specifying the number of objects that must be located in this neighborhood in order to be qualified as a high-density region. Each object that satisfies the given density requirements is called a *core object* and thus is considered as part of a cluster. To form clusters these core objects are connected by evaluating the overlap of their ε-neighborhoods, i.e. if two core objects are located in each other's ε-neighborhood they are connected and are members of the same cluster. To execute DBSCAN the user has to specify a density threshold via the parameters ε and *minPts*. Again, algorithm configuration has a vital influence on the obtained result, which can be seen in the example clusterings for our toy dataset, illustrated in Figure 2.3.

We start by choosing a relatively big ε and a small *minPts*, thus specifying a low density that results in the clustering depicted in Figure 2.3(a). This clustering clearly separates the four main populated areas of our toy dataset from each other and in doing so is quite similar to the first result obtained with k-means in Figure 2.2(a). By further raising the threshold for high-density areas, different results can be produced. For the clustering depicted in Figure 2.3(b) ε was halved while *minPts* was not changed, which leads to the correct identification of the 'L-shaped' and rectangular cluster in the lower left quadrant. In Figure 2.3(c) the value of *minPts* was quadrupled to increase the density threshold. In doing so the obtained clustering detects two clusters in the upper right quadrant that were connected by a small bridge of objects in former results. However, this configuration also has a negative side as it breaks up the clusters in the lower left into smaller parts as the core-objects located there are no longer connected using the given parametrization. Besides the changes in the discovered structures, higher density-thresholds also lead to an increasing amount of objects that are neither in a dense region nor connected to one. These objects are treated as noise and depicted by dots without coloring in Figure 2.3. In general, it is a problem of DBSCAN that the density threshold is defined globally and is thus valid for the whole dataset. This can be seen in Figure 2.3(d), where the configuration allows a correct identification of clusters in the upper half, but simultaneously classifies the structures in the lower half as noise.

Density-based clustering methods like DBSCAN possess many beneficial characteristics. They are able to identify clusters of arbitrary shape much better than partitional clustering methods and are furthermore able to handle noise by filtering objects in low density regions. In addition, they do not need a pre-specified number of clusters. On the downside, they are susceptible to the curse of dimensionality as high-dimensional data spaces are often sparsely populated, which hinders the differentiation between high- and low-density regions. Furthermore, some configurations can lead to the so-called 'single-link' effect, which means that otherwise separated object accumulations that are linked by small object strings are identified as one cluster because the dense areas are connected although the linking objects may be of low significance. Like partitional clustering algorithms, optimal configuration is a problem. While it is beneficial that the number of clusters must not be pre-specified,



(a) ε=1.0, minPts=5.

(b) ε=0.5, minPts=5.

(c) ε=0.5, minPts=20.

(d) ε=0.4, minPts=20.

Figure 2.3: DBSCAN Clustering Results with Different Densities.

the density threshold itself is a complex composite of ε and *minPts*. If we perceive density as objects per unit of volume, it is possible that different configurations of ε and *minPts*, specifying the same density, lead to different clustering results. Nonetheless, configurations that specify different densities can also result in the same clustering. Although some heuristics for parametrization exist, the task remains challenging and obtaining feasible parameters is not guaranteed.

Again there exist variations of this algorithm that deal with some of its shortcomings. One problem of DBSCAN is its singular global density threshold. Although all clusters exceeding this threshold are detected, there can be problems if the variance in cluster-densities above the global threshold is high. This holds especially for hierarchical clusters, e.g. a set of high-density clusters located inside a connected region of lower density that still exceeds the global density threshold. An algorithm that tries to tackle this issue is *Ordering Points To Identify the Clustering Structure (OPTICS)* [3]. This approach generates an ordering of the points of a dataset with concepts from DBSCAN, i.e. core-objects and reachability-distance, by employing the parameters ε and *minPts*. The produced ordering contains information on the local densities and thus the clustering structure inside the data, but generates no explicit clustering solution. Actual clustering extraction can be done interactively using a reachability plot or automated by analyzing the different degrees of steepness throughout the reachability plot to find the different cluster regions.

In addition, the class of density-based clustering algorithms consists of a variety of methods, each one providing an individual answer to its two basic problems: 'How to identify dense regions?' and 'How to connect dense regions?'. The *DENCLUE* algorithm [37], for example, utilizes kernel functions to model density and hill climbing to connect dense regions, while *SNN*[20] employs the probability

density of points for density-modeling and a shared nearest neighbor graph for connection. Further algorithms are based on Wavelet Transformation [59], physical laws like the Yukawa potential [6] or a grid structure [65].

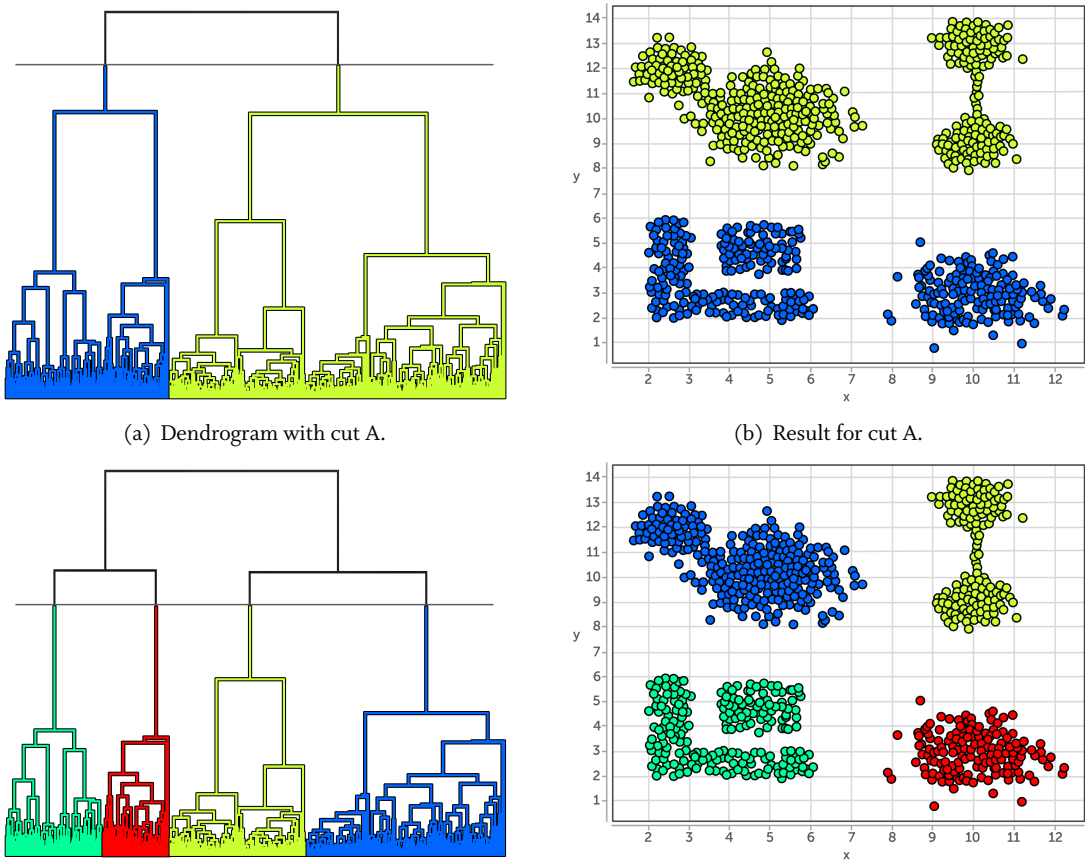
2.1.3 Hierarchical Clustering

In contrast to the two algorithm classes described so far, this class of clustering algorithms does not produce a single clustering result, but a hierarchy of clusters from which different clustering solutions can be extracted. Such a hierarchy is often presented as a tree structure, a so-called *dendrogram*. The root of the dendrogram contains a single cluster that includes all objects of the dataset, while each leaf corresponds to a singleton cluster with just one object. The construction of a dendrogram is the main task of hierarchical clustering algorithms and can be conducted in two ways. One way works in a bottom-up or *agglomerative* fashion and thus starts working at the leaf-level and uses all singleton clusters of a dataset as input. Starting from there, the two most similar clusters are identified and merged into a new cluster, which forms a new node in the dendrogram and substitutes both parent clusters during further similarity calculations. This merging procedure is repeated until only one cluster remains that represents the root of the dendrogram. In contrast, the second approach to dendrogram construction works in a *divisive* way by starting at the root and iteratively splitting the most heterogeneous cluster into two new clusters until only singleton clusters remain, i.e. the leaf-level is reached. Examples for these two approaches are the *agglomerative nesting* and *divisive analysis*[46] algorithms. To determine the similarity between clusters any distance measure can be used. Hierarchical clustering algorithms evaluate the similarity between groups of objects and not between a pair of objects. Therefore, the similarity measurement can be carried out in different ways, by assuming that the distance between two clusters corresponds to the: (i) maximum, (ii) minimum, or (iii) average element-to-element distance between the clusters.

- *complete-linkage*: the distance between two clusters corresponds to the maximum element-to-element distance between both clusters
- *single-linkage*: the distance between two clusters corresponds to the minimum element-to-element distance between both clusters
- *average-linkage*: the distance between two clusters corresponds to the mean element-to-element distance between both clusters

In order to generate an actual clustering result from the dendrogram, it needs to be cut at a specified height, thus collapsing into disconnected sub-trees that represent the clusters. The granularity of the clustering is influenced by the cutting height. While a cut near the root will produce a coarse clustering with few, but large clusters, a cut near the leaf-level will produce a finer grouping with many, but small clusters. Besides the employed distance measure and mode, the cut is the third parameter that must be specified by the user. In Figure 2.4 some example clusterings are depicted. These were generated with agglomerative nesting, using the Euclidean distance and average-linkage. The resulting dendrogram with two example cuts are shown in Figure 2.4(a) and Figure 2.4(a), while Figure 2.4(b) and Figure 2.4(c) show the clustering result for cut A and cut B, respectively.

Hierarchical Clustering algorithms have two main challenges: (i) determination of the intra-/inter-group similarity in order to make the decision to split/merge clusters and (ii) how to extract a final clustering solution from the generated dendrogram. Subsequently some algorithm variants that address these issues are described. All of them work in an agglomerative fashion. The *CURE*[27] algorithm mainly addresses the issue of inter-cluster similarity calculation. One way to determine this



(a) Dendrogram with cut A.

(b) Result for cut A.

(c) Dendrogram with cut B.

(d) Result for cut B.

Figure 2.4: Agglomerative hierarchical clustering examples.

similarity is evaluation of the distances between all objects by means of single-, average- or complete-linkage. Another way is to generate a centroid per object group and use the centroid to centroid distances for merging decisions. Both approaches have their strengths and weaknesses: centroids are fast to compute, but can only roughly approximate a group of points, while the calculation of pairwise distances is costly and makes the algorithm susceptible to noise. Therefore, the authors of *CURE* use a compromise between both approaches by selecting a group of well scattered objects as representative for each cluster. These objects are then shrunken towards the cluster centroid by a user-defined fraction, allowing a much better approximation of the cluster's shape. In contrast to the agglomerative nesting algorithm, CURE does not output the dendrogram, but a single clustering solution. The algorithm stops its merging procedure if a user-specified number of clusters is reached. A variation of CURE is *CURD*[52], which uses a density based approach inspired by DBSCAN to generate a cluster-specific number of representatives. An additional agglomerative algorithm is *CHAMELEON* [42] that measures the similarity between clusters with a dynamic model. CHAMELEON relies on a k-nearest neighbor graph and the measures of inter-connectivity and closeness for its merging decisions. The user has to specify thresholds for both measures and if these cannot be fulfilled the merging procedure stops and the latest clustering is exported as a result.

2.1.4 Assessment

This section shows that the field of traditional clustering offers a wide selection of algorithms that is ever growing. This makes the steps of algorithm selection and configuration a huge challenge for

amateurs and even experienced users. The provided algorithm examples were chosen in order to illustrate the predominant trend of specialization and variation which focuses on the creation of new algorithms with minor changes, e.g. via a switch of the distance measure or addition of a parameter. Typically, this is done to tailor algorithms for specific application scenarios, which leads to the problem that an optimal algorithm selection can only be made if each method's suitability for the task at hand is known. Besides selection, the setup of parameters is a non-trivial and algorithm-specific task. As reliable analytic ways for parameter determination are virtually non-existent, support for this step is only provided by heuristics, rules of thumb or a data-mining expert's assistance. This also affects the adjustment step, as adjustments are made via switching of algorithms and/or re-parametrization. All these problems lead to cluster-analysis becoming a trial-and-error procedure.

2.2 MULTI-SOLUTION CLUSTERING

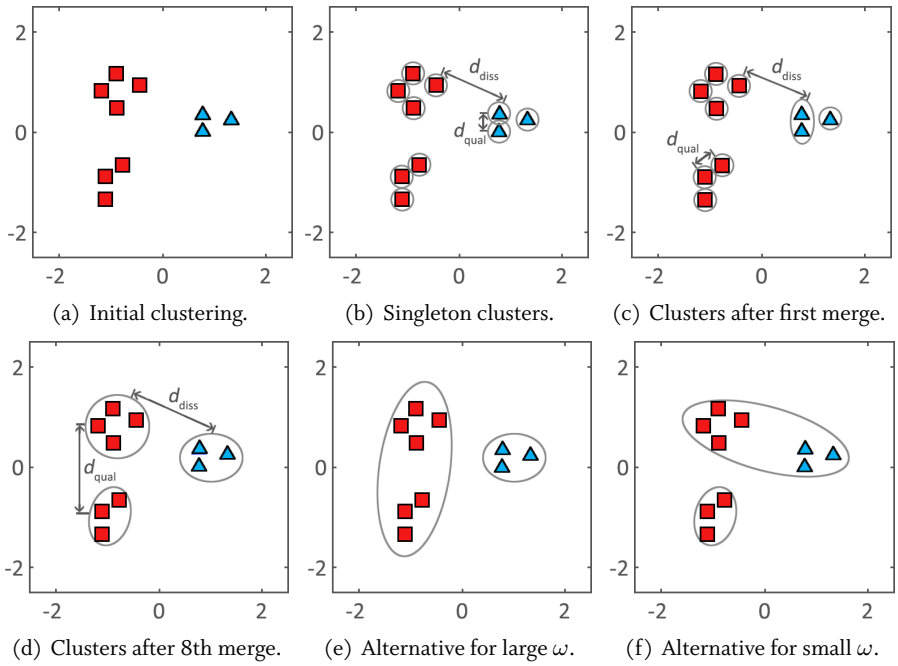
As previously described, traditional clustering often leads to multiple iterations in which different parameters or algorithms are tried until a satisfactory result is obtained. This practice implicitly generates multiple clustering solutions for the analyzed dataset. The concept of *multi-solution clustering* takes up this characteristic and utilizes it explicitly, whereas the mode of application changes to: *one* algorithm is executed using *one* set of parameters, generating *multiple* clustering solutions for the given data. The algorithms of multi-solution clustering can be divided into two main classes: *alternative* and *subspace* clustering methods.

2.2.1 Alternative Clustering

The main goal of this class is to provide alternative clustering solutions to the user. To create different alternatives, at first an initial clustering result is determined, using a traditional clustering algorithm. Based on the information contained in this initial clustering, alternative solutions are generated so that these alternatives are *dissimilar* to the initial solution.

As an example for alternative clustering, we describe the *COALA* [5] algorithm. Given a Clustering C with k clusters this method generates a dissimilar alternative S also having k clusters. In their approach, the authors use instance-based 'cannot-link' constraints to express this dissimilarity. These are derived from the initial clustering and are employed in the construction of the alternative. Such a constraint can be expressed as a pair of data objects (x_i, x_j) with $i \neq j$. A clustering satisfies this constraint, if x_i and x_j are not located in the same cluster. During a preliminary process, *COALA* generates one cannot-link constraint for each pair of objects which are members of the same cluster in C. In order to reach the maximum degree of dissimilarity from C, the alternative S must place as many objects as possible in different clusters that were in the same cluster in C. Although this approach seems plausible, strict adherence can lead to meaningless solutions, as a clustering that maximizes dissimilarity most likely does not comply with the general requirements of clustering, namely similar objects belong to the same cluster while clusters are dissimilar. This means besides being dissimilar, an alternative clustering must also satisfy a certain quality that is, in the case of *COALA*, expressed by the similarity of a pair of objects. The two goals of dissimilarity and quality can be inversely related, for which case *COALA* offers a parameter ω to control the trade-off between both goals.

We illustrate *COALA*'s method of operation using the small example shown in Figure 2.5. In Figure 2.5(a), we can see the starting point of the method: a dataset of 10 objects for which a clustering solutions C exists. The initial object assignments are represented by the data object's shape, showing



(a) Initial clustering. (b) Singleton clusters. (c) Clusters after first merge.

(d) Clusters after 8th merge. (e) Alternative for large ω. (f) Alternative for small ω.

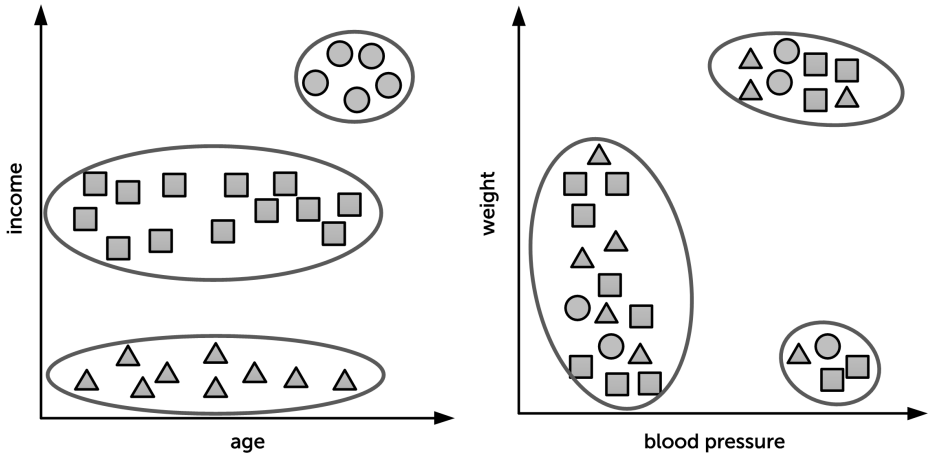
Figure 2.5: COALA example.

that C consists of $k = 2$ clusters, thus the sought after alternative also has two clusters. COALA works in a similar way as the previously introduced hierarchical clustering algorithms, which means each object is initially placed in its own cluster and the clusters are iteratively merged until k is reached. In each iteration two candidate cluster pairs for the merge are identified: one for the quality goal having the smallest distance (d_{qual}) of all pairs and one for the dissimilarity goal having the smallest distance and fulfilling the cannot-link constraints (d_{diss}). The decision between both candidates is based on an inequation: if $d_{qual} \leq \omega \cdot d_{diss}$ the quality pair is merged, else the dissimilarity pair. In Figure 2.5(b) we can see the initial cluster structure as well as the quality and dissimilarity candidates. As d_{qual} is much smaller than d_{diss} a quality merge is made. This procedure is repeated until the clustering only contains 2 clusters. The influence of ω can be described as follows: Choosing a high value for ω favors the quality goal, as the mentioned inequality is fulfilled more often. The alternative clustering shown in Figure 2.5(e) is obtained by using a large ω in our example. In contrast, a small ω prioritizes dissimilarity and would lead to the alternative clustering presented in Figure 2.5(f). Regarding our example, a too strong focus on quality is unfavorable as the obtained alternative and the initial clustering are identical. In this case, the dissimilarity goal must be prioritized to obtain a useful alternative. However, the trade-off between both goals must be adjusted individually for each application scenario.

Further examples for alternative clustering techniques include, but are not limited to: CAMI [14], which utilizes expectation maximization and mutual information to model quality and dissimilarity; or CIB [26] which is based on the information bottleneck principle.

2.2.2 Subspace Clustering

Subspace clustering techniques especially address high-dimensional datasets. The more features a dataset has, the harder it gets to generate clustering solutions that satisfy the goals of intra-cluster similarity and inter-cluster dissimilarity to an equal degree over all dimensions. In fact, clusters can



(a) Wealth subspace. (b) Health subspace.

Figure 2.6: Different subspaces of a dataset.

be observed in different subsets of the feature space. Such combinations of dimensions are called subspaces. By clustering the dataset according to these subspaces, different clusterings are obtained. The interpretation and accordingly the meaning of these clustering solutions depends on the respective dimensions and can differ between subspaces. Let us assume a customer dataset of a health insurance company as an example. It contains a large amount of features describing demographic, financial, and medical data. In Figure 2.6 two possible subspaces of this dataset are depicted. The first subspace contains dimensions that describe the wealth of a customer. A clustering of this subspace is shown in Figure 2.6(a) and consists of three clusters that represent different wealthy customers. Another grouping of customers can be made using the health subspace depicted in Figure 2.6(b). As the relevant features for this subspace are weight and blood pressure, clusters could represent different risk groups regarding diseases of the cardiovascular system. The object shapes in this figure still represent the cluster assignment obtained for the wealth subspace and show the differences between both subspaces from the viewpoint of object similarity. We can see that different meaningful clusterings can exist in different subspaces of high-dimensional datasets. The identification of such clusterings and subspaces is the goal of subspace clustering. One of the first subspace clustering algorithms is *CLIQUE* [1], which detects subspace clusters by dividing the dataspace into a grid of cells having an equal size in each dimension. A cluster is represented by a set of adjacent cells that exceed a specified population threshold and are thus considered as dense. The identification of subspace clusters is done in a bottom-up fashion, based on a monotonicity definition stating that: if a collection of points S is a cluster in a k-dimensional space, then S is also part of a cluster in any $(k-1)$-dimensional projection of this space. Accordingly the algorithm starts by determining the dense cells of each single dimension. By joining these one-dimensional dense cells, two-dimensional cells are generated that are again filtered using the population threshold. In this fashion k-dimensional cells are created from $(k-1)$-dimensional dense cells until no more dense cells can be found and the algorithm stops. A major concern in the area of subspace clustering is combinatory diversity, i.e. the number of possible subspaces to examine. To tackle this issue, approaches like *FIRES* [49] still start by identifying one-dimensional base clusters, but then employs selection techniques based on cluster overlap and similarity, to merge only the most promising candidates. Thus, a much better scalability is achieved. In contrast to CLIQUE, whose a priori mode of operation leads to an exponential runtime complexity, FIRES scales at most quadratic with respect to the number of dimensions. Another representative of this group of algorithms is *DensEst* [54] that uses 2d histograms and their correlations to estimate the potential of candidate subspaces.

2.2.3 Assessment

Regarding the four steps of the clustering process, multi-solution clustering only differs in one. During algorithm selection and configuration the same problems already known from traditional clustering occur. But in contrast, benefits in the adjustment step can be observed. As the user is presented with multiple solutions in the first place, the chances increase that some of them are satisfactory straight away. In such a case the user would select the most appropriate solution and would not need to make actual adjustments to the result.

2.3 ENSEMBLE CLUSTERING

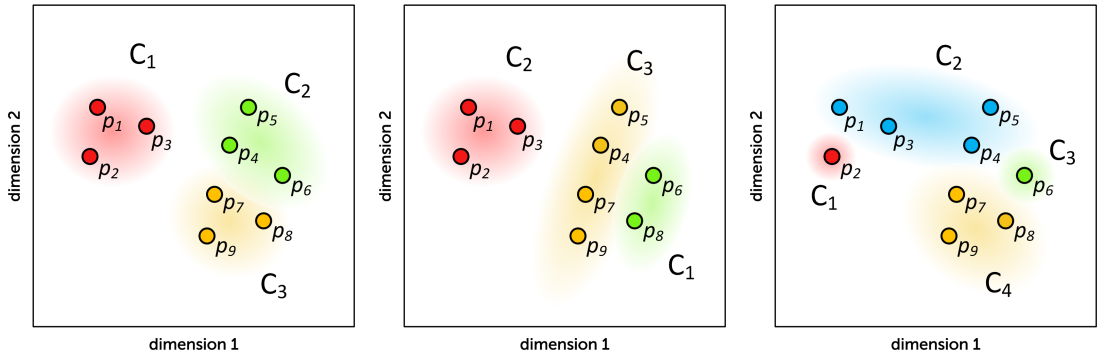
As previously described, traditional clustering often leads to multiple iterations in which different parameters or algorithms are tried until a satisfactory result is obtained. This practice implicitly generates multiple clustering solutions for the analyzed dataset. The concept of *Ensemble clustering* takes up this characteristic and utilizes it explicitly, whereas the mode of application changes to *multiple* algorithms that are executed using *multiple* sets of parameters, generating *multiple* clustering solutions that are combined into *one* final robust clustering.

This utilization of different traditional clustering algorithms with different parameter values aims to tackle the problem of some algorithms or parameterizations failing to work with certain datasets. The set of these multiple clusterings is called *ensemble*, while the final clustering solution generated from it is called *consensus*-clustering. Therefore, this approach is also called consensus-clustering. Approaches of this class can be divided into: *pairwise-similarity* approaches and approaches based on *cluster-labels*. At this point we focus on the pairwise-similarity approaches, as it is more relevant for our work.

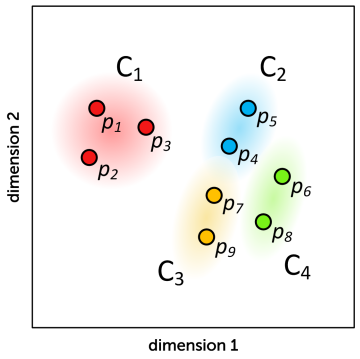
2.3.1 Pairwise-Similarities

In order to generate a single consensus-clustering from an ensemble, a so-called consensus function is needed. This function uses the information regarding cluster assignment of all ensemble members and incorporates them into a new clustering. Algorithms working on the basis of pairwise similarities, model the cluster assignments by evaluating the grouping of each object-pair over the whole ensemble [25, 62]. There are two cases of pairwise similarity: (i) a pair objects is part of the *same* cluster or (ii) a pair of objects is part of *different* clusters. For each clustering of the ensemble, these similarities are represented in the form of a so-called co-association matrix.

Let us assume the small example of a clustering-ensemble with four clusterings for a dataset consisting of 10 objects, as shown in Figure 2.7. All clusterings differ in number of clusters or cluster composition as they are generated using, e.g. different parameters. In Figure 2.7(d), the *local* co-association matrix for clustering C_4 is shown. A cell containing 1 shows that the respective pair of objects is located in the same cluster, e.g. (p_4, p_5), while a 0 indicates that an object pair is assigned to different clusters. For the generation of the consensus clustering, at first a global co-association matrix is built by adding up all local matrices and then normalizing each cell using the ensemble size. Thus, the global co-association matrix contains the relative frequency with which each pair of objects is located in the same cluster. For our example, the global co-association is depicted in Figure 2.8(a), as the ensemble contains four clusterings, the resulting values are multiples of one quarter. Based on this matrix, different consensus functions can be employed to extract the final solution.



(a) Clustering C_1. (b) Clustering C_2. (c) Clustering C_3.



(d) Clustering C_4.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	**1**	**1**	**1**	0	0	0	0	0	0
p_2	-	**1**	**1**	0	0	0	0	0	0
p_3	-	-	**1**	0	0	0	0	0	0
p_4	-	-	-	**1**	**1**	0	0	0	0
p_5	-	-	-	-	**1**	0	0	0	0
p_6	-	-	-	-	-	**1**	0	**1**	0
p_7	-	-	-	-	-	-	**1**	0	**1**
p_8	-	-	-	-	-	-	-	**1**	0
p_9	-	-	-	-	-	-	-	-	**1**

(e) Co-association for C_4.

Figure 2.7: An Example Clustering-Ensemble.

For our example, we use a very simple function based on [25], which aims to generate a consensus-clustering that shows minimal dissimilarities to all clusterings of the ensemble in terms of pairwise similarities. This basically means that if a pair of objects is located in the same cluster in the majority of the ensemble it should also be part of the same cluster in the consensus solution. Vice versa, this also holds for object pairs mostly located in different clusters. To achieve these goals, we simply remove all cells from the co-association matrix that contain a value smaller than 0.5 and use the remaining cells to generate the clustering. In Figure 2.8(a), the cells that fulfill the filter requirement are highlighted by a bold outline. These cells show that (p_1, p_2, p_3), (p_4, p_5) and (p_6, p_7, p_8, p_9) are members of the same clusters in at least fifty percent of the ensemble, thus forming the clusters of the consensus clustering depicted in Figure 2.8(b).

An alternative way to generate the consensus-clustering is to interpret the co-association matrix as an edge-weighted graph, use an algorithm for graph-partitioning like METIS [44, 43] to cut edges with a weight smaller 0.5 and build the consensus clusters from the disconnected subgraphs. Further approaches and examples of consensus functions can be found in [25, 62].

2.3.2 Cluster-Labels

Besides the pairwise-similarity approaches, there are other techniques which are only working with the cluster labels provided by the ensemble. These techniques are often less time-consuming as they do not generate co-association matrices, thus saving the necessary quadratic complexity. There exist various algorithms in this class [38, 62], e.g. *HGPA* [62] which works on a hypergraph consisting of vertices that represent the objects of the dataset and hyperedges that represent clusters. Contrary

to an edge that connects two vertices, a hyperedge can connect an arbitrary number of vertices. In HGPA, such a hyperedge connects all members of a cluster. To extract a consensus solution from the hypergraph, the HMETIS partitioning algorithm [41] is used to cut the minimal number of hyperedges necessary to break the hypergraph down into disconnected components that eventually form the consensus clusters.

Another example for ensemble-clustering using cluster-labels is *Ensemble-Merging* [38]. This method assumes that each clustering of the ensemble has the same number of clusters k and that each of these clusters is represented by a centroid. Through grouping of similar centroids of the ensemble, k global centroids are generated which are then used during a final assignment of dataobjects that results in the consensus clustering. Further techniques—e.g. probabilistic approaches—can be found in [11, 51].

2.3.3 Assessment

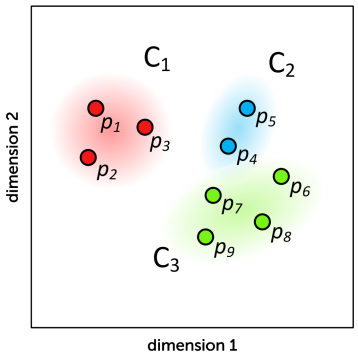
Ensemble clustering features some interesting benefits for the algorithm selection and configuration steps. One the one hand, the use of multiple algorithms and parameter values relieves users from the necessity of finding the single optimal algorithm-parameter combination. On the other hand, ensemble clustering determines more robust results and thus leads to an expanded applicability. Unfortunately, these benefits come with a huge drawback concerning adjustments. If adjustments are necessary due to an unsatisfactory result, the user not only has to decide on switching one algorithm or one set of parameters, but has to configure a whole set of algorithms including new choices like ensemble size and algorithm composition.

2.4 CLUSTERING INTERPRETATION

While the previous sections dealt mostly with the algorithmic side of the clustering practice, this section focuses on the ways of communication between the clustering algorithms and the user. Basically, there are two directions for this communication: (i) from algorithm to user, which is covered by the result interpretation step and (ii) from user to algorithm in the form of the adjustment step. We discuss both steps in combination, as they fundamentally depend on each other. During result

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	4/4	3/4	4/4	1/4	1/4	0	0	0	0
p_2	-	4/4	3/4	0	0	0	0	0	0
p_3	-	-	4/4	1/4	1/4	0	0	0	0
p_4	-	-	-	4/4	4/4	1/4	1/4	0	1/4
p_5	-	-	-	-	4/4	1/4	1/4	0	1/4
p_6	-	-	-	-	-	4/4	1/4	3/4	1/4
p_7	-	-	-	-	-	-	4/4	2/4	4/4
p_8	-	-	-	-	-	-	-	4/4	2/4
p_9	-	-	-	-	-	-	-	-	4/4

(a) Global co-association.



(b) Consensus Clustering.

Figure 2.8: An Example Consensus Clustering.

interpretation, the user evaluates and decides whether a clustering solution is satisfying or not. In the latter case, it is necessary to identify what makes the clustering unsatisfactory and derive what must be done to improve the result, i.e. what adjustments must be made. We start our discussion with available techniques for result interpretation, which can be divided into the two main areas of *quality measures* and *visualization*.

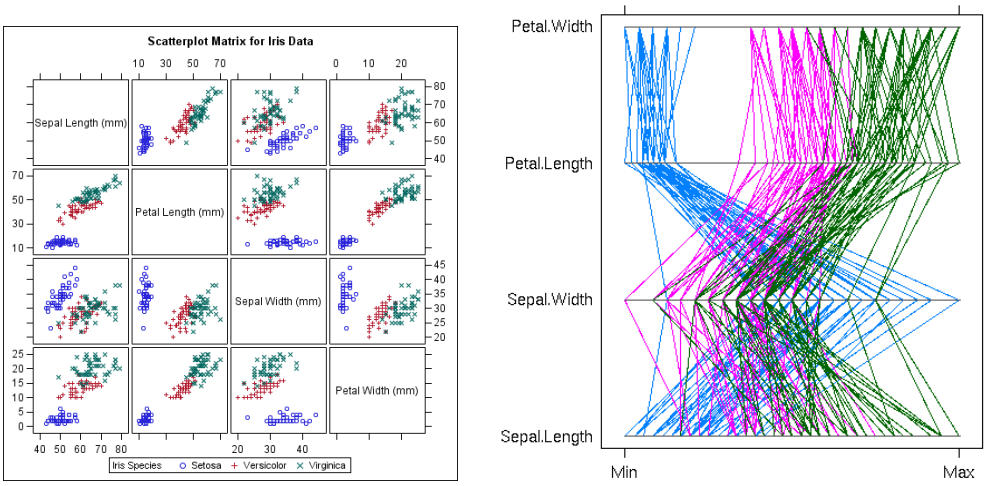
2.4.1 Quality Measures

The methods of this area try to answer the question 'How good is the obtained clustering?'. Answering this question is nearly impossible as there is no universally valid definition of clustering quality. As a result of this situation, multiple quality metrics exist [16, 18, 58]. One extremely popular approach to clustering quality that is often used in publications to evaluate the performance of the proposed algorithm, is comparison to a known solution. This known solution is usually built by application-domain experts, who manually label the data, and is considered optimal. The quality of a clustering solution is then measured by quantifying the deviation from this gold standard, e.g. by using the *Rand Index* [58]. Obviously this approach is not usable in real-world applications, as the optimal clustering of a dataset is typically unknown, which is the fundamental reason to use clustering in the first place. Therefore, most quality measures are based on the general goals of clustering, namely high intra-cluster similarity and high inter-cluster dissimilarity. Typically, a quality measure models these two goals and uses the ratio between them to express quality. Examples for such quality measures are *Dunns Index* [18] or the *Davis Bouldin Index* [16]. Each of these methods uses an individual definition of clustering quality, thus their expressiveness depends on the clustered data, the employed algorithm and parameters and the application scenario in general. This lack of universality means that quality measures can only be applied for an absolute result interpretation in well-known scenarios and in combination with application domain knowledge. Otherwise, they can only be used for orientation or the relative comparison of clustering results. In addition, the coarse granularity of quality measures makes them inappropriate for the derivation of adjustments, as typically whole clusterings or clusters are mapped to a single numerical value, which means information concerning the actual cluster structures is lost.

2.4.2 Visualization

The human being has exceptional visual perception capabilities that can be addressed for result interpretation by employing visualization techniques. Via graphical presentation of the dataset and the obtained cluster assignments it is possible to communicate fine grained information about the identified structures to the user. Displaying the raw dataset and its assignment to the identified groups allows the user the subjective interpretation of the obtained result without the bias added by the specific definitions of quality measures. As the user is often an expert of the application domain, he/she possesses background-knowledge that permits the evaluation of the clustering solution.

We already used a visualization technique in Section 2.1 regarding traditional clustering, by depicting clustering results as scatterplots, e.g. in Figure 2.2. These plots show each data object, its location in the two-dimensional space of our example dataset and its cluster assignment. Although scatterplots are a very convenient technique for the interpretation of our small examples, their usefulness suffers when it comes to large-scale datasets. Data-driven techniques like scatterplots always display all objects of a dataset, which can be a problem for high-volume data as there may be too many objects for the available display space leading to occlusions and other unfavorable effects. Besides that, the biggest problem is the presentation of an arbitrary number of dimensions on a two-dimensional medium like a computer monitor or this book. In addition to these technical problems, most humans



(a) scatterplot matrix

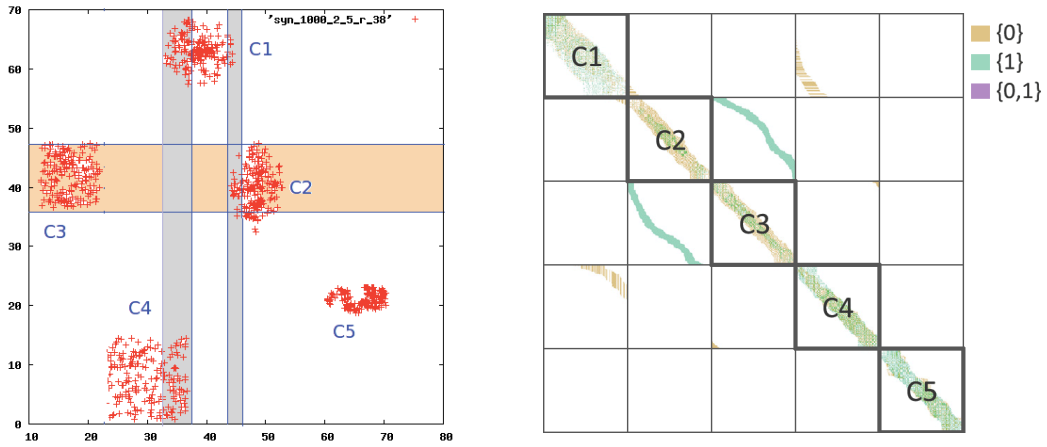
(b) parallel coordinates

Figure 2.9: Visualizations of the Iris dataset.

have problems with grasping high-dimensional spaces as our physical universe just spans three dimensions. Two general approaches for displaying clusterings of high-dimensional data are depicted in Figure 2.9 using the four-dimensional *Iris* dataset [24] as example (cluster assignment indicated by color). In Figure 2.9(a), a so-called *scatterplot matrix* is shown. This technique displays a scatterplot for each two-dimensional projection of the data's attribute space and arranges these plots according to the attribute pairings. While the simplicity of this approach is beneficial, there are significant drawbacks: (i) the matrix size scales quadratic with respect to the number of dimensions and (ii) only two-dimensional plots are used, making it hard to recognize structures with higher dimensionality.

A technique which tries to address these drawbacks are *parallel coordinates* [39] which are shown in Figure 2.9(b). The horizontal lines in this plot represent the dimensions of the dataset while each vertical line represents an object of the dataset. The vertical object-line intersects each horizontal dimension-line at a position that represents the location of the object in the range of the respective dimension. While this visualization technique allows the identification of high-dimensional structures, it tends to get confusing for high-volume data, as the number of vertical lines increases.

Besides such general visualization approaches, there are also methods designed for displaying clusters in high-dimensional spaces. An example for this group of methods is the *Heidi Matrix* [64] visualization that shows clusters and their overlap in different subspaces of the data, using k-nearest-neighbor (k-NN) relationships. To generate the Heidi Matrix, first all possible subspaces of the dataset are identified. Next, the k-NN's are computed for each object of the dataset in each subspace. Subsequently a bitvector whose length corresponds to the number of subspaces is generated for each object pair (p, q). If p and q are k-NN's in a certain subspace, its respective bit is set to 1. All object pairs are grouped and ordered depending on their cluster assignment. Finally, each object pair is visualized as a pixel, whose color is defined by the previously generated bitvector. To illustrate the interpretation of a Heidi Matrix we use the example shown in Figure 2.10. On the left we can see a scatterplot of a two-dimensional dataset with five clusters, on the right the corresponding Heidi image is shown. The blocks along the main diagonal are called intra-cluster blocks while the remaining blocks are called inter-cluster blocks and show the overlap of their adjacent clusters. As an example lets look at clusters $C2$ and $C3$ in Figure 2.10(a) that overlap along the vertical dimension 1, whereas this overlap is shown by a colored ribbon in the inter-cluster blocks between $C2$ and $C3$. The ribbon spans the



(a) Example scatterplot

(b) Heidi matrix

Figure 2.10: Heidi Matrix Visualization example.

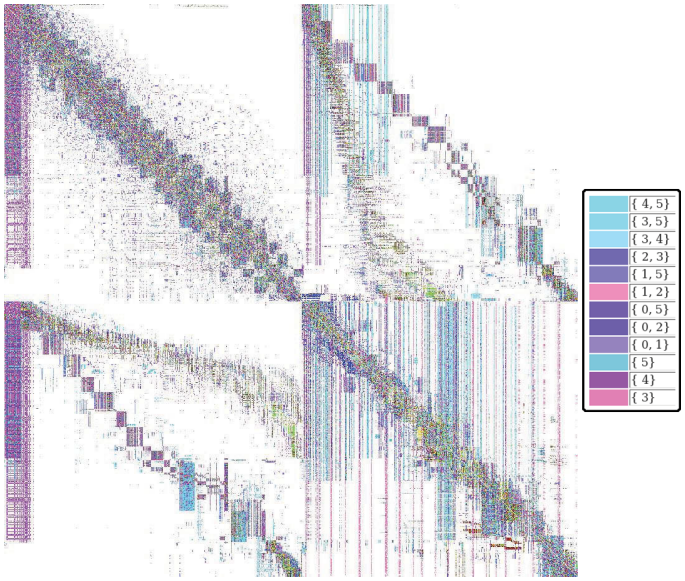


Figure 2.11: Heidi Matrix for Migrant Boats dataset.

whole width and height of the block, showing that cluster overlap is complete. The color of the ribbon is assigned to the dimension 1 in the legend, thus indicating the subspace in which the overlap takes place. We can also observe a small ocher ribbon between $C1$ and $C4$ showing that both clusters partially overlap in dimension 0, i.e. the x-axis of the scatterplot. Although the Heidi Matrix forms an interesting visualization approach for clusterings, its visual presentation is not easy to understand and tends to get more complicated for datasets of larger scale as shown in Figure 2.11 for the migrant boat data set of VAST 2008 Data Challenge.

Again, the techniques described here only present a small extract of all available methods. A short overview on the diversity of visualization techniques is given in [47]. There also exist different variations of the proposed techniques, which are designed with optimization or specialization in mind. For example, the authors of [66] enhance the parallel coordinates design by integrating multidimensional scaling plots and changing the curvature of lines to reduce cluttering. Furthermore, parallel coordinates and scatterplots are combined with other visualizations in [63] for the specific task of finding and visualizing subspaces, which allows the interpretation of alternative clusterings.

2.4.3 Adjustments

In contrast to the previous steps, little can be said regarding approaches to the adjustment of clusterings. In general, it is accepted that adjustments are made by changing the clustering algorithm or the algorithm-specific parameters. The problem with this approach is that the user must explicitly know how the clustering algorithm works and how its work is influenced by each parameter. Otherwise, it is not possible to achieve the intended adjustment. An opposite approach to this very technical low-level way of adjustment is proposed in [7]. In their highly theoretical work the authors propose two high-level parameters *split* and *merge*, with which the user can adjust a clustering. They prove that each clustering C can be adjusted into an arbitrary Clustering C' just by the subsequent splitting and merging of clusters and present some observations regarding upper bounds for the necessary number of split and merge steps. This approach can only be seen as a proof of concept as the assumed setting was very limited, i.e. the dataset was of small volume and only one-dimensional while the known optimal solution was presented to the user in order to allow the selection of the appropriate adjustment steps. Nonetheless, this approach shows a way of adjusting clusterings that is very interesting for non-expert users.

2.4.4 Assessment

Similar to the algorithm-centered steps of selection and configuration the area of result interpretation offers a variety of techniques. Quality measures are easy to interpret, but lack a universally valid notion of quality. Again, this makes selection of a technique challenging, as the choice of an unsuitable quality measure can effectively prevent that a satisfying result is recognized as such. Furthermore, quality measures compress all structural information of a clustering into a single numerical value, which means detailed information is lost. Visualizations do not suffer from this problem as they do not explicitly express quality, but present the whole clustering and leave its interpretation completely to the user. While this makes selection a less critical problem, there are other drawbacks. Visualizations are typically data-driven and communicate raw information on the dataset and clustering. As volume and dimensionality of the data increase, they communicate more information than the user can process/comprehend, which makes interpretation harder or even impossible.

2.5 CONSEQUENCES

This chapter gave a brief overview of the current clustering practice and its four basic steps. We described several available approaches with their associated methods and outlined the existing limitations regarding applicability and usability for non-expert users. In general, we can state that the combination of a lack of background-knowledge and the vast amount of choices for clustering algorithms, parameters and interpretation methods is responsible for the main problems concerning the creation of a clustering. Thus, the described clustering practice is carried out in an iterative fashion that includes—more or less random—variations during each step which effectively leads to a 'trial-and-error' mode of operation. Needless to say, this has a negative influence on result quality, runtime/number of iterations and ultimately user satisfaction. In summary, we can state that four major challenges arise from the contemporary clustering practice:

- **Fragmentation & Rigidity:** Although numerous algorithms are available, they are typically specialized and not versatile as they capture only singular facets of the data.

- **Decoupled Workflow:** The execution of the four necessary steps is not orchestrated. Each step forms a separate domain and is not considered as part of a bigger picture.
- **Challenging Interaction:** Available approaches for the communication of the result are either too compact or too vast, which makes it hard to understand and adjust a clustering. In addition, the implementation of adjustments requires knowledge of the inner workings of an algorithm, as parameters do not directly influence the result, but its creation.
- **Absence of Guidance:** Typically, users are left alone with decisions, reasoning and algorithm control. This prevents the meaningful application of clustering, especially for amateur users.

To tackle these challenges, it is necessary to completely reconsider the current practice of clustering. The individual steps must be tightly coupled and clustering itself must be evolved into a process that integrates existing approaches and covers all tasks regarding clustering creation and user interaction in an end-to-end fashion. By addressing usability and algorithms in a holistic fashion, such a process allows the versatile and meaningful application of clustering regardless of user experience.

PRINCIPLES OF A VERSATILE CLUSTERING PROCESS

3.1 Handling of Algorithms

3.2 Interacting with the Clustering

3.3 Iterative Refinement

3.4 Summary

In the previous chapter we assessed the state of contemporary clustering and its challenges for inexperienced users. Most of these challenges arise from a nearly complete absence of structure for the actual practice of clustering. The four steps introduced in Chapter 2 merely describe a generic course of action that leads to a clustering result. They state *what* needs to be done, but give no information on *how* to do it. Thus, the execution of clustering becomes a serious challenge for any user. Even though the 4-step practice is very generic and can be flexibly applied in many scenarios, its realization is often very specific and rigid. Typically each step is implemented by choosing an existing method or developing a custom solution for the task at hand.

The goal of this thesis is to change this state. We want to substantiate the abstract 4-step model and develop a structured process template that can be realized in a system and provides users with a default procedure for clustering. Thus, freeing them from most of the low-level and scenario-specific tasks that are normally necessary in order to execute clustering. For that purpose, we need to specify tasks and create structures for the integration of algorithm selection and configuration as well as result interpretation and adjustment. The guiding principle during development is to find a compromise between rigid predefinition and flexible adaptation. Our clustering process implements said principle through a fixed modus operandi that incorporates versatile components. Users can use a steady and familiar working environment when performing a clustering, and can still adapt to different application scenarios.

The template for our versatile clustering process has to implement two major tasks: The handling of algorithms and the interaction between user and clustering. Algorithm handling must take care of clustering creation and has to ensure broad applicability. This task is implemented in a component that manages the variety of clustering algorithms and their related aspects. Similarly the interaction between user and clustering is realized in a component that communicates clustering results and offers a universally valid set of controls for adjustment of the clustering. Both major tasks and the components that realize them, are integrated into an iterative process model that will enable the stable and user-friendly execution of clustering. In the following chapter, we will describe the different tasks in more detail and derive the requirements that their implementation must satisfy.

3.1 HANDLING OF ALGORITHMS

The first and foremost task our process needs to execute is the actual creation of a clustering. For this, it is necessary to tackle algorithm selection and configuration. As both affect clustering creation, we combine them into a single component that will manage all algorithm related aspects. From our assessment of contemporary clustering, we know that selecting a configuration of clustering algorithms is challenging due to the quantity of available methods and the different degrees of application-specialization. Since we aim for versatility, the algorithm management must be able to deploy arbitrary clustering algorithms and handle their configuration. Besides the challenges arising from the variety of clustering methods, we already mentioned the shortcomings of single-execution clustering. Typically, a single algorithm execution with a single set of parameters will not result in an optimal clustering result, which is why the summarization of different results is desirable. Based on these needs, we state the first task that the algorithm management must tackle as: *integration* of arbitrary clustering algorithms, different parameterizations and multiple solutions into a clustering result.

While integration mainly focuses on the output of the algorithm management, there is also an input that must be considered. This input is a set of algorithms and parameters that is either initially chosen according to the characteristics of the data / application-scenario, or results from modifications made

during the result adjustment step. Since the algorithm management integrates different algorithms, it would be necessary to provide this input with multiple algorithm-specific formats. This must be prevented as it represents exactly those low-level tasks we want to keep away from the user. In order to reach our goal of providing a universally valid set of controls, the second task of the algorithm management can be defined as: provision of a *control interface* that is able to implement initial configurations and subsequent modifications of a set of algorithms.

To create such an interface, we need to understand how various algorithms work and how they differ. This requires a way to abstract and generalize from different methods in order to find similarities. Understanding the workings of clustering is mandatory when algorithms are selected or modified for a particular dataset. Currently, the presentation of clustering algorithms in research papers and books is either highly descriptive or implementation specific. Both levels are not useful to us as the former is too vague while the latter is too detailed. As a solution to this problem, we state the third task of the algorithm management as: provision of a facility for *algorithm description* that is a compromise between the conceptual and the implementation level.

3.2 INTERACTING WITH THE CLUSTERING

Besides the creation of a clustering result, our process needs to handle result interpretation and adjustment. During result interpretation, the clustering result is communicated to the user, who interprets it, decides if and how the result should be changed, and derives modifications of algorithms / parameters that are forwarded during the result adjustment step. These two steps take care of the whole interaction between clustering procedure and user, which is why we combine them into a single component, responsible for interacting with the clustering.

Our assessment of contemporary clustering showed that there are two basic approaches for result interpretation: quality measures and visualizations. Quality measures have a pretty coarse granularity that omits information regarding cluster structures. Furthermore, their use is limited to relative comparisons between clusterings, as there is no universally valid definition of clustering quality that would allow an absolute rating. In contrast, visualizations offer structural information in a fine granularity, which can lead to overwhelmingly complex displays of information that cannot be interpreted effectively anymore. Both approaches have individual benefits: Quality measures summarize clustering characteristics, while visualizations present structures inside the data very detailed. As there are also serious drawbacks, choosing an existing quality measure or visualization technique to interact with the clustering is not an option. In our process we need a compromise between the coarse and fine-grained display of information. Therefore, a novel *hybrid-visualization* must be developed, whose subject will neither be the single object and its location inside the feature space, nor the clustering as a whole. The focus will be on the cluster-level in between those extremes. We want the hybrid-visualization to be visually simple and readable, which means that complex high-dimensional displays must be prevented. This requires the development of measures that characterize identified structures and are unaffected by dimensionality. As this hybrid-visualization concept forms the core for the whole interaction with the clustering, we denote the respective process component as visual-interactive interface.

Besides the visualization concept for result interpretation, it is necessary to develop a way of result adjustment that can be incorporated into the hybrid-visualization. We already stated that the algorithm management integrates a variety of algorithms, which would require the user to provide result adjustments in an algorithm-specific way. This is a very demanding and specific task for the user that we want to ease greatly. Therefore, the second task of the visual-interactive interface is to provide a compact and default set of *high-level feedback* operations that allow result adjustments regardless of the underlying clustering algorithms. Instead of parameters for a function, these high-level feedback operations should describe the actual change the user intends to realize in the clustering result.

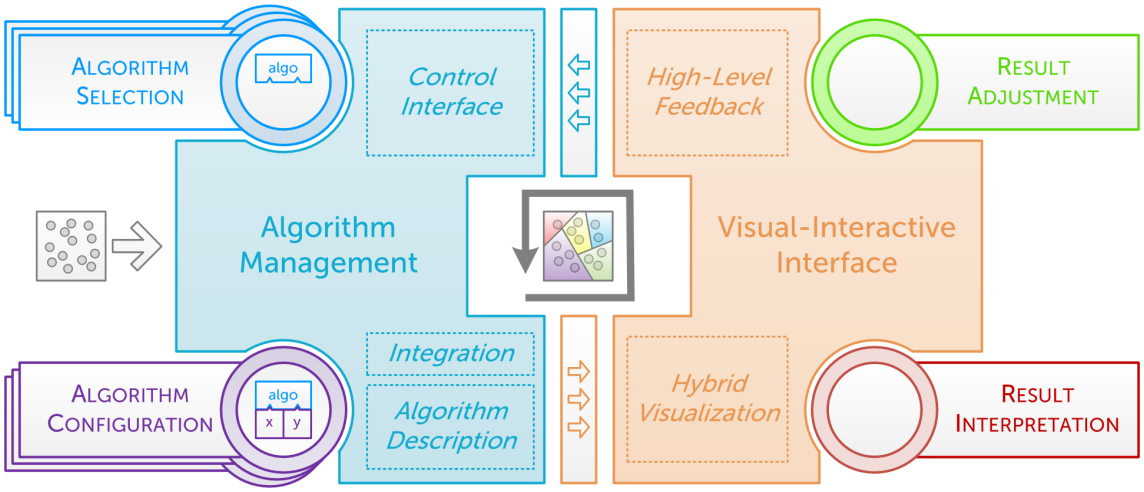


Figure 3.1: Template of the versatile clustering process.

3.3 ITERATIVE REFINEMENT

At the beginning of this chapter we mentioned that our process template should realize a default modus operandi for clustering. The modus operandi we know from contemporary clustering practice is build around the concept of single execution, thus its scope is limited to the one-time creation of a single clustering result for a set of data. Due to this straightforward procedure, adjustments to the result can only be made by executing an additional iteration with modified parameters and/or algorithm. Modification of a result is effectively realized by discarding the existing result and creating a completely new one. This makes adjustments unintuitive and impractical for the user. Assume the following example: during result interpretation a user concludes that only a single cluster of the result needs modification. Based on this, parameters are derived that hopefully influence the clustering algorithm to yield a result that implements the intended changes. This means that local changes are implemented indirectly through a global rebuilding, which always poses the risk that undesired changes are introduced into formerly satisfying parts of the result.

With our process we want a more direct and local character for the result adjustments. For that purpose, we define the modus operandi of our process as *iterative refinement* that allows the direct application of local changes without rebuilding the whole clustering result. In contrast to the repeated rebuilding, this approach is more continuous and enables users to adjust clustering results in a step-by-step fashion until they are satisfied with the outcome. This principle has a fundamental impact on the development of our clustering process. Figure 3.1 depicts the assembly of the described component into our clustering process. The algorithm management and the visual-interactive interface are connected via interfaces that pass on information about the clustering result and adjustments. At the center, we find the iterative refinement that is used to successively adjust the clustering result to the user's liking by utilizing the functions provided by both major components.

3.4 SUMMARY

In this chapter, we introduced the basic concept of a versatile clustering process. We created a template by defining components and identifying the tasks that they need to perform in order to realize our process idea. Two major components were described: the algorithm management that handles all

Figure 3.2: Screenshot of a prototypical process implementation.

aspects of clustering creation and the visual-interactive interface that provides a way for interaction between the user and the clustering. All components are embedded into an iterative process model that enables the step-wise refinement of a clustering result instead of its complete recalculation. Until now we only provided a template and some requirements. In the following chapters we will introduce algorithms and components for the actual realization of these concepts. To give you a glimpse of what to expect, Figure 3.2 shows a screenshot of a prototypical implementation of our process template that was assembled with our proposed methods and techniques.

ALGORITHM MANAGEMENT

4.1 Algorithm Description

4.2 Integration

4.3 Control Interface

4.4 Summary

The genesis of a clustering is a complex task that involves the selection, configuration, execution, creation, and adaptation of different algorithms. Our proposed algorithm management is the first essential component of our clustering process and handles all aspects regarding algorithms and clustering creation. In doing so, it does not represent a new technique for actual clustering, but provides a framework that eases and improves the application of existing methods. As outlined before, the algorithm management has to fulfill three main tasks.

The first one is to define a way of algorithm description that can be used to understand and compare clustering techniques. Basically, knowledge of the inner workings of a variety of clustering algorithms is what experience in the area of clustering is mostly about. For our algorithm management we develop a modular approach for description that consists of a basic template and a set of building blocks. Different algorithms are described by fitting different combinations of blocks into the template. With this, algorithms can be compared by means of their shared building blocks and modified by exchanging or adding blocks.

Second, integration handles the variety of available algorithms and combines the different clustering results they produce into a single solution, which constitutes the starting point for iterative refinement. Integration is carried out in two stages. At first, a set of clusterings is produced by using multiple traditional clustering techniques and parameterizations. In a second step, the structural information provided by the members of this set is aggregated into the initial clustering. For this, the algorithm management utilizes specific techniques of which we describe two.

The production and integration of clustering results are vital parts of our clustering process, which means users must be able to control them. Unfortunately, steps like algorithm selection and configuration constitute considerable challenges for the user, which puts us into a dilemma. On the one hand, algorithm management must relieve users from complex interactions and keep away much of the technical details of clustering. On the other hand, users must be in charge of the clustering process. This issue is addressed in the third and final task of the algorithm management, which is to provide an understandable control interface that is easy to use. Our provided interface is structured into multiple levels and contains understandable options for influencing the most important parts of clustering production and integration.

Please note that our algorithm management basically describes a conceptual template that defines certain tasks. This means, all subsequently introduced methods constitute options for fulfilling these tasks and realizing the whole component, but no exclusive or final solution.

Parts of the material in this chapter have been developed jointly with Wolfgang Lehner, Dirk Habich, Markus Dumat and Peter Volk. Section 4.2.1 is based on [34] and [33], which are both available via Springerlink. The algorithm proposed in Section 4.2.2 was published in [32] and [29].

4.1 ALGORITHM DESCRIPTION

In order to understand an unknown clustering algorithm we naturally turn to its description. Typically algorithm descriptions in literature contain a verbal part that illustrates an algorithms method of working by using the author's words. This verbal part is often accompanied by an implementation description based on the syntax of a specific programming language. Both parts have benefits and drawbacks: The verbal description excels when it comes to communicating the idea of an algorithm and its mode of operation, but utterly depends on the author's narrative skills. These can vary, which means that verbal descriptions are not consistent and can change for each individual author. In contrast, implementation descriptions that use the same programming language offer a certain degree of

consistency. This consistency is lost if different programming languages are used, which is a drawback of this description style. In addition, implementation descriptions often contain parts that do not directly concern the algorithm, but its effective implementation on a specific platform, e.g. data-structures or procedure calls. Thus, two descriptions of the same algorithm in the same programming language can differ if different data-structures are used. In summary we can state: While there are ways to create understandable algorithm descriptions, existing approaches are not consistent as they lack a common vocabulary.

There are two main reasons to seek consistency: It eases understanding and allows the comparison of algorithms on the basis of their descriptions. Especially the latter is interesting when an algorithm is selected for clustering. In contemporary clustering practice a rough categorization was established in an attempt to provide a way for high-level description and algorithm comparison. Different classes of clustering algorithms have been introduced to organize the multitude of existing methods, e.g. density-based and hierarchical algorithms. Unfortunately these classes are neither explicitly defined nor clearly differentiated from each other, which makes categorization very vague. The goal of this categorization is to identify similar clustering methods and to aid algorithm selection by providing statements like: algorithms of class X are suitable for datasets showing characteristic Y. However, due to the ambiguity of categorization such statements are generally imprecise.

All these issues are relevant to our clustering process. In order to handle the multitude of available clustering algorithms, we need an universal option for their description. A unified and consistent view on clustering algorithms will enable users to better understand their functionality. In the course of this section we present a novel approach to algorithm description. Starting from the archetype of clustering algorithms, we derive definitions and building blocks, which will constitute a consistent vocabulary for description.

4.1.1 Extracting the Essentials of Data Clustering

For our approach, we concentrate on the core of traditional clustering, i.e. the grouping of objects. We ignore the areas of pre-processing and post-processing that cover tasks like feature selection, data cleansing or noise reduction. Our goal is to disassemble clustering algorithms in order to identify their essential components. We begin by assuming the following general definition of data clustering:

> *"Data clustering is the partitioning of a set of points into groups—so-called clusters—in a way that similar points are put in the same cluster, while dissimilar points are located in different clusters."*

According to this definition, every clustering algorithm can be described as: "An algorithm that partitions a set of points into clusters, so that points in the same cluster are similar, while points located in different clusters are dissimilar." In this extremely abstract description, we can identify certain fundamental tasks that have to be performed in order to generate a clustering. The first fundamental task is pointed out in the first part as the grouping of points. In the latter part we find that this grouping is not performed at random, but follows some constraints. As points are not arranged arbitrarily, the second task of a clustering algorithm is to explicitly choose those points that should be grouped together. The criterion for the selection of the points for a group is their similarity. That means, an algorithm needs to measure the similarity of the points, which states its third task.

The three tasks we identified are interdependent and must be performed in sequence, as grouping cannot be done without choosing points or measuring similarity in advance. From the identified tasks and their sequential dependence we derive the *phases* of a clustering algorithm that form the frame for further building blocks:

- **evaluation** - During this first phase the similarity between all points or between all points and a set of references is measured.
- **selection** - In this phase the points that are eligible to be grouped together are selected according to the algorithms specification.
- **association** - During this phase the previously chosen points are associated with a cluster.
- **optimization** - This phase contains support tasks for the previous phases like parameter adjustments, iteration control or re-calculation of centroids.

The fourth phase is not directly derived from the clustering definition above, but originates from analyzing existing algorithms that often feature optimization tasks like parameter adjustment or target function maximization, which lead to multiple iterations of the first three phases. The optimization phase is mainly concerned with improving the result generated by the preceding phases. Therefore we consider it as optional, while the other phases are mandatory. Every clustering algorithm *requires* an evaluation, selection and association phase in order to complete the three fundamental clustering tasks identified earlier. Thus, we refer to this three-phase sequence as the *core* of a clustering algorithm.

The defined phases of a clustering algorithm specify a frame that needs to be fitted with additional building blocks in order to describe an algorithm. To find these blocks, we analyze each phase and look for its basic elements. We start with the evaluation phase, which is responsible for similarity measurement. In order to determine the similarity of two objects, a dedicated function is needed. In data clustering there are two general approaches to similarity measurement: similarity functions and distance functions. While the former express the degree of similarity the latter describe the amount of dissimilarity between objects. For clustering both can be used on the assumption that low dissimilarity corresponds to high similarity and vice versa. Since most of the clustering algorithms we encountered employ distance measures we assume that similarity is expressed via distances for the remainder of this chapter. With this we can define the *distance measure* as the first basic element of the evaluation phase. In general, a distance measure uses at least two inputs and produces one output value. The first input is made up of the *points* which are to be clustered. The second input contains the objects to which the first input is compared to in terms of similarity. Hence, we use the term *references* for it. References can occur in different forms. On the one hand, there exist algorithms like DBSCAN [21] that calculate all point-to-point distances and effectively use points also as references. On the other hand, approaches like k-means [50] employ a special set of representatives/centroids as second input for distance computation. After thorough research we found that references can appear in three forms: a clone of *points*, a subset of *points* or a set of objects that is disjoint from *points* but shares its domain. As the distance measure is a basic element of the evaluation phase, so are its two inputs. Naturally the next basic element has to be the output of the distance measure as it contains the similarity information that is needed in successive phases. We label this element as *distances*, which leaves us with a total of four basic elements for this phase. These allow a more detailed definition of the evaluation task. In essence, during evaluation the distance measure is used to create a relation between points and references that represents their similarity and is explicitly expressed in the form of distances. It is important to mention that besides the distance values itself, evaluation also provides relational information in the form of *point-distance-reference* triples.

Next, we regard the selection phase. It uses the point-distance-reference triples generated by evaluation as an input and chooses the points which are eligible to be grouped together. According to our initial clustering definition it is desired that only those points which are similar overcome the selection process and enter association. In order to extract these points, it is necessary to define the requirements that are needed to acquire the status 'similar'. Furthermore, all incoming points must

be tested on whether they fulfill said requirements or not. This is achieved by using *filters*, which are the basic element of this phase. Basically the selection phase utilizes a filter cascade to test each incoming point-distance-reference triple and only passes on those that comply.

After selection, we examine the association phase, which is the final phase of the mandatory *core* of a clustering algorithm. The input of association are the point-distance-reference triples that passed selection and must now be grouped in order to create a clustering. This *clustering* forms the output of this phase and is one of its basic elements. Like the distances from evaluation, a clustering is made up of relations, i.e. the affiliation between points and clusters. To create these, the association phase has to transform point-distance-reference triples into point-cluster tuples. Transformation is handled by the next basic element, which we denote as *association function*. For some algorithms a simple substitution is already sufficient to create a clustering, e.g. the association function of k-means assumes the references as clusters and effectively creates point-cluster tuples by removing the distance from the incoming point-distance-reference triples. However, not all clustering techniques work that way. Let us regard DBSCAN, where at first a core-object is associated with its neighborhood–by creating point-reference tuples–before the actual clusters are formed on the basis of overlapping neighborhoods. A direct creation of point-cluster tuples like in k-means is not possible in DBSCAN, because the clusters are not known in advance. Considering this issue, we introduce the basic element *adjacencies* as a stopover between the association function and the final clustering. With it, we describe the association phase as follows: incoming point-distance-reference triples are transformed by the association function into adjacencies, which are then transformed into the clustering. The transition from adjacencies to clustering often differs for individual algorithms and can be done in a variety of ways. This is why we do not appoint further basic elements for it on this conceptual level. Doing so would result in a large set of basic elements and contradict our goal of finding only fundamental components. We will address this problem in the next section.

After finishing the core, only the optimization phase is left to consider. The problem of variety explained just now for a part of the association phase, applies to this phase as a whole. As optimization can involve tasks like parameter adjustment, updates to points or references, iteration, etc. the derivation of a minimal set of basic elements is not feasible at this point. As stated before, we will solve this problem in the next section by moving to a different level of abstraction.

4.1.2 Formal Description

So far, we identified the essential tasks common to each clustering algorithm and derived a universal frame made up of the three core phases: evaluation, selection, association and the optional optimization. Furthermore, we defined the basic elements to fit into this frame: points, references, distances, filters, adjacencies, clustering, distance measure, and association function. The descriptions we provided are only abstract and verbal at the moment. In order to realize our description concept, it is necessary to concretize the basic elements. We decided to use mathematical definitions, which allows us to make general and clear descriptions, which can act as a formal intermediate between the abstract verbal and the implementation-specific description.

The set of basic elements can be separated into two groups, which we denote as *actors* and *interactions*. The set of *actors* corresponds to the objects with which a clustering algorithm works and contains points, references, distances, adjacencies and clustering. For their formal representation we utilize matrices as they are a practical approach to describe all of our actors with the same structure.

When it comes to the formal definition of a dataset for clustering, existing literature generally uses multi-dimensional vectors to express the location of data points inside a feature-space. According

to this approach, we can define the basic element *points* as a set P of f-dimensional vectors $\vec{p} = \{p_0, \ldots, p_f\}$ where $(0 \leq j \leq f)$ and with $n = |P|$. This set can be described as a matrix P by interpreting each vector $\vec{p_i}$ as a row $p_{i,*}$ of said matrix. The resulting matrix P has n rows and f columns. As the set of *references* can either be a subset of P or just share the same domain, we can describe it in a similar way. The corresponding set of vectors R contains k elements $\vec{r} = \{r_0, \ldots, r_f\}$ where $(0 \leq j \leq f)$ and is interpreted as matrix $R^{k \times f}$.

While P and R express values of features per point, the remaining actors are instantiations of relations between objects, e.g. point-distance-reference triples or point-cluster tuples. For the formal description of these actors, matrices are especially convenient as the objects involved in a relation correspond to a row and column pair which addresses the matrix element that holds the value of the actual relation. To exemplify this idea, we define the basic element *distances* as a matrix D with n rows and k columns, where n is the number of points and k is the number of references. Each element d_{ij} of D relates to a point/row $p_{i,*}$ of P and a reference $r_{j,*}$ of R. Thus, the value d_{ij} represents the distance between $p_{i,*}$ and $r_{j,*}$. This description can be smoothly translated to the point-cluster tuples that make up the clustering. Point-cluster tuples express the binary relation between a point and a cluster, thus the clustering can be described as a binary matrix C with n rows and a number of columns determined by the number of clusters found. A value of 1 at c_{ij} indicates an existing relation between the point $p_{i,*}$ and the cluster $c_{*,j}$, while 0 states the opposite. Accordingly, we can define *adjacencies* as a binary matrix A that has the same dimensions as D.

Choosing matrices for the descriptions of actors, makes *functions* the natural choice for the formal representation of the *interactions*: distance measure, filter and association function. As an example, we define a general function for the distance measure $dist$. It takes a pair of rows $(p_{i,*}, r_{j,*})$ from P and R as arguments and assigns a scalar value to it that represents the distance between the corresponding objects. The abstract function *dist* is defined as:

$$\begin{aligned} dist : \mathbb{M}^{n \times f} \times \mathbb{M}^{k \times f} &\rightarrow \mathbb{M}^{n \times k} \\ (P, R) &\mapsto D \\ d_{ij} &= function(p_{i,*}, r_{j,*}) \end{aligned}$$

Regarding the formal description of *filters* we face some challenges. A filter checks, whether a matrix or one of its elements fulfills certain conditions and passes them on or sorts them out accordingly. Thus, a filter is similar to an *if-then* statement. In order to describe this behavior with mathematical functions, it is necessary to break down the task and establish some conventions. The defining part of each filter is its *condition*. In mathematical terms, it can be described as a function with the codomain $0, 1$ that represents the results false and true. A simple threshold condition that is satisfied by all numbers smaller 10 is defined as:

$$\begin{aligned} threshold : \mathbb{R} \rightarrow \{0, 1\}, X \mapsto X^I \\ x^I = \begin{cases} 1, & \text{if} \quad x < 10 \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Adopting this form of notation for each condition would be pretty extensive, so we settle for a minimized version that only denotes the condition leading to true, i.e. 1, as the function name. Thus, the notation of the preceding definition of *threshold()* is reduced to $\langle x < 10 \rangle$.

With this convention done, we have to look into the consequence of a filter. While elements that fulfill the provided condition are left untouched, those who fail have to be sorted out or rather deleted. Actual deletion of elements or matrices cannot be expressed with a mathematical function. Therefore, we have to find a workaround for this issue. Let us regard k-means, where the selection phase chooses the minimal point-cluster distance for each point. Assume a row $d_{2,*} = (d_{21}, d_{22}, d_{23}, d_{24})$, of D whose components show the distances between point $p_{2,*}$ and the four centroids of R. The appropriate filter has to sort out all components that are not minimal. Without the chance of removal, it is necessary to define a neutral element to which all inputs that fail the condition are mapped.

For our scenario we state this neutral element as 0. This choice has the benefit that all of the mathematical functions we use naturally accept this neutral element and we do not need to specify explicit handling instructions. Unfortunately in the area clustering 0 can become a problem when it comes to handling distances, because a distance of 0 expresses identity. In such cases 0 should not be ignored but treated as regular value. In our scenario this can only happen in two situations: either when points have duplicates or the particular algorithm procedure leads to a distance measurement between a point and itself. In order to use 0 as a neutral element and focus on functions we make the assumption that P and R are free of duplicates. This takes care of situation one while situation two can be easily resolved by algorithm-specific handling. If duplicates must be considered we propose -1 as a fully valid neutral element. However this requires additional dedicated instructions for every building block. Based on these assumptions we select 0 as neutral element, which allows us to define the minimum filter as:

$$\begin{aligned} minFilter : \mathbb{M}^{1\times k} &\to \mathbb{M}^{1\times k} \\ (D, \langle x = min(D)\rangle) &\mapsto D^I \\ d^I_{ij} &= \langle x = min(D)\rangle\,(d_{ij}) \cdot d_{ij} \end{aligned}$$

Assuming d_{23} as the minimum of $d_{2,*}$, the filtered row becomes $d^I_{i,*} = (0, 0, d_{23}, 0)$. This approach requires that subsequent functions are aware of 0 as the neutral element and treat it accordingly. In summary, we describe filters as composite of a variable condition and a fixed consequence realized by a function that maps to the neutral element. In the context of clustering algorithms, a filter cannot exist without a condition.

The execution of filters during the selection phase creates a modified version of the initial distance matrix that we denote as D^I. It is passed on to the association phase, where it is processed by the association-function. The goal of this function is the transformation of distances into adjacencies, i.e. the point-distance-reference triples that made it past the selection phase are converted into point-reference tuples. Basically, this means that D^I is transformed into a binary matrix, where a value of 1 represents an adjacency. The selection phase already filtered out all object pairs that have no connection. These non-existent adjacencies have already been mapped to 0, so only the task of mapping every non-zero value to 1 remains. Based on this, we define the association function *assoc* as

$$\begin{aligned} assoc : \mathbb{M}^{m\times n} &\to \mathbb{M}^{m\times n} \\ D^I &\mapsto A \\ a_{ij} &= sgn(d^I_{ij}) \end{aligned}$$

where *sgn()* is the sign function. This function is quite convenient as it keeps the neutral element 0 and maps all positive values to 1. Although *sgn()* can yield -1 for negative inputs, this does not need to be considered in our setting as distances are always positive.

Now we possess three items that can be used for the formal description of our basic elements. These are *matrices* to describe all of our actors, *functions* for interactions, and *conditions* for filters. Besides covering the description of actors and interactions, these items enable us to close the gaps in the association and optimization phase we mentioned in Section 4.1.1. Those gaps were induced by the diversity of actions a clustering algorithm can conduct there. We assume that our formal description concepts are universal enough to handle this diversity and describe any functionality that can occur in an algorithm.

Although we identified basic elements and items for their description, we have not clearly defined our actual building blocks. The sole application of the formal description items is not an option, as they are too general. An algorithm created by combining matrices, functions, and conditions will not necessarily be able to perform the tasks essential for data clustering. Likewise, the basic elements cannot function as building blocks alone, due to the limitations regarding variety that we outlined. So in order to create a working vocabulary of building blocks we have to use both proposed groups in unison. Effectively this means that each basic element and each item for description constitutes a building block. To illustrate how both groups are interrelated, consider matrices, functions and conditions as *performers*, while the basic elements define *roles* for these performers. Some of these roles are mandatory and must be performed in order to create a proper clustering algorithm, e.g. the role of distance measure must be taken by a function. In Section 4.1.1 we introduced the *core* of a clustering algorithm that acts as a frame to fit our building blocks into. Now we are able to fill this frame and work out a detailed algorithm *core*.

- **evaluation** - This phase requires at least 4 building blocks: one function playing the role of distance measure and three matrices acting as points, references and distances.
- **selection** - This phase consists of at least one filter with its respective condition.
- **association** - For this phase 3 building blocks are mandatory: two matrices acting as adjacencies and clustering, as well as an association function.

Beyond that, arbitrary clustering functionality can be added to each of these phases, including optimization, by utilizing the existing building blocks.

We present our building blocks in a pseudocode notation. The respective syntax is introduced in the following. Matrices are denoted with single capital letters, e.g. D for the distances. Additional designation is done in the subscript and superscript of the letter. To distinguish different matrix versions we use the superscript: D^I and D^{II} are versions of D after 1 and 2 function applications, while D^x and D^{x+1} designate the versions of D that are in effect for the current and next iteration of the algorithm, respectively. With the subscript, matrices can be described in more detail, e.g. D_R denotes the distances between all references R.

The first pseudocode block introduces the syntax for functions. We use infix notation for the elementary functions: addition, subtraction, multiplication and entry wise multiplication, while prefix notation is used for all other functions.

$A \circ B \to C$	▷ *infix example: entry wise product*
$function(A) \to A^I$	▷ *prefix example: single input function*
$function(A, B) \to C^I$	▷ *prefix example: double input function*

The mandatory role of the distance measure is designated by adding the prefix *dist.* to the function name. This is necessary as different algorithms employ different distance functions. The role of association function is taken by the already defined function *assoc()* in all upcoming examples, which is why we keep the simple function name. Notation of the conditions is not so straightforward as conditions can occur in two contexts: as an input for filter functions and as standalone block. The following pseudocode shows the notation for both cases

---

filter$(M, \langle cond \rangle)$ ▷ *input for filter*
if $cond$ **then** $function$ ▷ *standalone use*

In both applications the condition itself is denoted as Boolean expression $\langle cond \rangle$ which is sufficient for its utilization as part of the filter. In standalone use a condition affects the control flow of the algorithm. To illustrate this, we embed $\langle cond \rangle$ in an if-then block, where the consequence contains the action that is executed if the condition is met. Besides standalone conditions, we have not discussed the description of the control flow. In our considerations so far this area did not really manifest, although it is crucial for almost every clustering algorithm. Because of that, we introduce the *loop* as an additional basic element. Incorporating loops into our scenario is delicate as they cannot be described as functions, which means we have to define them outside the mathematical domain. In order to describe clustering algorithms we basically need two types of loops: a *for-each* loop for element-wise traversal of datasets or clusterings and a *repeat-until* loop for conditioned iterations. These two loops are denoted with the following pseudocode:

---

for each $element$ **of** M **do**
    $\langle body \rangle$
end for $\rightarrow M^I$

repeat with A
    $\langle body \rangle$
until $cond$ **output** $\rightarrow B$

At the top we find the block for the for-each loop, which is generally used to traverse datasets or clusterings by element. The opening statement of the loop specifies the traversed matrix M and the element/granularity of traversal: row, column or component. In our scenario, element-wise traversal is done by splitting up the source matrix into element-matrices—rows, columns or components at the beginning of the loop. After the split, the elements are processed individually according to the instructions of the $\langle body \rangle$. As we need a single matrix as output, the processed elements must be re-assembled at the end of the loop, e.g. row-matrices are appended. This re-assembly is implicitly assumed and not denoted. The loop output is denoted with the assignment after *end for*.

In addition to the described functionality, we use for-each loops for the actual removal of rows and columns from matrices. Sometimes this is inevitable, when clustering algorithms delete references or clusters during optimization. With filters we introduced mapping to the neutral element 0 as a means to tackle deletion. While this works well and is necessary for the clear definition of functions, the handling of whole rows and columns of zeros can become challenging and lead to problems, e.g. if empty clusters occur in C. Some of those issues could be tackled by introducing constraints to each function to ignore all-zero rows/columns. But this would be complex and not an overall solution. Our described for-each loop provides an elegant way to solve this problem. With the insertion of an appropriate condition before matrix reassembly at the end of the loop, we prevent zero element-matrices from entering the output matrix. Since loops are defined outside the mathematic formalism anyway, adding row/column removal here provides us with a convenient tool without compromising the formal description of the remaining building blocks.

The *repeat-until* loop is used to represent conditioned loops. This kind of loop is used to control algorithm iterations or the minimization/maximization of target functions like the sum of squared errors in k-means. The stopping condition for the loop is always specified after the closing *until* statement. A repeat-until loop has one or more input matrices—denoted in the opening statement—which are continuously processed from iteration to iteration and an output matrix that is obtained when the loop finishes. This output matrix can be either a processed version of the input or an assembly of element-matrices generated during the loop. The particular type of output can be derived from the instructions of the $\langle body \rangle$ of the loop.

Now that we have described the last of our building blocks and their syntax, we move on to the next section, in which we demonstrate how clustering algorithms are transcribed using the proposed vocabulary.

4.1.3 Transcription of Algorithms

The description of algorithms with our building blocks may be unfamiliar at first as it requires to think in terms of matrices and does not offer well-known structures like maps or control flow elements like recursion. However, this limitation is necessary to reach consistency and we will illustrate the application of our building blocks by translating several well-known clustering algorithms. We start with the partitioning algorithms *k-means*[50] and its variant *ISODATA* [8]. After that, we proceed with the density-based *DBSCAN* [21] and conclude our examples with the hierarchical method *AGNES*[46]. During our descriptions, we refer to lines of pseudocode by putting the corresponding line numbers in brackets.

k-means

For our first transcription we choose the best-known clustering algorithm in existence: k-means [50, 53], which is shown in Algorithm 1. We start by going through all of our proposed core phases. First comes evaluation, where the similarities between the dataset and the initial cluster centroids are calculated. We already stated that this phase requires at least four building blocks: three matrices and one function. The dataset to be clustered is represented by the points matrix P while the references are contained in matrix R, whose k rows represent the initial centroids. The last matrix is the distance matrix D that contains the distances between P and R calculated with the distance measure. For our k-means translation, this mandatory role is taken by the euclidean distance. We adapt it to our matrix setting as function $dist.L_2$ which is defined as:

$$dist.L_2 : \mathbb{M}^{n\times f} \times \mathbb{M}^{k\times f} \rightarrow \mathbb{M}^{n\times k}$$
$$(P, R) \mapsto D \quad \text{with} \quad d_{ij} = \sqrt{\sum_{l=1}^{f} (p_{il} - r_{jl})^2}$$

where $p_{i,*}$ and $r_{j,*}$ are rows of their respective matrices. The resulting matrix D is used as input for the following selection phase. The goal of the selection phase is to decide which points are similar and should form clusters. In k-means each object is assigned to its nearest cluster, so selection must find the minimal distance for each object. This handling of each object is realized in a for-each loop that traverses D in a row-wise fashion (6).

Algorithm 1 k-means

1: **repeat with** R^x
2: **phase** EVALUATION
3: $dist.L_2(P, R^x) \rightarrow D$ ▷ $\mathbb{M}^{n\times f} \times \mathbb{M}^{k\times f} \rightarrow \mathbb{M}^{n\times k}$
4: **end phase**
5: **phase** SELECTION
6: **for each** $d_{i,*}$ **of** D **do**
7: $filter(d_{i,*}, \langle d_{ij} = min(d_{i,*})\rangle)$
8: **end for** $\rightarrow D^I$
9: **end phase**
10: **phase** ASSOCIATION
11: $assoc(D^I) \rightarrow A$ ▷ $\mathbb{M}^{n\times k} \rightarrow \mathbb{M}^{n\times k}$
12: $A \rightarrow C$
13: **end phase**
14: **phase** OPTIMIZATION
15: $updt(C^T, P) \rightarrow R^{x+1}$ ▷ $\mathbb{M}^{k\times n} \times \mathbb{M}^{n\times f} \rightarrow \mathbb{M}^{k\times f}$
16: **end phase**
17: **until** $R^x = R^{x+1}$ **output** $\rightarrow C$

Due to the evaluation phase each row $d_{i,*}$ contains all distances between a point $p_{i,*}$ and all rows of R. The loop body contains a filter function that keeps the minimum element d_{ij} from each row and maps all remaining elements to 0. At the end of the loop the processed rows are re-assembled into the filtered matrix D^I that is passed on to the association phase. For k-means, association is pretty simple as the minimum cluster distance for each object also states its cluster assignment. Thus, we apply the already defined function *assoc* and turn D^I into the binary adjacency matrix A. Since k-means uses the references of R as cluster representatives, matrix A already contains the final cluster assignments and is simply adopted as C (12).

Now that the core phases are finished and a clustering result was generated, k-means enters its optimization phase. Here, the centroids/references are updated for the next iteration of the algorithm. Each centroid is recalculated as the arithmetic average of all points that are currently assigned to it. In our setting this means, we have to use the information stored in C and P to create a new version of R with the correct dimensions. We realize the update by using the matrix-multiplication as a template and C and P as its input. Matrix C contains the point-cluster assignments and has the dimensions $n \times k$ with k being the number of references/centroids. The second input P has the dimension $n \times f$ with n being the number of points and f being the number of features of the dataset. By multiplication, we use the binary character of C to select parts of P and create an updated version of R. The new version must inherit the dimension of $k \times f$. Matrix multiplication of C and P requires that the number of columns of C matches the number of rows in P, which is not the case as $k \neq n$. To solve this, we transpose C to C^T which not only establishes the required match of columns and rows, but also guarantees that the result has the desired dimension $k \times f$. The function used for calculation of the update is defined as:

$$updt : \mathbb{M}^{k\times n} \times \mathbb{M}^{n\times f} \rightarrow \mathbb{M}^{k\times f} \tag{4.1}$$

$$(C^T, P) \mapsto R_{x+1} \tag{4.2}$$

$$r_{ij} = \frac{\sum_{l=1}^{n} c_{il} \cdot p_{lj}}{\sum_{l=1}^{n} c_{il}} \tag{4.3}$$

with $c_{i,*}$ being a row of C^T and $p_{*,j}$ being a *column* of P. In C^T each cluster is represented by a binary row. The function uses this row to select those rows of P that are assigned to the corresponding cluster. During matrix multiplication, each row of C^T is multiplied with every column of P. Thus, the members of the cluster are selected in each feature represented by $p_{*,j}$. These selections are summed up for each feature and divided by the number of cluster members that are obtained by summing up all binary elements of $c_{i,*}$. This results in an updated row/centroid of R^{x+1}.

The iteration of k-means is described, by surrounding the whole algorithm with a *repeat-until* loop that works by continuously updating the references/centroids of R^x. At the end of each iteration, the stopping criterion is evaluated before the loop is restarted (17). In our transcription we choose $R^x = R^{x+1}$ as stopping condition, i.e. the algorithm quits if the references/centroids no longer change and stabilized clusters can be assumed. Further possible stopping conditions are: reaching a fixed number of iterations or meeting the predefined threshold of a quality measure.

ISODATA

The ISODATA algorithm was introduced as a variant of k-means that is not bound by a fixed number of clusters k, but adapts this number during execution [8]. This adaptation is done by removing, splitting, and merging centroids after each iteration. Each of these actions is triggered in reference to 5 user-defined parameters that the algorithm introduces as:

- k_D the number of *desired* clusters.
- θ_N minimal cluster size, below which clusters are removed.
- θ_C minimal centroid-to-centroid distance, below which centroids are merged.
- θ_E maximal standard deviation inside a cluster, above which clusters are split.
- L number of merges allowed per iteration.

Due to the number of parameters and the substantial adjustment procedure, ISODATA is a much more complex algorithm than k-means. As a result, its description with our building blocks becomes quite extensive, which is why we break it up in three parts to improve traceability. The three parts are: *trunk*, *split branch* and *merge branch*. The description of the trunk part is shown in Algorithm 2. It is easy to see that the core of ISODATA—evaluation, selection, and association phase—is identical to the previously described k-means. Everything that distinguishes ISODATA is concentrated in its optimization phase, which is considerably larger in comparison with k-means. Although optimization strongly differs for both algorithms, its main goal remains the update of centroids/references for the next iteration.

The optimization phase begins in the trunk part, where the removal of small clusters is done (15-18). We already know that each column of clustering matrix C represents a cluster. In order to sort out clusters that are too small, a for-each loop is used for the column-wise traversal of C. A filter block in the body of the loop checks if a cluster exceeds the minimum size θ_N. If the sum of a column is too small, i.e. the cluster has not enough members, said filter maps all elements of the column to zero. Subsequently we deploy a standalone condition to prevent columns that only contain zeros from entering the reassembly of the result matrix C^I. For the remaining clusters, new centroids/references are created with the known function *updt()*. Similar to its application in k-means, we use the transpose of C^I and P as input and calculate an updated reference matrix R^I (19).

Algorithm 2 ISODATA trunk

1: **repeat with** R^x
2: **phase** EVALUATION
3: $dist.L_2(P, R^x) \rightarrow D$ ▷ $\mathbb{M}^{n\times f} \times \mathbb{M}^{k\times f} \rightarrow \mathbb{M}^{n\times k}$
4: **end phase**
5: **phase** SELECTION
6: **for each** $d_{i,*}$ **of** D **do**
7: $filter(d_{i,*}, \langle d_{ij} = min(d_{i,*})\rangle)$
8: **end for** $\rightarrow D^I$
9: **end phase**
10: **phase** ASSOCIATION
11: $assoc(D^I) \rightarrow A$ ▷ $\mathbb{M}^{n\times k} \rightarrow \mathbb{M}^{n\times k}$
12: $A \rightarrow C$
13: **end phase**
14: **phase** OPTIMIZATION
15: **for each** $c_{*,j}$ **of** C **do**
16: $filter(c_{*,j}, \langle \sum_{l=0}^{n} c_{lj} > \theta_N\rangle)$
17: **if** $\sum_{l=0}^{n} c_{lj} > 0$ **then** $\rightarrow c^I_{*,j}\ of\ C^I$
18: **end for** $\rightarrow C^I$
19: $updt((C^I)^T, P) \rightarrow R^I$ ▷ $\mathbb{M}^{k^I\times n} \times \mathbb{M}^{n\times f} \rightarrow \mathbb{M}^{k^I\times f}$
20: **if** $x \mod 2 = 1 \vee k \leq k_D/2$ **then** $\nearrow$ *split branch*
21: **if** $x \mod 2 = 0 \vee k > 2k_D \vee x = x_\Omega$ **then** $\nearrow$ *merge branch*
22: **end phase**
23: **until** $x = x_\Omega$ **output** $\rightarrow C$

At this point, the algorithm branches to either perform splitting or merging. The split branch is executed if the current iteration is odd or the current number of clusters k is less or equal half the desired number of clusters k_D (20). Merging is performed if one of the following conditions is met: the current iteration is even, the current iteration is the last iteration or k is larger than two times k_D (21). We describe the branching with two standalone conditions that reference each of the respective branches. Like k-means, the whole trunk part is surrounded by a repeat-until loop that describes the ongoing iteration of the algorithm. But this time, the stopping criterion is defined as reaching a predefined number x_Ω of iterations.

After regarding the trunk part, we deal with the branch of ISODATA that is responsible for splitting clusters. At first, we regard the original splitting process proposed in [8]. There are three conditions that matter when it comes to the execution of a cluster split. There is a main condition that is fulfilled, if a cluster exceeds the standard deviation threshold θ_E in at least one dimension. Furthermore, two additional constraints φ_1 and φ_2 are considered. The first constraint φ_1 is met if a cluster exceeds a size of $2\theta_N + 2$ and in addition has an average point-centroid distance that is higher than the average point-cluster distance of the clustering. The second constraint φ_2 is fulfilled if the current number of clusters k is less or equal half of the desired number of clusters k_D. If the main condition *and* either φ_1 *or* φ_2 are satisfied, the cluster is split into two new clusters. To realize this, the initial cluster centroid is duplicated, before 1 is added or subtracted in the dimension with maximal standard deviation. This addition/subtraction of offsets is necessary to differentiate the duplicates. Transcribing the splitting procedure with our matrix-based building blocks, requires the completion of three main tasks: condition handling, centroid duplication and offset addition. Subsequently we describe how these tasks are implemented. The final transcribed split branch is shown in Algorithm 3.

As a first step, we address condition handling by calculating the necessary values like standard deviations and average point-cluster distances $(2-5)$. By multiplying C^I and R^I that were generated in

the trunk part, we create a helper matrix $H^{n \times f}$. As a result of multiplication, each row of H contains the centroid to which the corresponding row in P is assigned. As H and P have the same dimensions, we can use matrix subtraction to calculate the distance from each point to its centroid in each dimension. The resulting matrix H^I and the transpose of C^I are used to compute the standard deviations with the function *sigma* defined as:

$$\begin{aligned} sigma : \mathbb{M}^{k \times n} \times \mathbb{M}^{n \times f} &\to \mathbb{M}^{k \times f} \\ ((C^I)^T, H^I) &\mapsto S \\ s_{ij} &= \sqrt{\frac{\sum_{l=1}^{n} c_{il} \cdot (h_{lj}^I)^2}{(\sum_{i=1}^{n} c_{il}) - 1}} \end{aligned}$$

that results in a matrix S, whose rows $s_{i,*}$ contain the standard deviation per dimension for each of the k clusters. The same inputs are used to calculate the average point-centroid distance per cluster with the function *avgDist* defined as:

$$\begin{aligned} avgDist : \mathbb{M}^{k \times n} \times \mathbb{M}^{n \times f} &\to \mathbb{M}^{k \times 1} \\ ((C^I)^T, H^I) &\mapsto D \\ d_{i1} &= \frac{\sum_{l=1}^{n} \sqrt{\sum_{j=1}^{f} c_{il}^I \cdot (h_{lj}^I)^2}}{\sum_{l=1}^{n} c_{il}^I} \end{aligned}$$

which yields the column matrix D_{avg}. Basically, both functions use $(C^I)^T$ for the cluster-wise selection of rows from H^I, which is an approach we already used in *updt()*. The selected values are squared and summed up before the root is extracted and the results are normalized. After that, the cluster sizes are calculated by applying the aggregation function *agg* defined as:

$$\begin{aligned} agg : \mathbb{M}^{m \times n} &\to \mathbb{M}^{1 \times n} \\ C^I &\mapsto C^{II} \\ c_{1j}^{II} &= \sum_{l=1}^{m} c_{mj}^I \end{aligned}$$

which results in a row-matrix C^{II} that contains the size of each cluster (6). These sizes are necessary to obtain the average point-cluster distance for the whole clustering. According to [8] this value is computed as the weighted average of the average distances per cluster that uses the respective cluster sizes as weights. To execute this calculation, we can employ *updt()* again and specify C^{II} and D_{avg} as its input. The resulting 1×1 matrix D_{avg}^I contains the overall average distance.

With the calculated matrices and given parameters we are able to evaluate the main condition as well as φ_1 and φ_2. These three conditions have different characters and relations with each other. The main condition and φ_1 address actual cluster characteristics, while φ_2 considers the global variables k and k_D. Furthermore, the additional constraints have an 'or' relationship. In our setting, this means

Algorithm 3 ISODATA split branch

1: **phase** OPTIMIZATION
2: $C^I \cdot R^I \rightarrow H$ ▷ $\mathbb{M}^{n \times k^I} \times \mathbb{M}^{k^I \times f} \rightarrow \mathbb{M}^{n^I \times f}$
3: $H - P \rightarrow H^I$
4: $sigma((C^I)^T, H^I) \rightarrow S$ ▷ $\mathbb{M}^{k^I \times n} \times \mathbb{M}^{n \times f} \rightarrow \mathbb{M}^{k^I \times f}$
5: $avgDist((C^I)^T, H^I) \rightarrow D_{avg}$ ▷ $\mathbb{M}^{k^I \times n} \times \mathbb{M}^{n \times f} \rightarrow \mathbb{M}^{k^I \times 1}$
6: $agg(C^I) \rightarrow C^{II}$ ▷ $\mathbb{M}^{n \times k^I} \rightarrow \mathbb{M}^{1 \times k^I}$
7: $updt(C^{II}, D_{avg}) \rightarrow D^I_{avg}$ ▷ $\mathbb{M}^{1 \times k^I} \times \mathbb{M}^{k^I \times 1} \rightarrow \mathbb{M}^{1 \times 1}$

8: $filter(D_{avg}, \left\langle d_{avg(ij)} \leq D^I_{avg} \right\rangle) \rightarrow H_{cond1}$
9: $filter(C^{II}, \left\langle c^{II}_{ij} < 2\theta_N + 2 \right\rangle) \rightarrow C^{III}$
10: $(C^{III})^T + H_{cond1} \rightarrow H^I_{cond1}$
11: **if** $k > k_D/2$ **then** $O^{1 \times f} \rightarrow H_{cond2}$
12: **if** $k \leq k_D/2$ **then** $Z^{1 \times f} \rightarrow H_{cond2}$
13: $H^I_{cond1} \cdot H_{cond2} \rightarrow H_{cond}$ ▷ $\mathbb{M}^{k^I \times 1} \times \mathbb{M}^{1 \times f} \rightarrow \mathbb{M}^{k^I \times f}$
14: $sgn(H_{cond}) \rightarrow H^I_{cond}$
15: $subst(H^I_{cond}, S, \theta_E) \rightarrow S^I$

16: **for each** $s_{i,*}$ **of** S^I **do**
17: $filter(s_{i,*}, \langle s_{ij} = max(s_{i,*}) \rangle)$
18: **end for** $\rightarrow S^{II}$
19: $agg((S^{II})^T) \rightarrow S^{III}$ ▷ $\mathbb{M}^{f \times k^I} \rightarrow \mathbb{M}^{1 \times k^I}$
20: $(S^{III})^T \cdot O^{1 \times k^I} \rightarrow H_{split}$ ▷ $\mathbb{M}^{k^I \times 1} \times \mathbb{M}^{1 \times k^I} \rightarrow \mathbb{M}^{k^I \times k^I}$
21: $H_{split} \circ I \rightarrow H^I_{split}$

22: **for each** $h^I_{i,*}$ **of** H^I_{split} **do**
23: **if** $\sum h^I_{ij} > \theta_E$ **then** $\begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow M_{clone}$
24: **if** $\sum h^I_{ij} \leq \theta_E$ **then** $\begin{bmatrix} 1 \end{bmatrix} \rightarrow M_{clone}$
25: $M_{clone} \cdot h^I_{i,*} \rightarrow h^{II}_{i,*}$
26: **end for** $\rightarrow H^{II}_{split}$
27: $sgn(H^{II}_{split}) \rightarrow H^{III}_{split}$

28: **for each** $s^{II}_{i,*}$ **of** S^{II} **do**
29: **if** $\sum s^{II}_{ij} > \theta_E$ **then** $\begin{bmatrix} 1 \\ -1 \end{bmatrix} \rightarrow M_{offset}$
30: **if** $\sum s^{II}_{ij} \leq \theta_E$ **then** $\begin{bmatrix} 0 \end{bmatrix} \rightarrow M_{offset}$
31: $M_{offset} \cdot s^{II}_{i,*} \rightarrow (h_{offset})_{i,*}$
32: **end for** $\rightarrow H_{offset}$
33: $sgn(H_{offset}) \rightarrow H^I_{offset}$

34: $H^{III}_{split} \cdot R^I \rightarrow R_{split}$ ▷ $\mathbb{M}^{k^{II} \times k^I} \times \mathbb{M}^{k^I \times f} \rightarrow \mathbb{M}^{k^{II} \times f}$
35: $R_{split} + H^I_{offset} \rightarrow R^{x+1}$
36: **end phase**

H^I_{cond1}

$$\begin{matrix} 0 \\ |c_2| \\ \bar{d}_3 \\ 0 \end{matrix}$$

(a)

$\rightarrow$

H_{cond}

$$\begin{matrix} 0 & 0 & 0 & 0 \\ |c_2| & |c_2| & |c_2| & |c_2| \\ \bar{d}_3 & \bar{d}_3 & \bar{d}_3 & \bar{d}_3 \\ 0 & 0 & 0 & 0 \end{matrix}$$

(b)

$\rightarrow$

H^I_{cond}

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{matrix}$$

(c)

$\rightarrow$

S^I

$$\begin{matrix} s_{11} & s_{12} & s_{13} & s_{14} \\ \theta_E & \theta_E & \theta_E & \theta_E \\ \theta_E & \theta_E & \theta_E & \theta_E \\ s_{41} & s_{42} & s_{43} & s_{44} \end{matrix}$$

(d)

$\rightarrow$

S^{II}

$$\begin{matrix} 0 & 0 & s_{13} & 0 \\ \theta_E & 0 & 0 & 0 \\ \theta_E & 0 & 0 & 0 \\ s_{41} & 0 & 0 & 0 \end{matrix}$$

(e)

$\rightarrow$

S^{III}

$$\begin{matrix} s_{13} & \theta_E & \theta_E & s_{41} \end{matrix}$$

(f)

$\rightarrow$

H_{split}

$$\begin{matrix} s_{13} & s_{13} & s_{13} & s_{13} \\ \theta_E & \theta_E & \theta_E & \theta_E \\ \theta_E & \theta_E & \theta_E & \theta_E \\ s_{41} & s_{41} & s_{41} & s_{41} \end{matrix}$$

(g)

$\rightarrow$

H^I_{split}

$$\begin{matrix} s_{13} & 0 & 0 & 0 \\ 0 & \theta_E & 0 & 0 \\ 0 & 0 & \theta_E & 0 \\ 0 & 0 & 0 & s_{41} \end{matrix}$$

(h)

$\rightarrow$

H^{II}_{split}

$$\begin{matrix} s_{13} & 0 & 0 & 0 \\ s_{13} & 0 & 0 & 0 \\ 0 & \theta_E & 0 & 0 \\ 0 & 0 & \theta_E & 0 \\ 0 & 0 & 0 & s_{41} \\ 0 & 0 & 0 & s_{41} \end{matrix}$$

(i)

$\rightarrow$

H^{III}_{split}

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{matrix}$$

(j)

H_{offset}

$$\begin{matrix} 0 & 0 & s_{13} & 0 \\ 0 & 0 & -s_{13} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ s_{41} & 0 & 0 & 0 \\ -s_{41} & 0 & 0 & 0 \end{matrix}$$

(k)

$\rightarrow$

H^I_{offset}

$$\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{matrix}$$

(l)

Figure 4.1: Matrix sequence for the split branch.

$$H_S^{III} \quad R^I \quad R_{split}$$

$$\begin{matrix}1&0&0&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\0&0&0&1\end{matrix} \cdot \begin{matrix}r_{11}&r_{12}&r_{13}&r_{14}\\r_{21}&r_{22}&r_{23}&r_{24}\\r_{31}&r_{32}&r_{33}&r_{34}\\r_{41}&r_{42}&r_{43}&r_{44}\end{matrix} \rightarrow \begin{matrix}r_{11}&r_{12}&r_{13}&r_{14}\\r_{11}&r_{12}&r_{13}&r_{14}\\r_{21}&r_{22}&r_{23}&r_{24}\\r_{31}&r_{32}&r_{33}&r_{34}\\r_{41}&r_{42}&r_{43}&r_{44}\\r_{41}&r_{42}&r_{43}&r_{44}\end{matrix}$$

(a) Centroid duplication.

$$R_{split} \quad H^I_{offset} \quad R^{x+1}$$

$$\begin{matrix}r_{11}&r_{12}&r_{13}&r_{14}\\r_{11}&r_{12}&r_{13}&r_{14}\\r_{21}&r_{22}&r_{23}&r_{24}\\r_{31}&r_{32}&r_{33}&r_{34}\\r_{41}&r_{42}&r_{43}&r_{44}\\r_{41}&r_{42}&r_{43}&r_{44}\end{matrix} + \begin{matrix}0&0&1&0\\0&0&-1&0\\0&0&0&0\\0&0&0&0\\1&0&0&0\\-1&0&0&0\end{matrix} \rightarrow \begin{matrix}r_{11}&r_{12}&r_{13}+1&r_{14}\\r_{11}&r_{12}&r_{13}-1&r_{14}\\r_{21}&r_{22}&r_{23}&r_{24}\\r_{31}&r_{32}&r_{33}&r_{34}\\r_{41}+1&r_{42}&r_{43}&r_{44}\\r_{41}-1&r_{42}&r_{43}&r_{44}\end{matrix}$$

(b) Offset addition

Figure 4.2: Execution of centroid splitting.

that if φ_2 holds, φ_1 does not matter anymore and vice versa. We model this behavior in the following way. First, we construct a filter matrix H_{cond1} that captures all centroids failing φ_1. If φ_2 holds, this matrix is neutralized and only the main condition is evaluated. If φ_2 does not hold, we have to ensure that all centroids referenced by H_{cond1} also fail the main condition. This is necessary, as a centroid cannot be split if neither φ_1 nor φ_2 is satisfied.

To handle φ_1 in Algorithm 3, we employ filters that select those cluster sizes and average distances that are not high enough to fulfill the given constraints $(8, 9)$. The resulting matrices H_{cond1} and $(C^{III})^T$ are added and form a column-matrix H^I_{cond1} where each row having a value greater 0 indicates a corresponding centroid/row that does not fulfill φ_1. Later on we want to use this information for element-wise manipulation of S, so we need to expand the $k \times 1$ column-matrix H^I_{cond1} to a $k \times f$ matrix that matches S. With this expansion we also want to consider φ_2. In order to add the required number of columns to H^I_{cond1}, it is multiplied with a $1 \times f$ matrix. If φ_2 holds, we multiply H^I_{cond1} with a zero matrix $Z^{1\times f}$, which results in a $k \times f$ matrix of zeros and effectively discards all information regarding φ_1. In doing so, we realize the 'or' relationship of φ_1 and φ_2. In the opposite case, H^I_{cond1} is multiplied with a row-matrix of ones $O^{1\times f}$, which produces a $k \times f$ matrix. Each row of this matrix that contains elements greater 0, corresponds to a cluster that fails φ_1. This matrix named H_{cond} is transformed into a binary matrix with the sign function (14).

In the next step we have to ensure that each cluster that fails φ_1 also fails the main condition. For this purpose, we substitute the standard deviations of these clusters with values that will not exceed θ_E and effectively prevent them from being split. As substitution value λ we choose θ_E as only standard deviations exceeding this threshold will trigger a split. The function for this task *subst()* is defined as:

$$subst : \mathbb{M}^{k\times f} \times \mathbb{M}^{k\times f} \rightarrow \mathbb{M}^{k\times f} \tag{4.4}$$

$$(H, S, \lambda) \mapsto S^I \tag{4.5}$$

$$s^I_{ij} = |h_{ij} - 1| \cdot s_{ij} + (h_{ij} \cdot \lambda) \tag{4.6}$$

and results in a matrix S^I. A value greater 0 at position (ij) in H^I_{cond} will result in an element s^I_{ij} with the value θ_E in S^I. In contrast, a zero element in H^I_{cond} means that the corresponding component from S is taken over to the new matrix. If φ_2 holds, H^I_{cond} becomes a zero matrix and *subst()* does

not change S, which is the behavior we desired. To exemplify the steps taken so far, we show a sequence of example matrices in Figure 4.1(a)-(d) that assume 4 clusters/centroids. In the top row we see a matrix H^I_{cond1} that indicates two clusters that fail φ_1, as the cluster size $|c_2|$ and the average point-centroid distance $\bar{d}_3$ do not exceed the required thresholds. The next three examples show the mentioned steps of expansion, binary transformation and substitution of S under the assumption that φ_2 does not hold. To prepare the final handling of the main condition, we use a for-each loop that traverses the rows of S^I. A maximum filter is applied to each row and leads to a matrix S^{II} that only contains the maximal standard deviation for each row/cluster $(16-18)$. An example S^{II} is depicted in Figure 4.1(e).

Now we are able to address the actual centroid splitting. Our basic idea for this task is to use matrix multiplication with a modified identity matrix to duplicate certain rows of R. The multiplication of a $k \times k$ identity matrix I with the $k \times f$ matrix R will obviously result in the same matrix R. If I is modified by duplicating its first row, the multiplication with R results in a $(k+1) \times f$ matrix R^I. This new matrix has an additional row, which is a duplicate of the first row of R. The construction of the modified identity matrix for splitting requires some preparation, so we begin by aggregating the transpose of S^{II} into a $k \times 1$ column matrix S^{III} with *agg()* (19). Next, $(S^{III})^T$ is expanded to a $k \times k$ matrix H_{split} by multiplication with a row-matrix of ones $0^{1\times k}$. Afterwards, we create the element-wise product of H_{split} and an identity matrix $I^{k\times k}$. The resulting H^I_{split} is a diagonal matrix, where the single non-zero element of each row represents the maximal standard deviation of its corresponding cluster (21). Examples for the evolution from S^{III} to H^I_{split} are shown in Figure 4.1(f)-(h).

To check the main condition and implement duplication, we iterate over the rows of H^I_{split} using a for-each loop $(22-26)$. For each row the main condition is evaluated by checking if its sum exceeds θ_E. If this condition is met, the row is duplicated by multiplication with a 2×1 matrix of ones $O^{2\times 1}$ (23). Rows that fail are multiplied with $O^{1\times 1}$ and left unchanged (24). Through this, the particular rows of the result matrix H^{II}_{split} are created (25) and re-assembled at the end of the loop (26). The obtained result is transformed into the binary H^{III}_{split} by applying the sign function. Examples for both matrices are shown in Figure 4.1(i) and Figure 4.1(j).

We can use the same approach again, for the creation of the $+1/-1$ offsets in the dimensions with maximal standard deviation $(28-33)$. The matrix S^{II} already contains these elements. It is traversed row-wise and each row that meets the main condition is duplicated by multiplication with a given 2×1 matrix (29). Only small modifications to this duplication matrix are required to create the necessary offsets. Rows that do not exceed θ_E are multiplied with a 1×1 zero matrix and become a single row of zeros. At the end of the loop, the processed rows are re-assembled into the result matrix H_{offset} which is transformed into H^I_{offset} by applying the sign function $(31-33)$. In Figure 4.1(k) and Figure 4.1(l) examples for these matrices are shown. The final steps of the splitting branch are the multiplication of H^{III}_{split} and R^I to get the duplicated centroids R_{split}, and the subsequent addition of H^I_{offset}. With this, the updated references R^{x+1} for the next iteration are created and the split branch is finished. Examples for the final operations are shown in Figure 4.2.

With the description of the split branch finished, we discuss the remaining merge branch of ISODATA. According to [8], the merging procedure evaluates inter-centroid distances and merges pairs of clusters that have a distance smaller θ_C. The user-defined parameter L limits the number of cluster pairs that can be merged during one iteration of the algorithm. Merging is implemented by first sorting all centroid pairs according to their distance and then combining them, starting with the smallest distance. Fusion of two clusters is done by calculating the weighted average of their centroids, whereas the cluster sizes are used as weights. If a cluster was part of a merge, it cannot take part in further merges during the current iteration. To translate this procedure into our matrix-based setting, we reuse the previously introduced idea of changing the number of clusters by multiplication with a

Algorithm 4 ISODATA merge branch

1: **phase** OPTIMIZATION
2: $dist.L_2(R^I, R^I) \rightarrow D_R$ ▷ $\mathbb{M}^{k^I \times f} \times \mathbb{M}^{k^I \times f} \rightarrow \mathbb{M}^{k^I \times k^I}$
3: $filter(D_R, \left\langle d_{R(ij)} < \theta_C \right\rangle) \rightarrow H_{dist}$
4: $I^{k^I \times k^I} \rightarrow H_{merge}$

5: **repeat with** H^x_{dist}, H^x_{merge}
6: $filter(H^x_{dist}, \left\langle h^x_{dist(ij)} = min'(H^x_{dist}) \right\rangle) \rightarrow H^I_{dist}$
7: $H^I_{dist} + (H^I_{dist})^T \rightarrow H^{II}_{dist}$
8: $sgn(H^{II}_{dist}) \rightarrow H^{III}_{dist}$
9: $H^{III}_{dist} + H^x_{merge} \rightarrow H^{x+1}_{merge}$

10: $agg(H^{III}_{dist}) \rightarrow H_{rem}$ ▷ $\mathbb{M}^{k^I \times k^I} \rightarrow \mathbb{M}^{1 \times k^I}$
11: $O^{k^I \times 1} \cdot H_{rem} \rightarrow H^I_{rem}$ ▷ $\mathbb{M}^{k^I \times 1} \times \mathbb{M}^{1 \times k^I} \rightarrow \mathbb{M}^{k^I \times k^I}$
12: $H^I_{rem} + (H^I_{rem})^T \rightarrow H^{II}_{rem}$
13: $sgn(H^{II}_{rem}) \rightarrow H^{III}_{rem}$
14: $O^{k^I \times k^I} - H^{III}_{rem} \rightarrow H^{IV}_{rem}$
15: $H^{IV}_{rem} \circ H^x_{dist} \rightarrow H^{x+1}_{dist}$
16: **until** $\sum H^{x+1}_{dist} = 0 \vee (x+1) > L$ **output** $\rightarrow H^I_{merge}$

17: $agg(C^I) \rightarrow H_{size}$ ▷ $\mathbb{M}^{n \times k^I} \rightarrow \mathbb{M}^{1 \times k^I}$
18: $O^{k^I \times 1} \cdot H_{size} \rightarrow H^I_{size}$ ▷ $\mathbb{M}^{k^I \times 1} \times \mathbb{M}^{1 \times k^I} \rightarrow \mathbb{M}^{k^I} \times k^I$
19: $H^I_{size} \circ H^I_{merge} \rightarrow M$

20: $distinct(M) \rightarrow M_{distinct}$

21: $updt(M_{distinct}, R^I) \rightarrow R^{x+1}$ ▷ $\mathbb{M}^{k^{II} \times k^I} \times \mathbb{M}^{k^I \times f} \rightarrow \mathbb{M}^{k^{II} \times f}$
22: **end phase**

modified identity matrix. In contrast to the split branch, this time we need to reduce the number of clusters. Let us assume an identity matrix I for R. We add up the first and second row of I and replace each of them with the calculated sum. This results in a matrix, whose first row now has a 1 as first and second element and whose number of rows is reduced by one. Multiplication of this modified I and R will result in a matrix with one row less than R and a first row that is the sum of the first two rows from R. Besides this merging, our transcription has to ensure the correct order of merges and must exclude already merged centroids from further processing. The fully transcribed merge branch is shown in Algorithm 4.1.3. For exemplification, we again provide a sequence of example matrices in Figure 4.3.

The merge branch begins with the calculation of the inter-centroid distances D_R. We implement it, by using the filtered references R^I from the trunk part as inputs for the already known euclidean distance. Condition handling is done with a filter that removes all distances larger θ_C. The resulting matrix H_{dist} contains only those distances that can lead to a merge (3). We also initialize H_{merge} as identity matrix for R^I, which will become the modified identity matrix for merging later. Next, it is necessary to establish the right processing sequence for the pair distances. Since actual sorting is not possible with our building blocks, we employ a repeat-until loop that continuously picks the minimum distance for processing $(5-9)$. The loop works on two matrices: H_{dist} is used for realization of the processing sequence while H_{merge} will collect the pairs of centroids that should be merged. At the beginning of the loop, a filter is deployed to select the minimum entry from H^x_{dist}, which is the

Figure 4.3: Matrices for one iteration of the merge branch.

working version of H_{dist} in the loop. As 0 is used as neutral element, we have to modify the minimum condition in order to prevent the selection of zero values. Thus, *min'()* of a matrix M refers to the smallest entry of M that is still greater than 0.

Filtering yields a matrix H^I_{dist} that contains only one non-zero element. Due to the previous filtering according to θ_C, the position (ij) of this minimum indicates the centroids to be merged. In order to merge centroids i and j, we need a single row with a 1 at position i and j in our modified identity matrix. To do this, we add H^I_{dist} and its transpose to retain symmetry. This addition results in matrix H^{II}_{dist}, which is transformed into a binary matrix and added to H^x_{merge} in order to create H^{x+1}_{merge} $(8, 9)$. In doing so, H^x_{merge} becomes an adjacency matrix for centroids. It starts out as I with each centroid on its own and is modified throughout the loop by adding pairwise associations. Example matrices for the steps described so far, are shown in Figure 4.3(a) through (d), where d_{12} is a minimum component smaller θ_C which indicates that centroids 1 and 2 can be merged.

After identification of the first pair for merging, H^x_{dist} must be modified for the next iteration of the loop to prevent the selection of already processed centroids from further merges. If centroids i and j are merged, all elements of their corresponding rows and columns in H^x_{dist} are set to zero. This

H_{select}

Σ_1	Σ_2	Σ_3	Σ_4	Σ_5

(a)

H^{II}_{select}

1	0	0	0	0

(b)

$m_{i,*}$

$\lvert c_1\rvert$	$\lvert c_2\rvert$	0	0	0

(c)

H_{filter}

$\lvert c_1\rvert$	$\lvert c_2\rvert$	0	0	0
$\lvert c_1\rvert$	$\lvert c_2\rvert$	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

(d)

M^{x+1}

0	0	0	0	0
0	0	0	0	0
0	0	$\lvert c_3\rvert$	0	0
0	0	0	$\lvert c_4\rvert$	$\lvert c_5\rvert$
0	0	0	$\lvert c_4\rvert$	$\lvert c_5\rvert$

(e)

$M_{distinct}$

$\lvert c_1\rvert$	$\lvert c_2\rvert$	0	0	0
0	0	$\lvert c_3\rvert$	0	0
0	0	0	$\lvert c_4\rvert$	$\lvert c_5\rvert$

(f)

Figure 4.4: Extraction of distinct rows.

guarantees that no centroid pair containing i or j gets selected by the filter again. In order to realize this removal, H^{III}_{dist} is aggregated into a binary row matrix H_{rem}. Each of its elements with a value of 1, indicates a column that took part in the last merge (10). Next, H_{rem} is expanded to a square matrix H^I_{rem} that has elements with value 1 throughout all columns that are to be removed. In the next two lines, H^I_{rem} is made symmetric by adding its transpose, before *sgn()* is applied to make it binary again. The resulting H^{III}_{rem} contains a 1 in every element that has to be removed from H^x_{dist}. This matrix is inverted by subtracting it from a matrix of ones with matching dimensions, which yields H^{IV}_{rem}. This matrix is multiplied element-wise with H^x_{dist} to create H^{x+1}_{dist} for the next iteration of the loop. Our described procedure only leaves those distances in H^{x+1}_{dist} that are still valid candidates for merges. The whole loop ends either if H^{x+1}_{dist} only contains zeros, which means all centroids have been processed, or if the iteration count $x+1$ exceeds L, which means the number of allowed merges for the current iteration is reached. Example matrices for this part are shown in Figure 4.3(f)-(i).

After the loop finishes, the weights for averaging must be integrated. For this, we use the *agg()* function on C^I to create a row-matrix of cluster sizes H_{size}. Expansion is used, to match the dimensions of this row-matrix to H^I_{merge}. By applying element-wise multiplication of H^I_{size} and H^I_{merge}, we construct the merge matrix M that contains the merged pairs and their respective weights. Example matrices are presented in Figure 4.3 (j)-(l).

Although matrix M contains all necessary information for the calculation of the merged centroids, it still has the same number of rows as R^I due to duplicates. Carrying out the centroid update by multiplication with the current M, leads to correct centroids, but does not reduce their number correctly. In order to tackle this issue, we have to get rid of the duplicate rows of M. For this purpose, we define the subroutine *distinct* which extracts all unique rows of a matrix. The subroutine is basically a modified repeat-until loop and is depicted in Algorithm 5.

The input matrix M of the subroutine enters the loop as working version M^x that is modified during iterations. Processing starts by aggregating M^x into a row matrix of cluster sizes H_{select} (2). A maximum filter is deployed to select the biggest cluster and results in $H^I_{select}(3)$. Subsequent application of $sgn()$ leads to the binary H^{II}_{select} (4). These steps are executed to implement a processing order for the rows of M^x as the for-each loop cannot be used here. By multiplying H^{II}_{select} with M^x, a particular row of M^x is extracted and becomes the first row $m_{i,*}$ of result matrix $M_{distinct}$. Example matrices for these steps are shown in Figure 4.4 (a)-(c), where the first unique row of example matrix M from Figure 4.3 is extracted. After getting a distinct row, all of its duplicates must be eliminated from M^x. For this, we first use *sgn()* to create a binary row $(m_{i,*})^I$, whose transpose is multiplied with $m_{i,*}$ to create a square filter matrix H_{filter}. By subtracting it from M^x, the processed row and its duplicates are set to zero, which effectively excludes them from further processing in the loop.

The resulting matrix M^{x+1} enters the next iteration, where the next unique row is extracted. When M^{x+1} becomes a zero matrix, all unique rows have been extracted and the loop ends with the output of the re-assembled $M^{distinct}$. Examples of H_{filter}, M^{x+1} and the final $M^{distinct}$ can be found in the bottom row of Figure 4.4.

Algorithm 5 Distinct Function: *distinct(*M*)*

1: **repeat with** M^x
2: $agg(M^x) \rightarrow H_{select}$
3: $filter(H_{select}, \left\langle h_{select(ij)} = max(H_{select}) \right\rangle) \rightarrow H^I_{select}$
4: $sgn(H^I_{select}) \rightarrow H^{II}_{select}$
5: $H^{II}_{select} \cdot M^x \rightarrow m_{i,*}\ of\ M_{distinct}$
6: $sgn(m_{i,*}) \rightarrow m^I_{i,*}$
7: $(m^I_{i,*})^T \cdot m_{i,*} \rightarrow H_{filter}$
8: $M^x - H_{filter} \rightarrow M^{x+1}$
9: **until** $\sum M^{x+1} = 0$ **output** $\rightarrow M_{distinct}$
10: **output** $M_{distinct}$

When the subroutine finishes, the only task left is create the updated centroids by executing *updt()* (21). The function uses $M_{distinct}$ and R^I as input and calculates R^{x+1} as weighted averages with the information from both inputs. After that, the merge branch is completed and the main loop in the trunk part begins its next iteration with R^{x+1}.

DBSCAN

So far, our described examples belong to the class of partitioning algorithms. In contrast, our next example DBSCAN [21] is a density-based clustering algorithm. This class of methods defines clusters as dense regions in space that are separated by regions of lower density. DBSCAN uses two user-defined parameters ε and $minPts$ to define a density threshold. With ε a neighborhood is defined around each point p. If this neighborhood contains at least $minPts$ additional points, p is considered as member of a dense area, i.e. a cluster and is labeled as *core-object*. Each core object is associated with all members of his neighborhood. In order to create clusters, core-objects with overlapping ε-neighborhoods are merged. Typically, this merging is done recursively, i.e. if p is a core-object each member of its ε-neighborhood is checked for the density condition and linked to p.

The fully transcribed version of DBSCAN is shown in Algorithm 6. Although the evaluation phase looks identical to the previously described algorithms, DBSCAN is different in this phase. While k-means and ISODATA work with distances between all points and a small set of representatives, DBSCAN calculates the distances between all points, which means $P = R$. As the selection phase requires row-wise traversal again, we use a for-each loop $(7 - 11)$. Before we start with the actual selection, we have to apply a small fix to D. As $P = R$, D contains the distances between all points, i.e. the main diagonal only contains values of 0 as it represents the distance between identical points. Since 0 is treated as a neutral element, the association ignores the corresponding points which leads to incorrect clusters. To solve this, we multiply an appropriate identity matrix with the scalar $\frac{\varepsilon}{2}$ and add the result to D $(5-6)$. With this, we can guarantee that the main diagonal will be populated after selection. Now we move on to the body of the loop. First, the loop uses a filter to remove all distances that are bigger than the ε-neighborhood (8). Next, we have to check the number of members in each neighborhood, which requires some preparation. First, we use $sgn()$ to make the current row $d^I_{i,*}$ binary, which eases member counting (9). A second filter is deployed to neutralize all binary rows whose sum of components does not exceed *minPts* i.e. the required neighborhood size (10). If

Algorithm 6 DBSCAN

phase EVALUATION
  $dist.L_2(P, R^x) \rightarrow D$ ▷ $\mathbb{M}^{n\times f} \times \mathbb{M}^{n\times f} \rightarrow \mathbb{M}^{n\times n}$
end phase
phase SELECTION
  $\frac{\varepsilon}{2} \cdot I^{n\times n} \rightarrow I^I$
  $D + I^I \rightarrow D^{II}$
  for each $d_{i,*}$ **of** D^{II} **do**
    $filter(d_{i,*}, \langle d_{ij} < \varepsilon \rangle)$
    $sgn(d_{i,*}) \rightarrow d^I_{i,*}$
    $filter(d^I_{i,*}, \left\langle \sum_{j=0}^{n} d^I_{ij} \leq minPts \right\rangle)$
  end for $\rightarrow D^{III}$
end phase
phase ASSOCIATION
  $assoc(D^{IV}) \rightarrow A$
  $resolveT(A) \rightarrow C$
  $distinct(C) \rightarrow C_{distinct}$
  $C^T_{distinct} \rightarrow C_{distinct}$
end phase

Algorithm 7 Resolve Transitivity Function: $resolveT(M)$

repeat with M^x
  $M^x \cdot M^x \rightarrow M^I$
  $sgn(M^I) \rightarrow M^{x+1}$
until $M^x = M^{x+1}$ **output** $\rightarrow M^{x+1}$

a neighborhood has not enough members, all elements of the corresponding row are mapped to 0. With the selection phase done, association starts with the familiar application of $assoc()$ (14).

After that, we face a novel challenge as $assoc()$ only associates each core-object and its ε-neighborhood. But in order to get the final clusters, overlapping ε-neighborhoods must be merged. We cannot utilize the originally proposed solution by recursion, as our matrix based building blocks do not offer this. Overlapping neighborhoods manifest in A in the form of transitive associations, e.g. $a_{12} = 1; a_{23} = 1$ shows that points $1, 2, 3$ are associated, but the connection of 1 and 3 is indirect as the explicit $a_{13} = 1$ is missing. To solve this issue, we create a subroutine *resolveT* that resolves transitivity and thus connects overlapping ε-neighborhoods (15). The subroutine is based on a repeat-until loop and is presented in Algorithm 7. It resolves transitivities by repeated multiplication of the input matrix with itself. After each multiplication the result is transformed into a binary matrix M^{x+1}, which becomes the input for the next iteration. If neither M nor M^{x+1} change anymore, transitivity has been resolved completely and has been replaced by direct associations. The output C contains the final clusters. Example matrices for this association are shown in Figure 4.5, where the first three columns of A^x show the indirect cluster assignment of p_1, p_2 and p_3. Although these points form a cluster, the direct adjacency of p_1 and p_3 is missing. After multiplication, all adjacencies are explicit in A^{x+1}. While this solves the problem of merging overlapping ε-neighborhoods into clusters, it also leads to a new problem, as C now contains duplicate columns/clusters. Since all adjacencies are explicit now, each member of a cluster is associated with every other member of this cluster, i.e. a cluster with 4 members manifests in 4 identical columns in C.

To get rid of the duplicates, we use the *distinct* subroutine that was already used in the merge branch of ISODATA. Matrix C is used as input and the resulting matrix $C_{distinct}$ contains only the unique

final clusters identified by DBSCAN (16). At the end of the association phase, $C_{distinct}$ is transposed as we want clusters to be represented in columns. After the association phase is completed, DBSCAN ends as it has no optimization phase or global loop.

AGNES

We have already described partitioning and density based clustering methods with our building blocks. To complete the three big classes of clustering algorithms, we chose a hierarchical method as our last example. Hierarchical Clustering seeks to create a hierarchy of clusters, from which a clustering solution is derived. There are two general strategies to create this hierarchy. Agglomerative approaches start by putting each point in its own cluster and build up a hierarchy by successively merging pairs of clusters. In contrast, the divisive approach starts with one cluster that contains all points and constructs the hierarchy by splitting it up into new clusters repeatedly. In this section, we transcribe the *Agglomerative Nesting (AGNES)* algorithm [46] which is a simple representative of the first strategy. This algorithm is a little different from the ones we transcribed so far as it actually starts with an initial clustering—each point in its own cluster—and creates not one, but a set of clusterings. Each clustering forms a level of the cluster hierarchy later on. We implement this behavior with our building blocks, by employing a global loop that outputs one clustering/hierarchy-level after each iteration. Actual dendrogram construction and hierarchy handling is not covered as we consider these tasks out of scope for our setting. The fully transcribed *AGNES* is shown in Algorithm 8, where we use the single-linkage criterion for the determination of distances between clusters. Example matrices are shown in Figure 4.6.

In the first line we initialize the clustering C as an $n \times n$ identity matrix, which assigns each point of the dataset to its own cluster. After that, the evaluation phase uses the familiar euclidean distance. Like for DBSCAN, $P = R$ also holds for AGNES, which means the dataset itself acts as reference. The upcoming phases of selection and association are surrounded by a repeat-until loop, which implements the repeated output of clusterings, necessary to build the hierarchy-levels. AGNES terminates if the hierarchy is finished and all objects are part of the same cluster. As only one pair of clusters is merged in each iteration, the loop has to be iterated $(n-1)$-times.

Upon entering the selection phase it is necessary to modify the distance matrix before applying filters. AGNES does not use representatives/centroids for distance measurement between clusters. Instead, the set of distances between all members of two clusters is evaluated. As there exist different ways to do this, the linkage criterion is used to specify which distance is selected to describe the similarity of both clusters. For our transcription we choose *single-linkage*, i.e. the distance between two clusters corresponds to the minimum distance between their members. After evaluation, D contains not only the desired inter-cluster distances, but also the uninteresting intra-cluster distances. These must be removed to ensure the correct execution of selection. To realize this, we use the current clustering and

$$
\begin{array}{c} A^x \\ \begin{matrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{matrix} \end{array}
\rightarrow
\begin{array}{c} A^I \\ \begin{matrix} 2 & 2 & 1 & 0 & 0 & 0 \\ 2 & 3 & 2 & 0 & 0 & 0 \\ 1 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 2 \end{matrix} \end{array}
\rightarrow
\begin{array}{c} A^{x+1} \\ \begin{matrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{matrix} \end{array}
$$

Figure 4.5: Resolving Transitivity for DBSCAN.

Algorithm 8 AGNES with minimum linkage.

1: $C^x \rightarrow I^{n \times n}$
2: **phase** EVALUATION
3: $dist.L_2(P, R^x) \rightarrow D$ ▷ $\mathbb{M}^{n \times f} \times \mathbb{M}^{n \times f} \rightarrow \mathbb{M}^{n \times n}$
4: **end phase**
5: **repeat with** C^x
6: **phase** SELECTION
7: $O^{n \times n} - C^x \rightarrow H$
8: $H \circ D \rightarrow D^I$
9: $filter(D^I, \left\langle d_{ij}^I = min'(D^I) \right\rangle) \rightarrow D^{II}$
10: **end phase**
11: **phase** ASSOCIATION
12: $(assoc)(D^{II}) \rightarrow A_{temp}$
13: $A_{temp} + A_{temp}^T \rightarrow A_{temp}^I$
14: $A_{temp}^I + C^x \rightarrow A$
15: $resolveT(A) \rightarrow C^{x+1}$
16: $distinct(C^{x+1}) \rightarrow C_{distinct}$
17: $C_{distinct}^T \rightarrow C_{distinct}$
18: **end phase**
19: **until** $x = n - 1$ **output** $\rightarrow \{C_{distinct}\}$

subtract it from a matrix of ones O with matching dimensions (7). In doing so, the obtained matrix H contains a 0 in each component that represent a pair of points inside a cluster. After element-wise multiplication of H and D all intra-cluster distances have been set to 0 in the distance matrix D^I (8). A filter with the zero-aware $min'()$ condition is used to select the smallest inter-cluster distance. The resulting D^{II} is passed on to association and contains only a single non-zero component. Examples for the first four matrices C^x, H, D^I and D^{II} are shown in Figure 4.6 (a)-(d).

Association starts with the familiar *assoc()* function, resulting in binary matrix A_{temp} where a component a_{ij} with value 1 indicates the merge between the two clusters that contain the objects referenced by i and j. To restore the symmetric character of association in AGNES we add A_{temp} and its transpose. After that, we add C^x in order to establish the new connection in the current clustering. We already used this approach in the merge branch of ISODATA. As the linkage criterion uses only a single object-to-object association to merge two clusters, we face the issue of indirect/transitive assignment again. Like with DBSCAN, we solve this by deploying the *resolveT* subroutine with A as input (15). This results in the clustering C^{x+1} that will be used for the next iteration. As with DBSCAN, the resolution of transitivity leads to duplicate columns in C^{x+1} which is solved again by application of the *distinct* subroutine (16). The resulting $C_{distinct}$ represents the unique clusters of the current iteration and a level of the clustering hierarchy. Examples for the different adjacency and clustering matrices are shown in Figure 4.6 (e)-(h).

While duplicate columns must be eliminated in $C_{distinct}$, it is necessary to keep them in the working matrix C^x in order to ensure the complete removal of intra-cluster distances from D during the further selection phases. After the end of the association phase, the algorithm enters its next iteration in the selection phase. Evaluation must not be repeated, as no changes to P or R occur, which also leaves D unchanged. Only the removal of intra-cluster distances changes with the evolution of C^{x+1}. After the selection and association phase are repeated $(n - 1)$ times, all objects are located in one cluster and AGNES terminates. Our proposed transcription of AGNES with single-linkage can also be modified for other linkage types. Maximum/complete linkage can be realized by adding a row-wise filtering to the selection phase. The implementation of average linkage requires additional building blocks in evaluation and the inclusion of the evaluation phase into the surrounding loop, as average calculation must be done on D in each iteration.

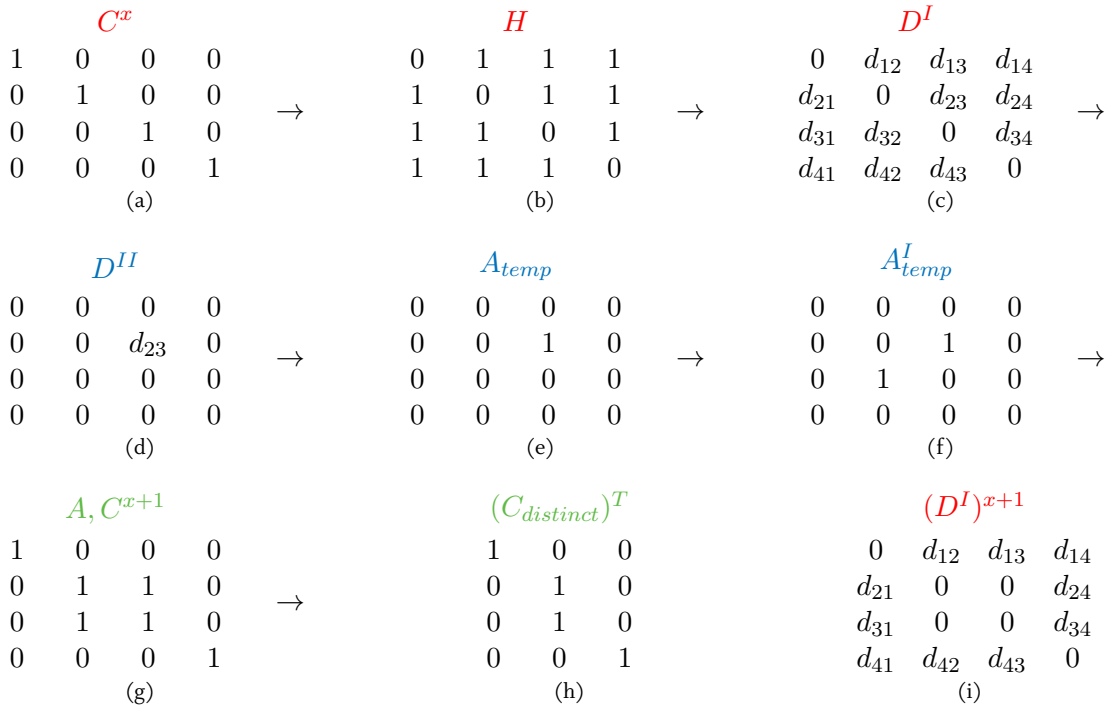


Figure 4.6: AGNES matrix sequence.

4.1.4 Summary

With our proposed building blocks, we developed a vocabulary for algorithm specification that is based on functions and matrices. The utilization of a mathematical syntax allows descriptions that are consistent and formal, but not overly specific like programming languages. Our building blocks still allow recognition of the abstract tasks an algorithm performs. Thus, our algorithm description represents a hybrid between the two extremes of verbal- and implementation-centered description we introduced at the beginning of this section.

Furthermore, the transcription of our example algorithms uncovered the existence of several description blocks that are used by multiple algorithms, e.g. removal of duplicates and resolution of transitivity. If such blocks or subroutines occur very often, they can be integrated in the vocabulary as higher-order building blocks, which makes transcriptions easier. For the description of four different algorithms we only needed to define the seven functions: *dist.L2()*, *sigma()*, *assoc()*, *avgDist()*, *updt()*, *agg()*, *subst()* as well as the two subroutines: *resolveT()* and *distinct()*. Obviously, many of these nine building blocks e.g. *agg()* will find use in further algorithm transcriptions. In addition to this, our building blocks can be used to evaluate the similarity between algorithms to a certain degree. Figure 4.7 shows a comparison of the descriptions made for DBSCAN and AGNES. Identical building blocks are highlighted in blue. It is easy to see that both methods are very similar. Evaluation and Association phase are nearly identical. Association differs only in two lines that are necessary to handle the iterative character of C in AGNES. The main differences between both methods, are located in the selection phase. This kind of block-wise view could also lead to the easy modular creation of new algorithms. For example, a new clustering algorithm could be created, by using DBSCAN as base and switch its selection phase with that of k-means. An additional potential use-case for our description vocabulary is optimization. Transcriptions of algorithms could act as intermediate programs that can be transformed into runnable programs by platform- or language-specific compilers.

Algorithm 6 DBSCAN

1: **phase** EVALUATION
2: $dist.L_2(P, R^x) \rightarrow D$
3: **end phase**
4: **phase** SELECTION
5: $\frac{\varepsilon}{2} \cdot I^{n \times n} \rightarrow I^I$
6: $D + I^I \rightarrow D^{II}$
7: **for each** $d_{i,*}$ **of** D^{II} **do**
8: $filter(d_{i,*}, \langle d_{ij} < \varepsilon \rangle)$
9: $sgn(d_{i,*}) \rightarrow d^I_{i,*}$
10: $filter(d^I_{i,*}, \left\langle \sum_{j=0}^{n} d^I_{ij} \leq minPts \right\rangle)$
11: **end for** $\rightarrow D^{III}$
12: **end phase**
13: **phase** ASSOCIATION
14: $assoc(D^{IV}) \rightarrow A$
15: $resolveT(A) \rightarrow C$
16: $distinct(C) \rightarrow C_{distinct}$
17: $C^T_{distinct} \rightarrow C_{distinct}$
18: **end phase**

Algorithm 8 AGNES with minimum linkage

1: $C^x \rightarrow I^{n \times n}$
2: **phase** EVALUATION
3: $dist.L_2(P, R^x) \rightarrow D$
4: **end phase**
5: **repeat with** C^x
6: **phase** SELECTION
7: $O^{n \times n} - C^x \rightarrow H$
8: $H \circ D \rightarrow D^I$
9: $filter(D^I, \left\langle d^I_{ij} = min'(D^I) \right\rangle) \rightarrow D^{II}$
10: **end phase**
11: **phase** ASSOCIATION
12: $(assoc)(D^{II}) \rightarrow A_{temp}$
13: $A_{temp} + A^T_{temp} \rightarrow A^I_{temp}$
14: $A^I_{temp} + C^x \rightarrow A$
15: $resolveT(A) \rightarrow C^{x+1}$
16: $distinct(C^{x+1}) \rightarrow C_{distinct}$
17: $C^T_{distinct} \rightarrow C_{distinct}$
18: **end phase**
19: **until** $x = n - 1$ **output** $\rightarrow \{C_{distinct}\}$

Figure 4.7: Similarities of DBSCAN and AGNES.

4.2 INTEGRATION

In order to provide versatility and manage the variety of available clustering algorithms, our clustering process needs the ability if integration. On the one hand this means that it must be able to work with different algorithms and different parameterizations. On the other hand, it must abandon the single-execution paradigm of traditional clustering and aim to integrate different single-execution results into a single, more robust solution. Shifting the focus of working to sets of multiple clustering configurations also has a beneficial impact on algorithm selection and configuration. These steps are no longer restricted to find a single optimal combination of algorithm and parameters, which puts less pressure on the user. After gathering these requirements for the integration task, we found the concept of ensemble clustering to be the natural choice for its realization. So far, the character of existing approaches for ensemble clustering is passive and rigid. They are simply used for the posterior aggregation of a set of clustering results, while the actual working focus stays with the traditional clustering algorithms. To implement our integration task, the role of ensemble clustering must be emphasized. It must become the focal point for working with the data, while traditional clustering is shifted to a subordinate position. To achieve this, it is necessary to develop novel approaches for ensemble clustering, of which we introduce two in this section.

4.2.1 Flexible Clustering Aggregation

In Section 2.3, we outlined ensemble clustering methods based on pairwise similarities. For a dataset P, consisting of n points $\{p_1, \cdots, p_n\}$ different clusterings with varying algorithms and parameters are created. These results $\{C_1, \cdots, C_e\}$ form a clustering ensemble E which is aggregated to form a consensus solution $\bar{C}$. Consensus is generated by evaluating the similarity of cluster assignments for each pair of points in each clustering of E. This similarity can take on two values: a_+ for pairs with

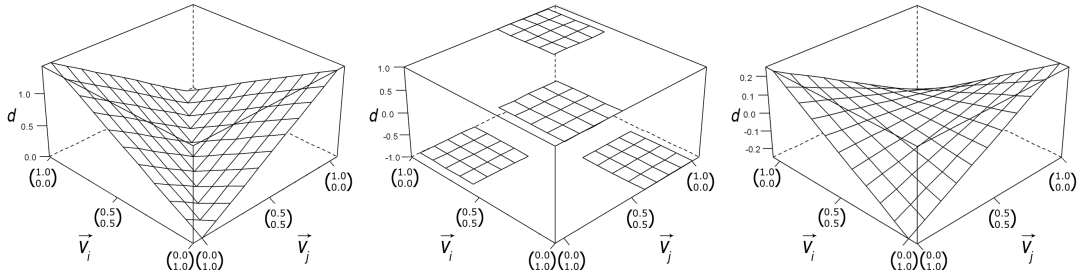
equal cluster assignments, and a_- for pairs with different assignments in C. With these binary values, co-association matrices are created for all clusterings. These are summed up and averaged to construct a global matrix that contains the relative frequency of a_+ for each pair of points throughout E. Those pairs that are clustered together in at least half of E are used to create the consensus solution $\bar{C}$.

While this approach has many benefits, it also exhibits the passive and rigid character we mentioned earlier. The aggregation process itself is not controllable and thus works in a quasi-autistic fashion. The construction of $\bar{C}$ can only be influenced indirectly by modifying the traditional clustering algorithms that make up E. In order to use this type of ensemble clustering for our integration task, we must establish a way for direct control of the aggregation. By developing a flexible clustering aggregation approach, we can relocate the focus of working to the level of ensemble clustering. In order to achieve flexibility and direct control, we first regard the aggregation input, where we introduce additional information. This allows the expansion of pairwise similarity values, which also improves their accurateness in general. By utilizing these enhancements, we revise the aggregation itself and establish direct control options.

Input Enrichment

Pairwise similarity values like a_+ and a_- for (p_i, p_j) are created by evaluation of their cluster labels. Identical labels cause a_+, while different labels lead to a_-. Such an approach requires absolute labels that represent the exclusive assignment to a cluster and are created by almost all traditional clustering algorithms. For further reference, we denote this type of assignment as *crisp*. Due to their nature, crisp assignments lead to a loss of fine-grained information regarding the similarities of an object to the clusters in general. A crisp label only states the cluster to which an object has the highest similarity and thus discards all similarities to other clusters. This becomes an issue if an object has an equal degree of similarity to multiple clusters. In such a situation, cluster affiliation becomes effectively undecidable and any crisp assignment would be inaccurate. To overcome this issue, soft clustering techniques like *fuzzy c-means* [10] and refinement techniques for crisp clustering results like a-posteriori [68] have been developed. Instead of a scalar label, cluster membership is indicated by a vector that contains the degree of similarity of an object to each cluster. Thus, soft cluster assignments contain more fine-grained information than their crisp counterparts. The utilization of this additional information during aggregation is the key to realize our flexible approach and fulfill the requirements of our integration task. We denote soft cluster assignments as vectors $\vec{a_i}$ with elements a_{il}, describing the relation between p_i and the l-th cluster of C.

In order to construct a consensus clustering from soft assignments, it is necessary to determine the similarity of a pair of vectors. A simple solution for this task is to assume that p_i and p_j are members of the same cluster if their assignments $\vec{a_i}$ and $\vec{a_j}$ are identical. However, this strict condition would greatly reduce the occurrence of a_+ pairwise assignments. Therefore, the constraint is eased from identity to similarity of assignment vectors. This principle is already employed by some existing ensemble clustering concepts for soft input sets [28, 68]. Both approaches use well-known distance measures—e.g. the euclidean distance in [28]—to calculate the similarity between vectors and derive pairwise-similarities. If the calculated similarity exceeds a certain threshold, the respective pairs are considered as a_+ or else as a_-. The major problem of these approaches is the use of common distance measures. We illustrate this problem by assuming the following example: a clustering C with $k = 2$ and a set A of 11 assignment vectors $\vec{a_i}, 0 \leq i \leq 10$ that represent different soft assignments, satisfying $\sum_{l=1}^{k} a_{il} = 1, 0 \leq a_{il} \leq 1$, and $\forall a_{il} = i/10$. As we want to examine pairwise similarities, we generate 121 vector pairs $(\vec{a_i}, \vec{a_j})$ via the Cartesian product $A \times A$. We start by applying the L_2 norm, i.e. the *euclidean* distance to $A \times A$. In Figure 4.8(a), the obtained results are shown. A vector pair $(\vec{a_i}, \vec{a_j})$ is specified via its position on the $\vec{v_i}$- and $\vec{v_j}$-coordinate axes, while the corresponding



(a) L2-norm (b) correlation coefficient (c) covariance

Figure 4.8: Different Distance Measures Applied to two-dimensional Vectors.

z-value represents the L_2 distance for this pair. For example, the pair $\vec{a_i}^T = (1, 0)$ and $\vec{a_j}^T = (0, 1)$ in the left corner of Figure 4.8(a) has a distance of $\sqrt{2}$. When measuring the distance between two vectors, L_2 only considers their norm, but not their direction, which is a major drawback in our scenario. It is possible that pairs $\vec{a_i}; \vec{a_j}$ have identical L_2 distances, regardless of p_i and p_j actually being members of the same cluster or not. For example, the pair $\vec{a_i}^T = (0.1, 0.9)$ and $\vec{a_j}^T = (0.3, 0.7)$ is located in cluster 2, i.e. the pairwise similarity is a_+. Assuming $\vec{a_i}^T = (0.6, 0.4)$ and $\vec{a_j}^\top (0.4, 0.6)$, shows that both points are members of separate clusters and their pairwise similarity is a_-. Although both examples have different pairwise similarities their distances are equally measured at $\sqrt{0.08}$ with L_2. Obviously this leads to incorrect decisions in the construction of $\bar{C}$, especially if thresholds or clustering algorithms are employed to evaluate pairwise similarities.

To overcome this issue, we examine distance metrics that take the direction / composition of vectors into account. At first, we look at the *Pearson correlation coefficient* (ϱ) assuming a_+ for positive and a_- for negative linear dependency between $\vec{a_i}$ and $\vec{a_j}$. In Figure 4.8(b), two pairs of separated planes are depicted, which illustrate the results of our experiment. Examination of vector pairs and their corresponding ϱ, confirms our assumption about the relation between the value of $\varrho(\vec{a_i}, \vec{a_j})$ and pairwise-similarity. The correlation coefficient offers two advantages: direction awareness and a direct relation between the pairwise similarity and the algebraic sign of the ϱ-value. Further inspection of Figure 4.8(b) shows gaps between the planes, which originate from pairs including at least one vector with zero variance ($\sigma^2 = 0$). The Pearson correlation coefficient is defined as the ratio of the covariance of two vectors and the product of their standard deviations. Therefore, $\sigma^2 = 0$ leads to a division by zero, which makes ϱ undefined. To avoid this problem, said division is excluded from ϱ, which leaves the *covariance*. Results for the application of covariance as distance measure are shown in Figure 4.8(c). The depicted behavior is similar to ϱ but has no undefined areas and shows continuous distance values. The two previous experiments have shown that assignment vectors with zero variance result in a special behavior of ϱ and covariance. While ϱ is not defined for these cases, the covariance yields zero.

From the clustering point of view, assignment vectors with zero variance are an interesting phenomenon. Such an assignment is defined as $\vec{a_i}, \forall a_{il} | a_{il} = \frac{1}{k}$ and states that the respective object p_i has an equal degree of similarity to all clusters of C. Basically, this makes determination of an explicit cluster affiliation impossible for this object. From now on, we refer to this special kind of assignments as *fully balanced* assignments. As it is impossible to decide to which cluster an object p_i with a fully balanced assignment belongs, it is also not possible to determine the pairwise similarity of any pair containing p_i.

Undecidable Pairwise Assignments

The existence of fully balanced assignments and the resulting issue of undecidable pairwise similarity, requires the expansion of the present notion of pairwise similarity. Until now, existing ensemble clustering approaches assume that pairwise similarity is dyadic, i.e. has two values: a_+ and a_-. To handle object pairs with undecidable assignments, an additional value must be defined for these cases. We denote this value as $a_?$, thus making pairwise-similarity triadic. In order to correctly determine the pairwise similarity for any pair (p_i, p_j) in a clustering, we need to know whether $\vec{a_i}$ or $\vec{a_j}$ are fully balanced. Typically, this is the case if each component of $\vec{a_i}$ equals $\frac{1}{k}$, but there is also an additional form of undecidable assignment that occurs with vectors having multiple maximum components. Assume an object p_i with $\vec{a_i}^\top = (0.4, 0.4, 0.2)$ for a clustering with $k = 3$ clusters. Although we can state, that p_i is not a member of cluster 3, it is impossible to specify whether the object effectively belongs to cluster 1 or 2. Therefore, this kind of assignment is undecidable too, and we denote it as *balanced* assignment. However, a vector $\vec{a_i}^\top = (0.6, 0.2, 0.2)$ containing multiple equal, but not maximum components a_{il} still shows a clear cluster association.

Based on this observation, we define a function $balanced(\vec{a_i})$ that tests if the cluster assignment of an object p_i is fully balanced or balanced.

$$balanced(\vec{a_i}) \begin{cases} false & \text{if } \exists! a_{il} \in \vec{a_i}(a_{il} = max(\vec{a_i})) \\ true & \text{otherwise} \end{cases} \tag{4.7}$$

If $\vec{a_i}$ contains exactly one maximum component, a clear cluster affiliation exists and $balanced(\vec{a_i})$ returns *false*. Otherwise, multiple maximal components exist and indicate a *balanced* or *fully balanced* assignment. Besides testing of decidability, it is necessary to evaluate whether two objects p_i and p_j belong to the same cluster of C. We assume that each p_i belongs to the cluster to which it has the strongest degree of similarity, i.e. we regard the maximum component of $\vec{a_i}$. If the maximum components of two assignment vectors $\vec{a_i}, \vec{a_j}$ are located in the same dimension, p_i and p_j belong to the same cluster. In contrast, objects with maximum components in different dimensions of $\vec{a_i}$ are also members of different clusters. We define a co-occurrence function $co-occur(\vec{a_i}, \vec{a_j})$ that states if a pair of objects (p_i, p_j) is located in the same cluster or not:

$$co-occur(\vec{a_i}, \vec{a_j}) \begin{cases} true & \text{if } \exists l(a_{il} = max(\vec{a_i}) \wedge a_{jl} = max(\vec{a_j})) \\ false & \text{otherwise} \end{cases} \tag{4.8}$$

By combination of the two proposed functions, it is possible to define a function *pSim* that determines the pairwise-similarity of any object pair in a given clustering C:

$$pSim(\vec{a_i}, \vec{a_j}) \begin{cases} 1 & \text{if } (\neg balanced(\vec{a_i}) \wedge \neg balanced(a_j)) \wedge co-occur(\vec{a_i}, \vec{a_j}) \\ -1 & \text{if } \neg co-occur(\vec{a_i}, \vec{a_j}) \\ 0 & \text{otherwise.} \end{cases} \tag{4.9}$$

If none of the involved objects features an undecidable cluster assignment and if both objects are clearly related with the same cluster of C, $pSim$ returns 1 which signifies a_+ for p_i and p_j. The pairwise assignment a_- is indicated by $pSim$ resulting in -1, in which case it is not relevant if the pair of objects contains undecidable assignments. Assume a clustering with 3 clusters: a balanced

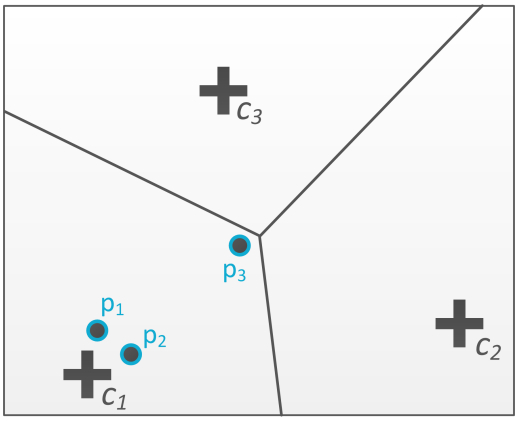


Figure 4.9: $\vec{v_i}$ with Different Significances.

assignment $\vec{a_i} = (0.4, 0.4, 0.2)$ and a clear assignment $\vec{a_j} = (0.1, 0.1, 0.8)$. Although it is impossible to state to which cluster p_i belongs, it is definitely not the cluster p_j belongs to. The result 0 indicates $a_?$ and is only obtained for object pairs that co-occur and contain undecidable assignments. With *pSim* the problems described at the beginning of this section are solved and pairwise similarity can be correctly determined for any arbitrary object pair.

Scoring Reliability

By definition, all values of our triadic pairwise similarity are absolute, which means that all pairwise similarities are equal regarding their significance. In the following section, we propose that in the context of our scenario, the calculated pairwise similarities for certain pairs of objects, can have different levels of reliability. Consider the example shown in Figure 4.9, which shows a clustering with $k = 3$ clusters and their respective centroids c_1, c_2 and c_3. The gray lines mark the borders of the area of influence each cluster has. An object located on those lines or at its intersection points has an equal degree of similarity with all adjacent clusters and thus a balanced or fully balanced assignment, respectively. The two depicted objects p_1 and p_2 have a very strong relation with c_1 and only negligible similarity with the remaining clusters. If our function $pSim$ is applied to the pair (p_1, p_2) it results in 1 resp a_+. Regarding the third object p_3, we can state that it still has the strongest degree of similarity with c_1, but also shows a nearly equal similarity with c_2 and c_3, which brings its assignment $\vec{a_3}$ close to being a fully balanced assignment. Application of $pSim$ to (p_1, p_3) correctly determines that both objects belong to the same cluster c_1. However, comparing both pairwise similarity results, we would intuitively say that a_+ stated for (p_1, p_2) looks more reliable.

This rather subjective way of evaluating the reliability of pairwise similarities takes two properties into account: First, the explicitness of cluster assignments $\vec{a_i}$ and $\vec{a_j}$, i.e. their dissimilarity to the fully balanced assignment. Second, the component-wise similarity of both assignment vectors. The degree of satisfaction of these requirements constitutes our notion of reliability. In the example depicted in Figure 4.9 the pairwise similarity of (p_1, p_2) exhibits a clear relationship to c_1 and high component wise similarity thus the determined a_+ is very reliable. Further examples of pairs that maximize both relevant criteria can be found at the corners of the planes shown in Figure 4.8. It is plausible to assume that, starting from these locations, reliability decreases when approaching the middle of the plane or one of its bisectors—the gray lines in Figure 4.9—where balanced or fully balanced assignments are located. The covariance partly shows this desired behavior in Figure 4.8(c) with high values at the

corners and low or zero values in the middle of the plane. Based on this observation, we derive a scoring function $score(\vec{a_i}, \vec{a_j})$, using the covariance as a template:

$$score(\vec{a_i}, \vec{a_j}) = \sum_{l=1}^{k} \left(|a_{il} - \frac{1}{k}| \cdot |a_{jl} - \frac{1}{k}| \right) (i \neq j, 1 \leq l \leq k) \tag{4.10}$$

For scoring, this function considers the component-wise distance from the fully balanced assignment and the similarity of the respective assignment vectors. If a pairwise similarity obtains a high score via this function, it has also a high reliability. With the proposed function, reliability values can only be evaluated in relation to other reliability scores. It is possible to state that the pairwise similarity a_+ for (p_1, p_2) is more reliable than for (p_1, p_3) but we cannot determine whether each of these values is reliable on an absolute scale. Absolute statements require the normalization of results to a fixed range. For our approach we want *score* to result in 1 if a pairwise similarity has maximum reliability. To achieve this, we examine those pairs for which maximum reliability occurs and derive a normalized function $nscore$ [34]. We begin by examining the most reliable a_+. Assuming a clustering with $k = 3$ clusters, this case is given for a pair of p_i and p_j with assignments $\vec{a_i}^T = \vec{a_j}^T = (1, 0, 0)$. For simplification we assume that components like $a_{il} = 1$ and $a_{il} = 0$ can occur, although the strict definition for soft cluster assignments demands $\forall a_{il} | 0 < a_{il} < 1$. With this, both assignment vectors are identical and also identical to the centroid to whose cluster they are assigned. Applying $score$ to this pair of assignment results in:

$$(|1 - \frac{1}{3}| \cdot |1 - \frac{1}{3}|) + (|0 - \frac{1}{3}| \cdot |0 - \frac{1}{3}|) + (|0 - \frac{1}{3}| \cdot |0 - \frac{1}{3}|) = \frac{2}{3} \tag{4.11}$$

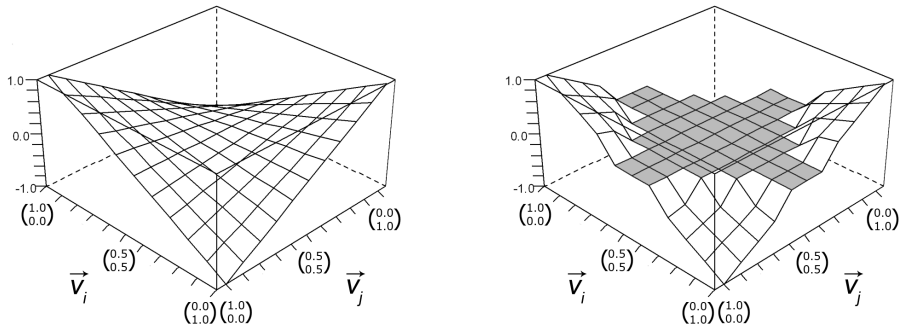
This shows that the maximum reliability actually depends on the number of existing clusters, which is a consequence of the characteristics of soft assignments. With increasing k the components of the fully balanced assignment $\frac{1}{k}$ decrease in value, thus the distance values from the fully balanced case change. In case of a maximum a_+, the pair of assignment vectors each have a single component with value 1 in the same dimension, while all remaining components are 0. Thus, the reliability score is made up of one summand $|1 - \frac{1}{k}|^2$ and $k - 1$ summands $|0 - \frac{1}{k}|^2$. Summarized, the maximum reliability of a_+ with respect to k is defined as $\frac{k-1}{k}$.

Let us now examine a_- with maximum reliability. Assuming the same setting as before, such a case occurs for a pair of assignment vectors that have a single component with a value of 1 that is located in different dimensions, e.g. $\vec{a_i}^T = (1, 0, 0)$ and $\vec{a_j}^\top = (0, 0, 1)$. For this pair $score$ yields:

$$(|1 - \frac{1}{3}| \cdot |0 - \frac{1}{3}|) + (|0 - \frac{1}{3}| \cdot |0 - \frac{1}{3}|) + (|0 - \frac{1}{3}| \cdot |1 - \frac{1}{3}|) = \frac{5}{9} \tag{4.12}$$

Again, the maximum score depends on k, the maximum reliability of a_- consists of 2 summands $|1 - \frac{1}{k}| \cdot |0 - \frac{1}{k}|$ and $k - 2$ summands $|0 - \frac{1}{k}|^2$, which can be summarized to $\frac{3k-4}{k^2}$. This shows that the maximum reliability of a pairwise similarity is also affected by the kind of similarity present for the pair that is evaluated. This means we have to find a way to integrate the result of $pSim$ into our normalization function, which is done by defining $pSimNorm$ as:

$$pSimNorm(\vec{a_i}, \vec{a_j}) = \frac{pSim(\vec{a_i}, \vec{a_j}) \cdot score(\vec{a_i}, \vec{a_j})}{norm};$$
$$norm = \begin{cases} \frac{k-1}{k} & \text{if pSim}(\vec{a_i}, \vec{a_j}) = 1 \\ \frac{3k-4}{k^2} & \text{if pSim}(\vec{a_i}, \vec{a_j}) = -1 \\ 1 & \text{if pSim}(\vec{a_i}, \vec{a_j}) = 0 \end{cases} \tag{4.13}$$



(a) $t = 0$ (b) $t = 0.3$

Figure 4.10: Results of $pSimNorm$ with and without Filtering.

whose result contains all the information we need regarding the pairwise-similarity of two objects. If the result is positive, the similarity of the pair is a_+ while a negative result indicates a_-. The normalized reliability of the pairwise similarity is expressed via the absolute value of the result. Our function yields 0 for pairs that contain undecidable assignments and are thus classified as $a_?$. Following our interpretation an undecidable pairwise similarity always has a reliability of zero, but reliability is actually not important for $a_?$ as it stays undecidable no matter what reliability score is obtained.

Controlling our Approach

In order to make ensemble clustering controllable, we integrate our proposed expansions into the aggregation procedure. The idea described by Gionis et al. in [25] is adopted as basic consensus procedure. Using our function $pSimNorm$, we determine the pairwise similarities of every object pair in all clusterings of E. When deciding on the pairwise similarity for (p_i, p_j) in the consensus result $\bar{C}$, we employ a majority decision and choose the pairwise similarity occurring the most for (p_i, p_j). If no majority can be identified, e.g. due to equal occurrences of different pairwise similarities, we assume $a_?$ for the corresponding pair in the consensus clustering because the final assignment is effectively undecidable. Although this method allows the construction of a consensus solution, it still lacks an option for control.

We introduce such an option by utilizing the provided reliability scores to filter all pairwise similarities. For this, a simple filtering function is defined that maps all results of $pSimScore$ whose absolute values do not exceed a certain threshold t to 0. With t a lower bound for reliability is specified that a pairwise similarity needs to pass in order to be reliable enough to be considered in the aggregation. All pairwise-similarities that do not satisfy this constraint are assumed as undecidable and become $a_?$. With filtering, it is possible to create an area of undecidability that allows us to mark not only balanced/fully balanced assignments as $a_?$, but also those assignments in their proximity.

In Figure 4.10(a), the results of our function $pSimNorm$ for the experimental setting from the beginning of this section are shown. We observe the desired behavior of maximum reliability scores at the plane's corners. Take for example the left corner with $\vec{a_i}^\top = (1, 0)$ and $\vec{a_j}^\top = (1, 0)$, the pairwise-similarity for this pair is a_+ with maximum reliability, so $pSimNorm$ yields 1. The reliability drops linearly towards zero as pairs approach the planes middle and its bisectors. Located at the center of the plane we find the pair $\vec{a_i}^\top = (0.5, 0.5)$ and $\vec{a_j}^\top = (0.5, 0.5)$, which contains two fully balanced assignments and is thus undecidable, i.e. $pSimNorm$ yields zero. When filtering is applied with a threshold of $t = 0.3$ an area of undecidability forms around the center of the plane and its bisectors, which is illustrated by the flat gray region in Figure 4.10(b). None of the object pairs in this area satisfies the filtering criterion, and hence they are classified as $a_?$.

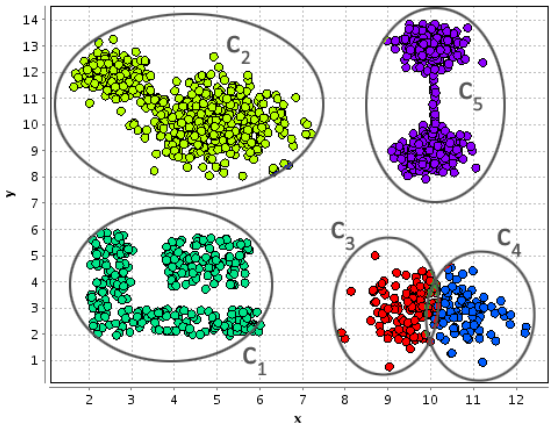
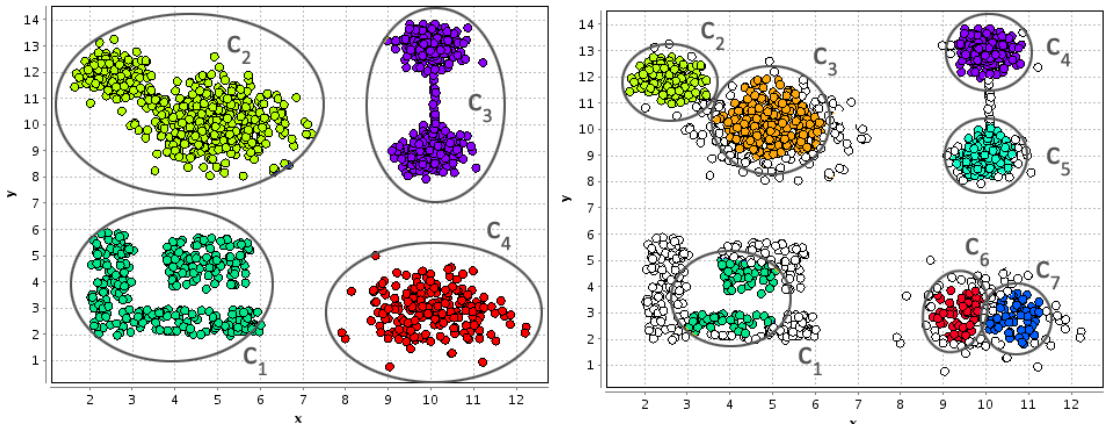


Figure 4.11: $\bar{C}$ Using Existing Pairwise Consensus Procedure.

So far, we proposed a way to determine a pairwise similarity value for each pair and an option to control the amount of $a_?$ via reliability filtering with t. In doing so, we divide our clusterings into stable cores made up of a_+ and a_- with high reliability that are robust against t and surrounding areas of undecidable $a_?$. These areas are the key to aggregation adjustments. To illustrate the workings of our control method, we again use the synthetic dataset introduced in Section 2. This data contains 2 clusters which are very close, but not linked and two cluster pairs connected via bridges of different length and width. Figure 4.11 depicts the consensus clustering, obtained by employing the consensus procedure introduced in [25]. The clustering-ensemble was generated using *k-means* with different parameterizations. Due to the characteristics of the dataset, it is very unlikely that single runs of *k-means* will produce the optimal clustering, even if many iterations with different parameters are made. The obtained consensus result shows five clusters, of which c_1, c_2, and c_5 might be divided further while the remaining two clusters should be merged. As this aggregation result is not optimal and requires adjustments, traditional aggregation approaches, force the user to: (i) modify parameters/algorithms of E, (ii) recreate E and the consensus solution, and (iii) evaluate $\bar{C}$ again until the desired adjustments occur.

With our approach, we use the same setup as before, but change the underlying clustering algorithm to *FCM* [10], a soft clustering version of *k-means* that produces the necessary soft cluster assignments. Concerning the handling of $a_?$, we have to regard two alternatives since we cannot determine if undecidable pairs are in the same cluster or not. Therefore, we define two strategies: one mapping $a_?$ to a_+ and another one that maps it to a_-. With this, we can control aggregation and adjust its result by modifying t and the handling of $a_?$, without changing E. We start by choosing $a_? \rightarrow a_+$ and increase the threshold t. When reaching $t = 0.1$, the result shown in Figure 4.12(a) is obtained, where the two clusters in the lower right have been fused. This fusion is caused by points located along the border between both former clusters. Having nearly equal affiliations to both clusters, the pairwise similarities typically have a low reliability. Therefore, many of them become $a_?$ when filtering is applied. Due to the mapping of $a_?$ to a_+, both clusters are connected. If t increases further, more clusters connect which will ultimately result in the unification of all points in a single cluster.

Next, we use $a_? \rightarrow a_-$ and start to increase t again. Until $t = 0.4$ the consensus result $\bar{C}$ does not change. At this point, an additional cluster forms that contains all objects the algorithm was unable to assign to a cluster because they are labeled as $a_?$ or a_- in all of E. Those objects are put into a *noise* cluster for convenience and presentation. Actually, each object is a singleton cluster for itself since no affiliations to other objects or existing clusters can be determined, which is a novel trait that cannot occur in existing aggregation approaches. Increasing t also leads to an increase of *noise*, especially in areas that are equally influenced by multiple clusters. When $t = 0.8$ is reached, the aggregation



(a) Result for $t = 0.1$ and *merge*

(b) Result for $t = 0.8$ and *split*

Figure 4.12: Several Aggregation Results.

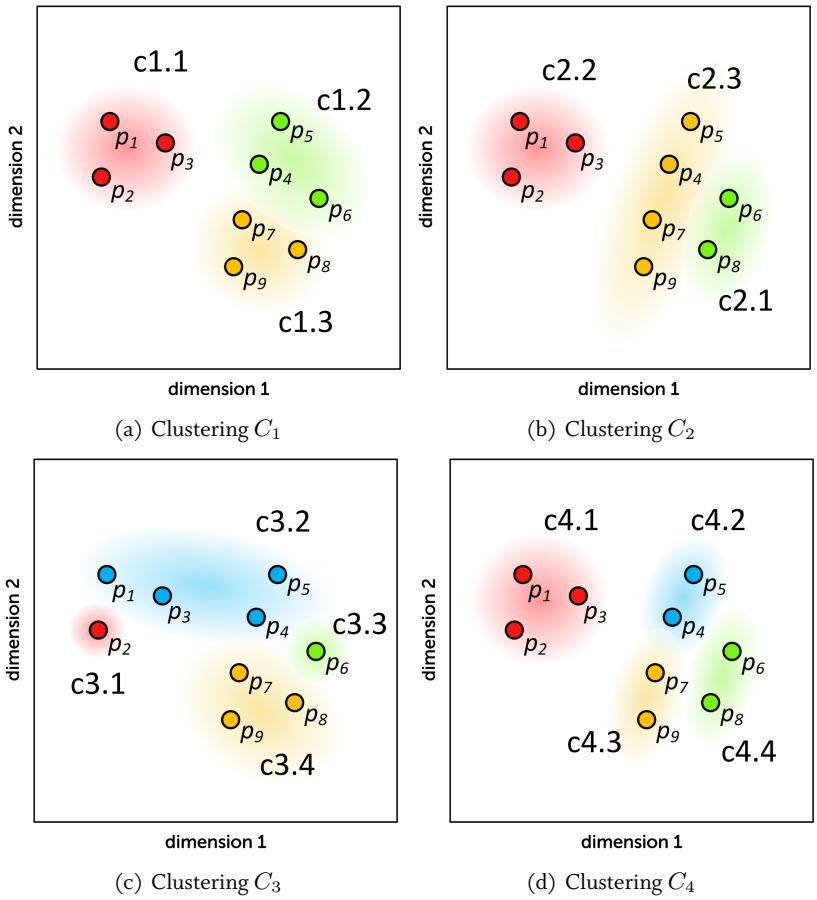
result depicted in Figure 4.12(b) is obtained, where *noise* is marked by circles without coloring. We notice that the clusters in the upper quadrants were divided by the *noise*. The reason for this division are clusterings of E that found multiple clusters in that area. With stronger filtering, more a_{-} occur along the border of those clusters and change the distribution of pairwise similarities throughout E until the multi-cluster partitioning takes hold in the consensus solution.

4.2.2 Frequent Groupings

Our flexible aggregation approach presents an implementation of our integration task that is based on pairwise similarities. The underlying idea of this approach is to perform a majority decision on cluster assignments. Although this brings many benefits, working with object pairs, i.e. the smallest groupings possible, can lead to certain drawbacks. As they present the basic elements for the construction of $\bar{C}$ it is possible that characteristics of larger structures are lost due to aggregation on this fine-grained level. Problems can for example arise from transitive effects. Assume an object pair (p, q) that occurs in the same cluster in half of E and a second pair (q, r) that occurs in the other half of E. Such a setting can lead to (p, q, r) being placed in the same cluster of the consensus solution, even if (p, r) never occurs in the same cluster in the whole ensemble. Due to the nature of majority decisions, all structural features that do not occur in the majority of clusterings are omitted although they might occur in a still considerable fraction. For these reasons we propose an alternative approach for consensus clustering, named *frequent groupings*. In contrast to the flexible clustering aggregation, where the most frequent pairwise assignments form $\bar{C}$, the frequent groupings approach identifies groups of objects that occur frequently in E and combines them into $\bar{C}$.

The Frequent Groupings Concept

Clusters of a consensus solution are sets of objects that are frequently assigned to the same cluster throughout the underlying clustering-ensemble E. To exemplify this, we use the small running example depicted in Figure 4.13. It is based on a dataset $\mathcal{P} = \{p_1, \ldots, p_9\}$ of nine objects in a two-dimensional feature space. For P, a clustering-ensemble $\mathcal{E} = \{C_1, C_2, C_3, C_4\}$ with four clusterings is created. Each one, exhibits a different number of clusters and a different cluster composition. In order to identify sets of objects that occur together frequently, the similarity of cluster assignments of



(a) Clustering C_1 (b) Clustering C_2

(c) Clustering C_3 (d) Clustering C_4

Figure 4.13: The clustering-ensemble of the running example.

objects must be evaluated throughout the ensemble. Our previously described aggregation approach implemented this by counting the co-occurence of object pairs. In contrast, the goal of frequent groupings, is the identification of larger groups of objects that frequently occur together in E. This problem definition is very similar to the task of frequent itemset mining [2], which is why we turn to this concept to derive the initial concept for our approach.

Frequent itemset mining assumes a set of n items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ and a set of transactions $\mathcal{T} = \{t_1, \ldots, t_m\}$. Each transaction has a unique id and contains a subset of $\mathcal{I}$. A set of items $\mathcal{X} \in \mathcal{I}$ is called frequent if its *support* exceeds a given threshold. The support of $\mathcal{X}$ is defined as the fraction of transactions of $\mathcal{T}$ that contain $\mathcal{X}$. At this point, the analogy to ensemble clustering becomes apparent. While frequent itemset mining tries to identify items that co-occur in many transactions, ensemble clustering searches for objects occurring together in the majority of clusters. Subsequently we map the concepts of $\mathcal{I}$, $\mathcal{T}$ and *support* to the domain of ensemble clustering in order to describe a method that allows the identification of *frequent-groupings*.

While the analogy of items $\mathcal{I}$ and the objects of the dataset is easy to see, matching of the transaction concept is intricate. As $\mathcal{T}$ is a set of transactions that contain elements of $\mathcal{I}$, an intuitive equivalent to $\mathcal{T}$ could be the clustering-ensemble E. In this case, the multiple clusterings of E would represent transactions, containing elements of the dataset. Typically, each clustering assigns all objects of a dataset to a cluster. If we assume the analogy of transaction and clustering, each transaction contains all items of P. This mapping completely prevents ensemble clustering as it effectively states that all objects occur together in each clustering. Thus, identification of prevalent groups of objects is made

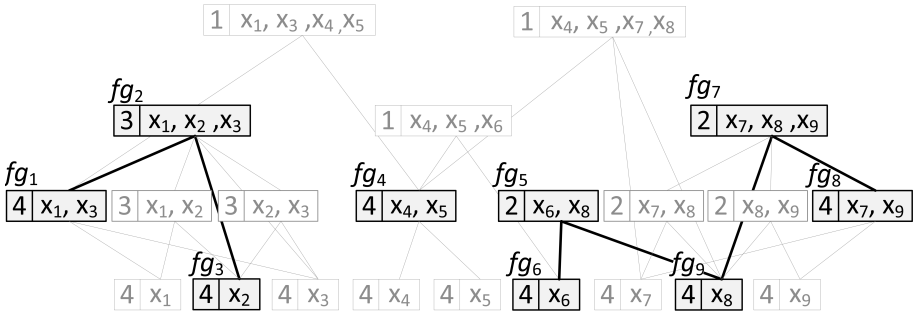


Figure 4.14: Structure of the frequent-groupings generated from the running example.

impossible. For this reason, we translate $\mathcal{T}$ as the set of *clusters* from all members of the clustering-ensemble. With this each cluster becomes a transaction, which is fitting as each of them contains a subset of the data. Based on this mapping, the *support* of a set of data objects $\mathcal{X}$ shows the fraction of clusters in the clustering-ensemble, in which $\mathcal{X}$ occurs. If $support(\mathcal{X})$ exceeds a certain threshold we call $\mathcal{X}$ a *frequent-grouping*. A high support of $\mathcal{X}$ also shows that this set of objects is robust because it was identified as part of the same cluster in many clusterings, regardless of the employed algorithm and/or parameters.

In terms of our running example, P acts as itemset while the clustering-ensemble E provides a set of 14 clusters/transactions. As transactions are required to have unique identifiers we label them in a specific format, e.g. $c1.2$ marks cluster 2 of clustering C_1. To simplify the calculation of support we make the following assumption: each $p_i \in \mathcal{P}$ is assigned to exactly one cluster in each $C_i \in E$. With this, an object can at most occur in one cluster, i.e. transaction per clustering. For the group of objects (p_1, p_2, p_3) from our running example this means a support of 0.75. The three objects occur in the three clusters $c1.1$, $c2.2$, and $c4.1$, which makes three out of the four clusterings that constitute E.

In order to create frequent-groupings, we have to specify a threshold for the support that is used to decide whether a group of objects occurs frequently or not. Following the assumptions of [25] we regard a set of objects as frequent, if it occurs at least in 50% of the clustering-ensemble. For our example this is given with two clusterings. The frequent-groupings obtained from the example are depicted in Figure 4.14 in the form of a graph structure. Each node represents a frequent-grouping and contains its associated objects as well as its support. Edges indicate subset/superset relations between different nodes. To build this structure, we initially create and insert a node for each cluster found in the ensemble. If a node already exists in the graph structure it is not inserted again, but the support of the existing node is increased by one e.g. the objects $\{x_1, x_2, x_3\}$ are contained in the clusters $c1.1$, $c2.2$, and $c4.1$, thus only one node with a support of three is generated. All nodes that are not frequent, i.e. have a support less than two are filtered—e.g. $c3.2$—and displayed in a faded grey. For each remaining frequent-grouping a new set of nodes that contains all of its possible subsets is created. At last, all nodes that are not *closed* are filtered. This means that each node that has a direct superset with the same support is removed from the graph. Eventually this procedure leads to the nine frequent-groupings $fg_1, \ldots, fg_9$ displayed in Figure 4.14. Please note that the described procedure only illustrates the formation of the depicted graph structure for our example. There already exist sophisticated methods for mining closed frequent itemsets that can be applied to generate frequent-groupings more efficiently [67].

If the obtained frequent-groupings are interpreted as clusters, a consensus clustering can be generated by combination of different clusters. This combination must ensure that each object of P is assigned to a cluster. As frequent-groupings overlap, multiple *alternative* combinations can be produced. In our running example, six alternative consensus clusterings can be created and are shown in Table 1.

Actual construction of these alternatives is described in the following section. The possibility to create multiple $\bar{C}$ is a novelty in comparison with existing aggregation approaches. With it, our approach incorporates the idea of alternative clustering we introduced in Chapter 2. The idea of alternative clustering is to create a clustering result with a traditional clustering algorithm and successively construct alternative solutions that are dissimilar to the initial clustering. In contrast, our alternatives are not created with dissimilarity to an initial solution in mind. They naturally result from the frequent groupings concept. Furthermore, the alternative solutions generated with our approach feature a certain degree of robustness. For each cluster of an alternative, a support-defined consensus exists throughout the ensemble. Thus, our approach represents a hybrid between alternative and ensemble clustering that combines the benefits of both domains, namely alternative results and robustness.

Complete Extraction of Alternative Consensus Clusterings

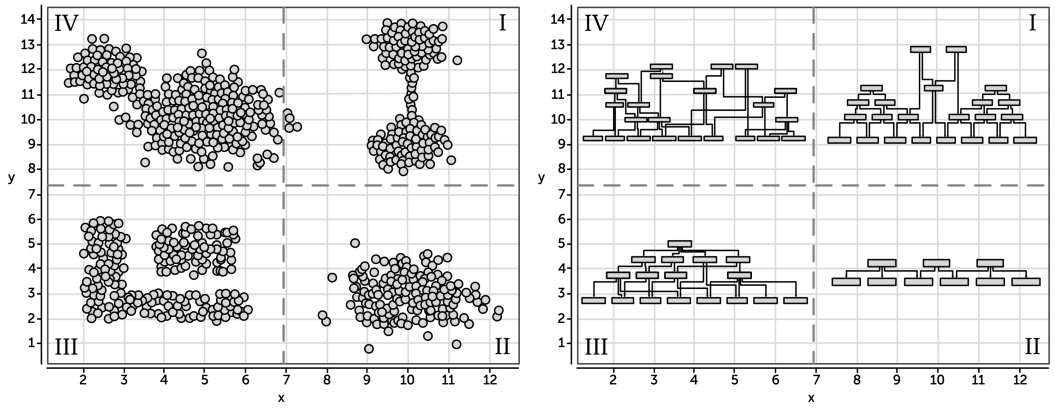
With the basics of frequent groupings described, we move on to specify ways for their identification and the creation of alternative clustering solutions. For the purpose of illustration we use a running example again. It consists of a synthetic two-dimensional dataset, containing ca. 1.500 points and is depicted in Figure 4.15(a). The roman literals shown at each corner of the scatterplot are used to reference the quadrants of the feature space. Although this is still a compact setting its scale is large enough to prevent manual processing—as in the introductory example—, but small enough to be comprehensible. We create a clustering ensemble of 10 different clusterings by using the *k-means* clustering algorithm with $k = 2, 3, 4, 5, 6, 7, 8, 9, 10, 15$ and a different initialization for each run. The dataset was crafted in order to provide structures that are hard to identify for partitioning algorithms like *k-means* e.g., the non-spherical clusters in quadrant III.

We already mentioned, the parallels of our frequent-groupings approach and the generation of frequent-itemsets in Section 4.2.2. Therefore, we examined existing algorithms for frequent-itemset mining in order to identify an efficient method for generating frequent-groupings. In our example scenario, the size of $\mathcal{I}$—1500 objects—is considerably higher than the size of $\mathcal{T}$, which contains the 69 clusters from the clustering-ensemble. We assume that in most clustering scenarios, the number of objects will be higher than the number of clusters, and thus chose to employ the *CARPENTER* algorithm[56] for the extraction of frequent-groupings, as it is optimized for such a setting. CARPENTER works by enumerating transactions and intersecting them. It also utilizes different pruning techniques to optimize its runtime.

In order to use CARPENTER for the computation of frequent-groupings, we extended the algorithm so that the particular support value is stored with each frequent-grouping. For our running example, we applied this method using a minimum support of 50% and obtained a set of 72 frequent-groupings. From these, we want to generate multiple clustering alternatives, which is done by interpreting the frequent-groupings as clusters that are combined into a clustering where *each* element of P is included in *exactly one* frequent-grouping/cluster. In a valid clustering alternative all clusters are disjoint and

A_1	$\{x_1, x_2, x_3\}\,\{x_4, x_5\}\,\{x_6\}\,\{x_7, x_8, x_9\}$
A_2	$\{x_1, x_2, x_3\}\,\{x_4, x_5\}\,\{x_6, x_8\}\,\{x_7, x_9\}$
A_3	$\{x_1, x_2, x_3\}\,\{x_4, x_5\}\,\{x_6\}\,\{x_8\}\,\{x_7, x_9\}$
A_4	$\{x_1, x_3\}\,\{x_2\}\,\{x_4, x_5\}\,\{x_6\}\,\{x_7, x_8, x_9\}$
A_5	$\{x_1, x_3\}\,\{x_2\}\,\{x_4, x_5\}\,\{x_6, x_8\}\,\{x_7, x_9\}$
A_6	$\{x_1, x_3\}\,\{x_2\}\,\{x_4, x_5\}\,\{x_6\}\,\{x_8\}\,\{x_7, x_9\}$

Table 4.1: Alternative consensus clusterings of the running example.



(a) Scatterplot of the data. (b) Connected components / stem clusters per quadrant.

Figure 4.15: Running example for our scenario.

their union contains the complete underlying dataset. Other definitions of clustering alternatives are possible, e.g. objects can have soft assignments with multiple clusters, but are not considered in our approach. The decision problem of combining frequent-groupings in a way that the given conditions are satisfied, is also known as the *exact cover problem* which belongs to the class of NP-complete problems.

Extraction Procedure As a method to find all solutions of an exact cover problem, Donald E. Knuth proposed *Algorithm X*[48]. This algorithm works with an incidence matrix, representing the specific problem of exact cover and uses a recursive, depth-first search with backtracking to find all possible solutions. Knuth also proposes a concept for the efficient implementation of Algorithm X called *Dancing Links (DLX)*[48], which we use to build all clustering alternatives. Application of DLX to the running example resulted in a very high number of alternative clusterings. With an input of 72 frequent-groupings the algorithm produces about 570.000 different valid clustering alternatives. Taking the scale of our example into account, it is safe to assume that the bulk of generated alternatives will be very similar.

The extent of this behavior results from the degree of overlap between the clusters of ensemble E and between the derived frequent groupings. In this context, very small and almost complete overlaps are critical as these are responsible for the creation of very small groupings. Assume a setting with E containing 2 clusterings and two clusters: $c1.1$ with 300 points and $c2.1$ with 305 points. If both clusters have a nearly complete overlap of, e.g. $|c1.1 \cap c2.1| = 298$, three frequent groupings would be identified: a single large one that represents the intersection of 298 points with a support of 100% and two very small groupings with 2 and 7 objects that represent the remaining points of the clusters, respectively, and have a support of 50% each. Now, we assume a very small overlap of $|c1.1 \cap c2.1| = 10$, again three frequent groupings are formed: one with 10 points for the intersection that has a support of 100% and two larger ones with 290 and 295 objects and a support of 50%. Considering the number of clusters and the fact that frequent groupings are also created from intersections of frequent groupings, the number of sets that contain only few points can get quite high. As the number of very small frequent-groupings increases, so does the number of clustering alternatives as more combinations are possible. In addition, the created results are likely to exhibit a high similarity as they only differ in the assignment of few objects. Typically it is desirable to extract only interesting $\bar{C}$ from the vast amount of possible alternatives. In order to reduce the number of generated alternatives and identify distinctive solutions, we introduce three straightforward methods:

Filtering: One possible approach is to prevent the formation of very small frequent-groupings before the actual alternatives are build. On the one hand, this reduces the number of components that are available for the construction of clusterings. On the other hand, the minimal frequent-grouping size is increased and generated alternatives become more dissimilar and potentially more interesting. There are two issues that need to be addressed: First, frequent groupings with very high overlap must be eliminated. This can be achieved by merging of such groupings into a single grouping. After this, the graph is rebuild and the number of small groupings decreases as the causing overlaps are no longer present. Second, small overlaps can be removed from the graph by deleting groupings that do not exceed a certain size. Unfortunately, filtering is troublesome to implement as optimal thresholds for size and overlap must be determined. In addition, each removal of a frequent grouping can lead to a situation in which it is not possible to cover each element of the dataset with a frequent-grouping. As a precaution, continuous tests for coverage are necessary. Furthermore, complex optimization decisions can emerge, e.g. if multiple groupings do not satisfy the thresholds, but not all of them can be removed without violating the coverage criterion.

Scoring: Another way of finding interesting alternatives is the application of quality or similarity measures to the generated alternatives. In doing so, all obtained solutions are ranked by a certain score and presented as a top-k list. For this approach, one or more appropriate measures for clustering-similarity and/or clustering-quality must be selected, which is challenging as the number of available metrics is high and their applicability in different settings is often unclear. Due to the high number of clusterings, the calculation of these measures needs significant computation effort. Furthermore, the top-k ranking requires the identification of an optimal k.

Greedy Top-k Selection: This approach is somewhat similar to the described *Filtering*, as it tries to limit the number of clustering alternatives by reducing the number of frequent-groupings. Instead of removing the smallest groupings, all frequent-groupings are sorted by their two essential characteristics, size and support, in descending order. From this sorting, the *top-k* frequent-groupings are chosen as input for the construction of clustering alternatives. If the value for k is chosen randomly, it cannot be guaranteed that the selected groupings will cover the whole dataset and one valid clustering can be constructed. In order to ensure at least one valid result, we used a kind of brute-force approach. Beginning with $k = 1$ DLX is executed and if no result is returned, k is incremented and the algorithm is run again. This is repeated until at least one clustering is returned. Although this approach is not very elegant, it can narrow the generated alternatives down to a number in the single digits. However, after continued testing we found the effectiveness of this method to be highly unpredictable and strongly dependent on the underlying clustering-ensemble. We applied the described procedure to the dataset from our running example and used different ensemble configurations. Some configurations returned 10 or less alternatives, while other configurations returned much larger result sets with up to 300.000 clustering alternatives.

All in all, the presented approaches have considerable drawbacks and do not offer an optimal way to reduce the number of possible results to a compact set of interesting clustering alternatives. For this reason, and the ensuing lack of user-friendliness, we do not consider complete extraction as the primary option for integration in our algorithm management.

Directed Extraction of Alternative Consensus Clusterings

Pure algorithmic extraction of all possible clustering alternatives proved to be not optimal due to various issues. So, we need to find an alternative to the automated *all-at-once* approach for the creation of clustering alternatives. For inspiration, we look back at our flexible clustering aggregation, where a starting solution is created that can be refined further by the user. We adapt this modus operandi

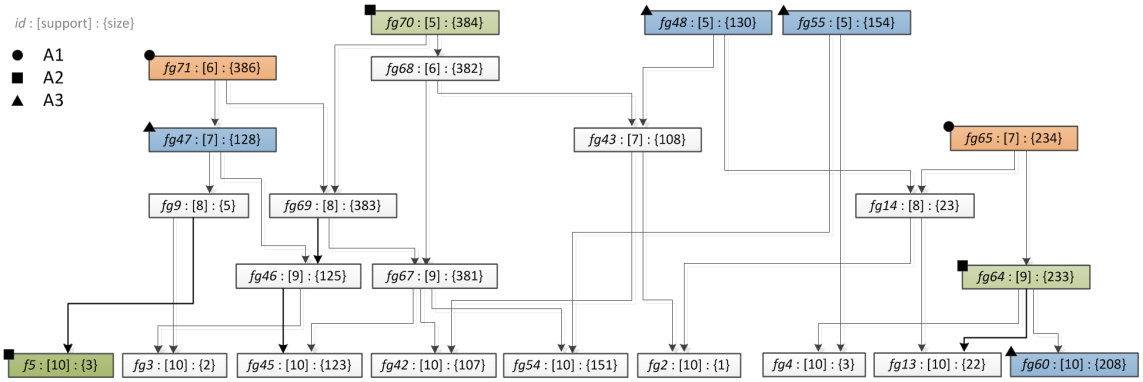


Figure 4.16: Frequent-groupings and clustering alternatives for quadrant 4 of the running example.

for our frequent groupings by providing an initial clustering alternative that acts as starting point for extraction. From this clustering further alternatives are successively created under the direction of the user. Guidance for this directed extraction is provided by the relationships contained in the graph structure of frequent-groupings, to which we refer as *fg-graph* from now on. Before we describe the directed extraction in detail, we introduce an example to illustrate our proposed methods.

The running example is based on the 72 frequent groupings that were generated from the running example in the previous section. All these groupings have a support of at least 50% and form the fg-graph shown in Figure 4.15(b) which consists of 4 connected components. Each of these components represents one quadrant of the example dataspace. This one-to-one correspondence is a coincidence that results from the clustering-ensemble used in our running example. Different ensemble configurations also lead to different connected components. A more detailed view of the connected component for quadrant IV of our running example is depicted in Figure 4.16. The display is similar to the graph we already described in Figure 4.14, but due to its larger size and complexity some changes had to be made. Again, each node represents a frequent-grouping and shows its attributes according to the pattern $id : [support] : size$. As the groupings have grown in size, the list of individual members is removed and only the number of members is displayed. To increase readability, the nodes are placed according to their support with the lowest at the top and the highest at the bottom. Thus, groupings that just satisfy the minimal support requirement and occur in 5 of 10 clusterings are found at the top tier, while groupings that occur in all clusterings of the ensemble are located at the lowest tier of the graph structure. Again, edges indicate subset/superset relations.

Constructing the Starting Point In order to establish our directed extraction procedure, we first have to construct a clustering solution that acts as starting point for the subsequent creation of clustering alternatives. We begin this construction with the connected components of the fg-graph. Each of these components contains all frequent groupings and thus all possible alternative cluster assignments for a part of the dataset. Based on those properties we also refer to connected components as *stem clusters*, because like stem cells they can differentiate into multiple cluster configurations. All *stem clusters* are disjoint and their union contains all objects of the dataset. Based on this, a clustering solution could be simply derived by adopting each stem cluster as a cluster. However, such a clustering would not be a valid result as it does not represent the occurrence of clusters in E correctly. Instead, it just unites a set of overlapping frequent groupings. In order to create a valid initial clustering, each *stem cluster* must be transformed into an explicit configuration of clusters.

With the frequent grouping concept, a clustering alternative can be created from an initial solution by substitution of an initial frequent-grouping with its subsets. Therefore, the frequent groupings of

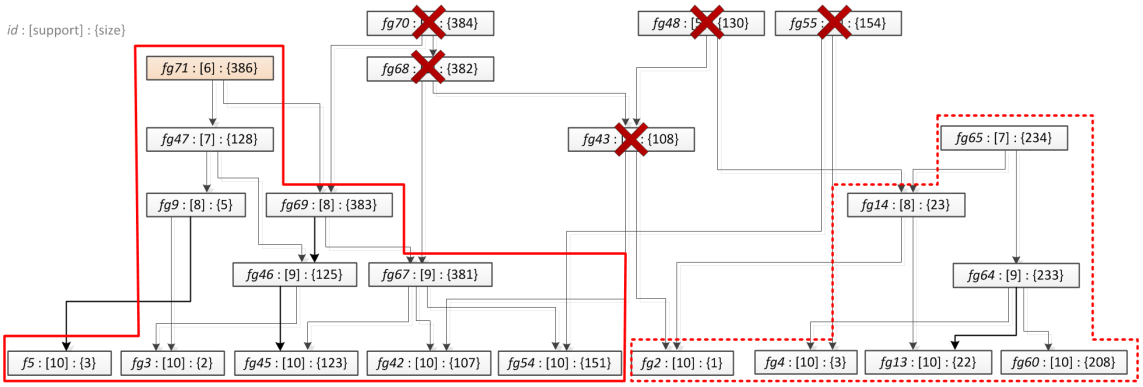


Figure 4.17: Transformation algorithm with starting point *fg71*.

the initial clustering should have a high number of subsets in order to ensure a maximum of possible alternatives. To ensure this, we select those nodes of a stem cluster that have no superset as clusters of the initial clustering. Due to this characteristic and their position in the fg-graph, we also refer to them as *roots*. Typically, *roots* are few in number and contain high quantities of objects. Initial clustering created from them, will normally feature a small number of big clusters. The stem cluster in Figure 4.16 has 5 *roots*, namely *fg71*, *fg70*, *fg48*, *fg65*, and *fg55*. In order to find explicit cluster configurations for the stem cluster, we apply a transformation algorithm to each of these roots. The transformation algorithm traverses the frequent groupings graph in a top-down fashion, starting at a provided root. Each run results in one explicit configuration, which means that the number of roots also determines the maximum number of alternative configurations for the stem cluster. If a stem cluster has exactly one root, then this root represents its single most general configuration and becomes a cluster for the initial clustering. Further transformation is not necessary in this case. If multiple roots exist, each one is processed by the following algorithm:

1. Select a root. (In the first iteration the root is provided. In further iterations the root with the highest size is selected.)
2. The selected root becomes a cluster for the configuration.
3. Mark the successor set of the selected root.
4. Delete all nodes from the graph that are not part of the current root's successor set but have a path into it. This is done to remove all remaining groupings that overlap with the selected cluster.
5. Delete the current root and its successor set.
6. If the graph contains no more nodes, the algorithm terminates and outputs the resulting cluster configuration. Otherwise the algorithm is repeated from step one.

The process of this algorithm is illustrated in Figure 4.17. It starts with the first of the four roots *fg71*, which is selected and becomes the first cluster of configuration *A1*. The successor set of *fg71* is marked, which is depicted by a frame in Figure 4.17. Nodes are deleted according to step 4 and marked by a cross in the figure. After the successor set of *fg71* is deleted only one connected component remains. The single root of this connected component is *fg65*, which is selected for the second iteration of our algorithm. After this iteration, the graph contains no further nodes and our algorithm terminates. The resulting cluster configuration *A1* contains the two clusters *fg71* and *fg65*. This configuration is created again, when the algorithm starts from root *fg65*. Applied to root *fg70*, the algorithm yields the

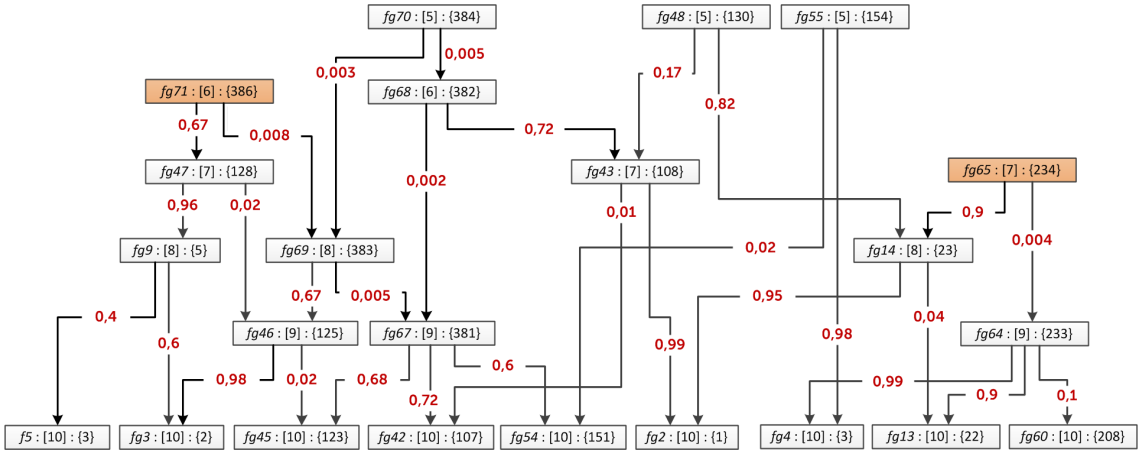


Figure 4.18: Fg-graph with initial clustering and weighted edges.

alternative configuration $A2 = \{fg70, fg64, fg5\}$. Further executions for roots *fg48* and *fg55* result in identical configurations, which we summarize as $A3 = \{fg48, fg55, fg60, fg47\}$. These results prove the predictability of our algorithm. From 5 roots of the stem cluster, 5 configurations are created of which 3 are unique. All unique configuration alternatives are shown in Figure 4.16, where they are marked with different symbols and colors.

After the differentiation into explicit clusters is done, a compact list of alternatives exists for each stem cluster. To create an initial clustering alternative that covers the whole dataset, one configuration must be selected for each stem cluster. As multiple initial clusterings are possible, we need to decide which one should be used as starting point for the directed extraction. One way to approach this decision, is to present the configurations for each stem cluster to the user and let him/her select one. However, this method is troublesome for the user especially if the number of stem clusters increases. Hence, we use a simple ranking to select a single configuration. Our goal is to find a starting solution that offers a maximum of possible alternatives, so we evaluate each configuration by counting the members of the successor sets of its frequent groupings. For our example this ranking yields 16 for $A1$, 13 for $A2$ and 12 for $A3$, which means $A1$ is selected as it offers the most potential for further alternatives. The reason for this is, that $A1$ only contains root nodes while the remaining configurations also contain frequent groupings that are placed on lower tiers of the graph, which reduces the number of subsets.

Directing the Extraction Now that we are able to derive an initial clustering, we turn to the actual extraction. The first extracted clustering alternative is always given by the initial clustering. This initial clustering is represented as a selection of nodes in the fg-graph. In order to create an alternative clustering, each of these nodes can be substituted with the subsets indicated by the edges of the fg-graph. With this in mind, our method of alternative creation can be pictured as a downward motion along said edges that starts at the nodes of the current clustering. This increases the number of clusters, as one grouping is always substituted with multiple subsets. Directed extraction is realized by giving the user control over this downward motion and thus allowing him/her to influence the extracted alternative. While this approach is easy to understand in theory, its actual application is challenging. There are two main issues that need to be addressed. First, the range of the downward motion must be considered, i.e. it must be decided, which subsets of a node are used for its substitution. Second, substitution cannot be done by only replacing a node of the fg-graph with its subsets. It has to adhere to the exact coverage criterion, which means that each object must be located in exactly one cluster of a clustering.

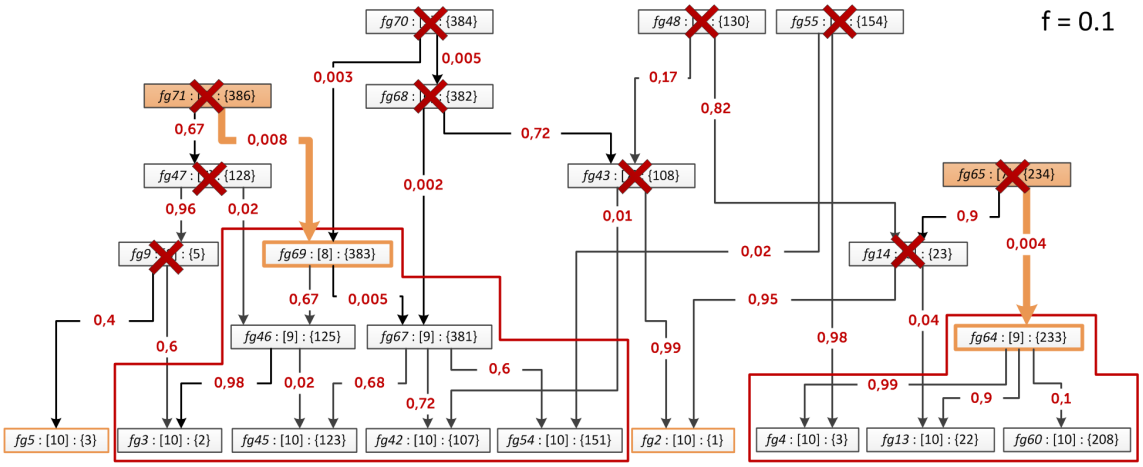


Figure 4.19: Extracted alternative with small f.

A simple and straightforward approach to the selection of subsets for substitution would be to choose the direct child of each initial node. However, this has some drawbacks. On the one hand, a node typically has multiple direct successors that can overlap. In this case, not all child nodes can be used for substitution due to the exact coverage constraint and hence the selection problem remains. On the other hand, the utilization of direct successors limits the effective range for each extraction to one hop in the fg-graph. This is disadvantageous, as subsets that differ only slightly from their initial nodes cannot be skipped in order to prevent alternatives with high similarity to the initial clustering. To enable a wider range selection and provide an estimation of the extracted alternative's similarity, we introduce edge weights to the fg-graph. These are calculated as

$$\omega = 1 - \frac{n_{child}}{n_{parent}}$$

where n_{child} is the size of the subset and n_{parent} is the size of the initial frequent grouping. This ratio indicates how much of the initial node is kept in the subset after substitution. By subtracting this ratio from 1, we get the fraction of objects that are covered by different nodes and thus an estimate on the difference to the initial clustering. The estimates expressed by these weights are rough and can be inaccurate, but are a sufficient indicator for similarity between initial and extracted solution. In Figure 4.18 the fg-graph for our running example is depicted with its respective edge weights and the initial clustering $A1$.

During directed extraction, the user specifies a value f between 0 and 1 to indicate how different the extracted alternative should be. Small values translate into a short downward motion in the fg-graph, while large values allow a higher range and are more likely to create alternatives with high dissimilarity to the initial solution. The process of selecting a frequent grouping for substitution of an initial node works as follows. Starting from the initial node, a path is created by successive selection of edges according to their weight. The provided f acts as a budget for traveling along this path and is consumed in each step. At each node, the edge with the highest weight—less or equal the current f—is chosen, f is reduced by this weight and we follow the selected edge to the next node, where the whole procedure is repeated. If f reaches zero or all outgoing edges of the current node have a weight that exceeds f, the path creation is finished and the current node/frequent grouping is selected as substitute for the initial node.

After the selection is finished, the clusters of the initial solution are substituted with the chosen frequent groupings. In order to create a valid clustering alternative, exact coverage must be assured. Substitution fixes certain clusters for the new clustering solution, but as these are subsets of initial

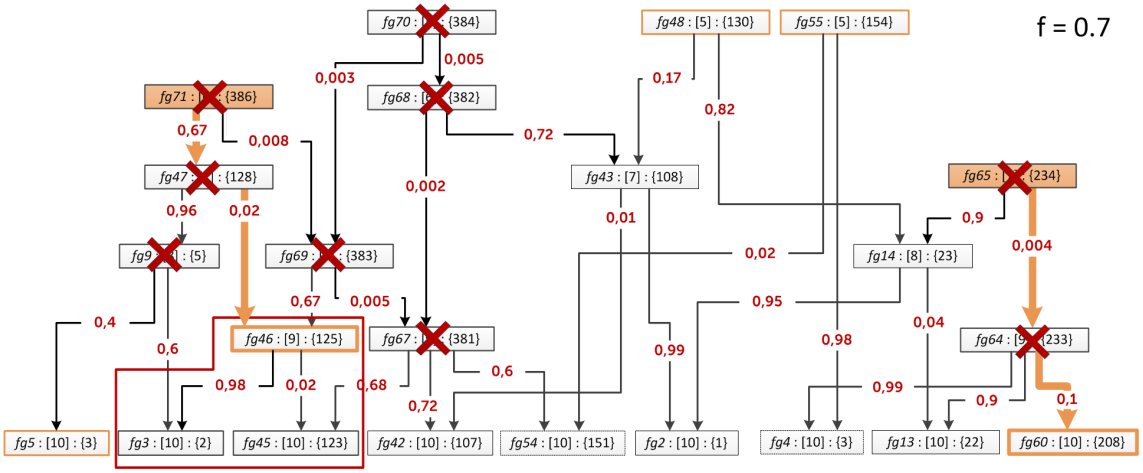
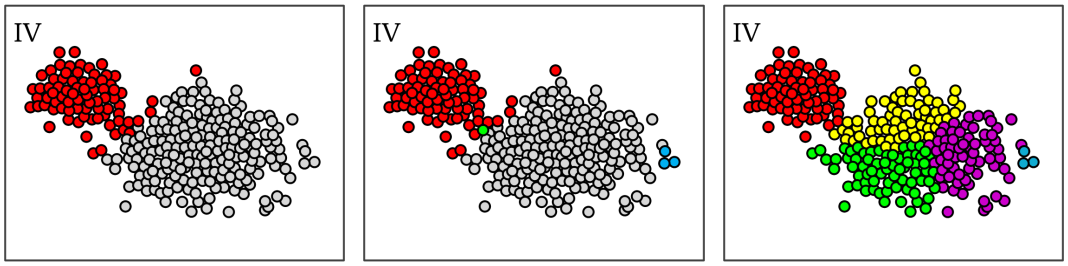


Figure 4.20: Extracted alternative with large f.

clusters, there is a remainder of objects that must also be assigned to clusters. For this assignment, we resort to the fg-graph and employ the transformation algorithm from the previous section again. To exemplify its course of action, we use our running example. We assume that the user wants to extract an alternative that is only slightly different from the initial $A1$ and sets the value of f to 0.1. In Figure 4.19 the corresponding extraction procedure and its result are shown. Subset selection starts at the nodes *fg71* and *fg65* of $A1$. Node *fg71* has two outgoing edges, of which only the one leading to *fg69* can be taken with the given f. We move to this node and decrease f by 0.008 after which we cannot go further since f is too small to follow any of the outgoing edges. Path-finding terminates and *fg69* is chosen as substitute for *fg71*. The same procedure is applied to *fg65*. The small f only allows to take the low-weight edge to *fg64*, where selection stops as f is depleted. With the selection of substitutes finished, we complete the new clustering alternative by applying our transformation algorithm. During the construction of an initial solution, the transformation algorithm was provided with a single starting node and selected further nodes for processing during execution. This time, the algorithm begins by processing all substitute nodes, before it continues to work independently. In our example, the algorithm starts by making *fg69* and *fg64* the first clusters of the new alternative. After that, their successor sets are selected—depicted by the red frames in Figure 4.19—and all nodes that have a path into these successor sets, but are not part of them, are removed. At last, *fg69* and *fg64* are removed along with their successor sets. With the substitute nodes processed, the algorithm starts to work autonomously to complete the clustering. Like before, it does so by selecting the largest remaining root of the graph. In our example, only *fg5* and *fg2* remain in the graph. Both nodes are roots as they have no superset, but they are also leafs as they have no successors. Because of its larger size, *fg5* is selected first, becomes a cluster and is removed after running through the known process steps. Subsequent, *fg2* is selected as the last cluster. The transformation algorithm terminates and produces the alternative clustering $\{fg69, fg64, fg5, fg2\}$. As intended, the small value of f results in a clustering that is very similar to *A1*. The two big initial clusters remain mostly unchanged and two very small clusters are introduced that only cover a total of 4 objects.

In order to extract a clustering alternative with higher difference, f is increased to 0.7, which allows a bigger range for substitute selection. The extraction is illustrated in Figure 4.20. Starting at *fg71*, the edge with the highest weight covered by f leads to *fg47*. There, the remainder of f is used to make a last hop to *fg46*. At this node f becomes insufficient and *fg46* is chosen as substitute. From *fg65*, selection finds a path to *fg60* which becomes the substitute as it is a leaf node and has no further subsets. The following application of the transformation algorithm adopts both substitutes as new clusters and removes their successor sets and linked nodes as before. After this is done, the remaining fg-graph contains three root nodes *fg5*,*fg48* and *fg55*. Since these nodes neither have overlapping successor sets



(a) Initial clustering (b) Alternative with $f = 0.1$ (c) Alternative with $f = 0.7$

Figure 4.21: Scatterplots of clustering from the running example.

nor paths into the successor set of a different root, the transformation successively adopts them as clusters. The whole procedure results in the clustering alternative $\{fg46, fg60, fg5, fg48, fg55\}$. For the purpose of better illustration, the initial clustering and the two extracted alternatives are shown as scatterplots in Figure 4.21. We can see that the directed extraction works as desired. Small values for f lead to alternatives that exhibit only negligible changes in cluster assignment in comparison with the initial clustering. In contrast, the clustering alternative in Figure 4.21(c) that was extracted with $f = 0.7$ shows a significant change in cluster composition.

Ways of direction So far, we described the specification of f as the only way to direct the extraction of alternative clusterings. Already, this enables users to extract different alternatives with different similarities to the initial clustering. The described extraction procedure is a top-down approach with an initial solution located at the upper tiers of the fg-graph. The user-guided downward motion/substitution creates alternative clusterings by increasing the number of clusters. In this, we find a resemblance to the direct control options of the flexible clustering aggregation. While subset substitution corresponds to the split handling strategy, the reliability threshold t corresponds to f. Actually, our concept of directed extraction can be further adapted to provide the same control semantics as the flexible clustering aggregation.

By applying directed extraction in a bottom-up fashion, it is possible to emulate the merge strategy and create alternatives by reducing the number of clusters. For this a different starting solution must be created that contains initial frequent groupings with many supersets. A convenient initial clustering for bottom-up extraction can be created by selecting all leaf-nodes of the fg-graph. These have only supersets and fulfill the coverage criterion. The selection of substitutes via edge weights and the composition of clustering alternatives with our proposed transformation algorithm still works smoothly in this altered setting.

User control can be expanded, by allowing not only the definition of selection range, but also its direction for each cluster of the initial solution. In doing so, the user gains full control and can create alternatives by merging and/or splitting certain parts of the initial solution. However, such a modus operandi requires an adjusted initial solution. It should contain initial clusters that have subsets as well as supersets, in order to provide options for combining and dividing clusters. While substitute selection and combination are realized as before, the sequence in which directions are given must be considered. The creation of an alternative by simultaneously issuing a superset substitution and a subset substitution for different initial clusters might result in a collision, as substitutes can overlap. One possible solution to this issue would be the introduction of methods for collision detection and handling. But this complicates the whole extraction procedure. The problem can be solved easier, by sequential execution of multiple substitution. Assuming the described scenario, where a superset substitution and subset substitution are issued in parallel, the extraction changes as follows. The simultaneous application is unraveled by first executing the superset substitution, which results in an

alternative clustering with a reduced number of clusters. This clustering can act as an intermediate and is used as initial clustering for the still pending subset substitution, which finally leads to a clustering that incorporates both desired changes. Switching the initial clustering between these steps is not a problem, as any valid clustering derived from the fg-graph can act as starting point for directed extraction.

4.2.3 Summary

Via integration, our process becomes versatile and can manage a variety of clustering algorithms. Instead of sticking to the traditional single-execution paradigm, integration works with sets of multiple clustering configurations. For the realization of this task we chose the concept of ensemble clustering and evolved it from a passive post-processing technique into an active and controllable method for working with the data. We described two novel methods that are able to fulfill the task of integration.

The first one—flexible clustering aggregation—works on the fine-grained level of object pairs. In order to create it, we incorporated soft cluster assignments into ensemble clustering and subsequently expanded the existing pairwise similarity concept. By developing a score for the reliability and handling strategies for undecidable pairwise assignments, we established a way for direct control of the aggregation procedure. Thus, the aggregation itself becomes the point of action for working with the clustering. As desired, adjustments to the consensus result can be executed without touching the underlying clustering ensemble.

As a second technique we proposed frequent groupings which realizes integration in a more coarse-grained way. While pairwise approaches construct a consensus solution from object pairs that occur together in the majority of the clustering ensemble, frequent groupings identify prevalent sets of objects and create a consensus by combination. This allows the construction of multiple robust clustering alternatives and tackles some drawbacks of pairwise methods. Our approach also fulfills the remaining requirements for our integration task. It features control options that are similar to the flexible clustering aggregation and allows direct adjustments of the consensus clustering. With this, it can act as a focal point for clustering execution and does not require the user to directly interact with the underlying clustering ensemble.

4.3 CONTROL INTERFACE

Controlling the clustering process is the major issue during the execution of a clustering analysis. It is the point of interface between user and algorithm, and thus has a crucial impact on the whole course of the clustering creation. As described in Chapter 2, control over the clustering is algorithm specific and often very challenging. In this section, we will describe the realization of the control interface for our algorithm management. This will somewhat differ from the previous descriptions. While the implementations of algorithm description and integration are done by methods that can exist on their own, the control interface is derived from them. Its character depends not only on the capabilities offered by the remaining elements of the algorithm management, but also on the used modus operandi. Subsequently are some basic characteristics that allow us to characterize a control interface are introduced and the control options provided by our clustering process are described.

Regarding the application of control, we can state that it can be issued at different points. Hence, each control interface offers different *levels* of control to a user. Control can also address different areas of

the supervised process, i.e. it has a certain *scope* of effect. Furthermore, control decisions made on a certain level can be enforced in different ways and thus express an individual *character*. Since one of the most important concerns of our work is smooth application of control by users, we also consider usability related characteristics of control. These contain the *support* available to the user and the minimal level of *experience* a user needs to work with the interface.

Our versatile clustering process strongly differs from the traditional clustering practice. Single algorithm execution and parametrization have been replaced by an ensemble of clusterings that is processed by our controllable integration. In addition, we established iterative refinement as the modus operandi of choice for our process. All these changes lead to a novel control interface that consists of the three levels depicted in Figure 4.22.

The primary level of this new interface is provided by our integration task and contains the control options offered by our proposed ensemble clustering methods. Our desired modus operandi requires that the control that is enacted on this level must have a local scope. This is a considerable contrast to traditional clustering, where control is enacted by changing individual algorithm parameters, in order to modify the clustering as a whole. These modifications have an indirect character that affects the functioning of the clustering algorithm and thus influences the created result. Our control options focus on the local scope, which means modifications are applied to individual clusters instead of the whole clustering. This can be implemented smoothly with frequent groupings and the flexible clustering aggregation. Frequent grouping can naturally realize control on the cluster level. As the whole approach works on the granularity of frequent groupings, i.e. clusters, it does not matter whether the available control options are applied to a set of frequent groupings or to a single frequent grouping. To enable the cluster scope for the flexible clustering aggregation, small adjustments are necessary. So far, all issued control decisions like filtering or $a_?$-handling where applied to all object pairs of the clustering. In order to limit the area of effect to individual clusters, we select all object pairs of the particular cluster and apply the control decisions only to this subset. If we recall the control options of both ensemble approaches, we see that they are identical. Both offer an option to reduce the number of clusters—$a_? \mapsto a_+$ or superset substitution—an option to increase the number of clusters—$a_? \mapsto a_-$ or subset substitution—and a value that represents the force with which these options should be enacted—t resp. f—that runs between 0 and 1. In contrast to traditional control, the character of these options is direct as they specify an effect and not its cause, which greatly eases adjustment. Let us assume that a user works with DBSCAN and wants to reduce the number of clusters with its traditional means of control. To realize that, his intention must be translated into appropriate values for ε and $minPts$, which requires full understanding of DBSCAN's functionality. By using the primary level of our control interface, translation into algorithm specific parameters is not necessary. In fact, integration acts as an abstraction layer for the algorithms of the ensemble. Regardless of the employed clustering techniques and their individual parameters, the user can adjust the result with the same general set of control options. This also renders the variety of additional methods for parameter selection unnecessary. Typically, each algorithm offers some kind of heuristic approach to choose suitable parameters. However, this does not guarantee optimal parameters and still leaves most of the decision for the user. All in all, the primary level of our control interface supports the user significantly by offering local, direct, and constant options for result adjustment. Effective handling of traditional algorithms and their control interfaces requires at least an intermediate level of experience. With our proposed setting, the necessary qualifications are lowered and even novices are enabled to use our control interface effectively.

The secondary level of our new interface partially resembles traditional control. It deals with the configuration of the clustering ensemble, i.e with the parameterization of a set of traditional algorithms. Like its traditional counterpart, this level has a global scope and an indirect character. But in contrast, it offers more support. Traditional control puts vital focus on finding an optimal set of parameters.

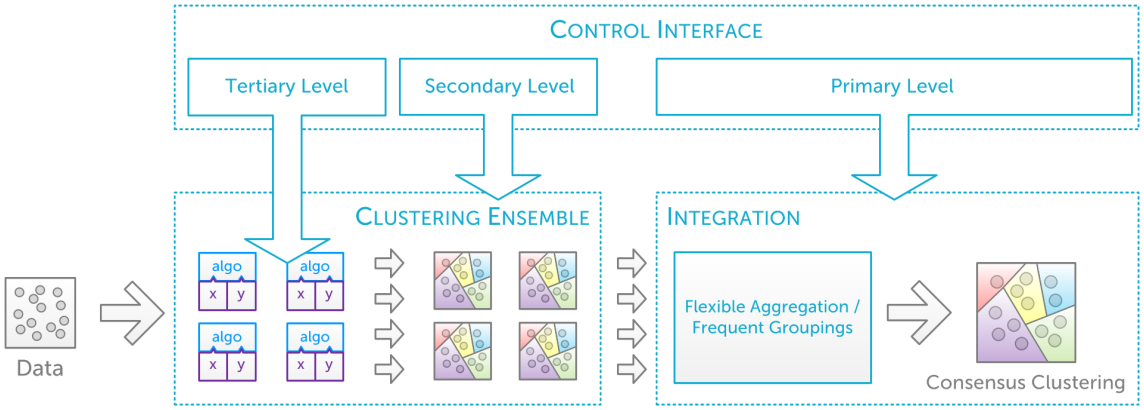


Figure 4.22: Overview of our control interface and its levels.

The parametrization of our clustering ensemble can be regarded as a sort of recipe to get a good starting point that can be refined further on the primary level. Due to this, parameter selection is no longer a do or die task. It is sufficient to use a fixed set of general parameter values, e.g. a set of parameterizations that result in 5 clusterings with $2, 4, 6, 8$ and 10 clusters. Basically, each parameterization that results in a non trivial clustering, i.e. a clustering with more than one and less than n clusters, can be considered as valid addition to the ensemble. The size of the ensemble can be varied depending on the application context, but has no influence on the clustering quality [23]. Besides parameter selection, we also have to consider ensemble composition. As multiple algorithms are used simultaneously it becomes possible to create heterogeneous ensembles. In order to decide which algorithms should be combined, our building blocks for algorithm description can offer support. By comparing building block descriptions, the similarity of available algorithms can be assessed and ensembles can be configured accordingly. Although the second level of control seems to be challenging, its impact is softened by integration. The goal of ensemble configuration and composition is to provide its input clusterings. Even if the results turn out to be not optimal, there is still the primary level which allows the refinement of the solution. If more quality is desired, experts could create several broadly usable configurations that would provide a stock of initial recipes. All these possibilities should enable even novice users to work on this level of control.

Although it might not seem obvious at first, there is an additional level of control that is more extensive and detailed than the previous two. Customization of algorithms allows to control a clustering result by specific adaptation of the method that creates it and forms our tertiary level of control, i.e. the secondary level of traditional control interfaces. As any kind of modification can be introduced to the algorithm, the scope and character of this level can only be described as individual. Therefore, it is mainly a domain for clustering experts. However, our approach for algorithm description can provide some support. In Section 4.1 we outlined the potential for creating new clustering algorithms by interchanging modules of existing building block descriptions. If this could be realized, it should allow users with intermediate experience access to this level of control.

In summary, our new control interface takes advantage of all the capabilities offered by our implementations of integration and algorithm description. Three levels of control offer a multitude of adjustment options and support the user during their application. The primary level directly interfaces with the ensemble clustering methods of integration and realizes the modus operandi that was desired for our clustering process. By using a compact and stable set of three direct and understandable parameters, integration acts as an abstraction layer for control and liberates users from dealing with individual clustering algorithms and their parameters. The more technical secondary level gives users control over the clustering ensemble. Taking advantage of the nature of ensemble clustering and our approach for algorithm description allows us to provide support for inexperienced users. Even the third level that addresses the expert domain of algorithm customization can be made more accessible by utilizing our building blocks approach.

4.4 SUMMARY

Algorithm management is responsible for the handling of all algorithm related tasks in our clustering process. This includes coping with the variety of existing algorithms and creation as well as modification of a clustering result. The realization of this goal requires the accomplishment of three tasks. First, a consistent way for algorithm description must be provided. For this, we identified the main phases and basic elements common to every clustering algorithm. By using a mathematical syntax based on matrices and functions, we created a set of building blocks for the consistent description of clustering algorithms. Our approach represents an intermediate between verbal and implementation-specific algorithm presentation and was illustrated during the transcription of several well-known clustering methods. Besides consistent descriptions, our approach also enables the comparison of algorithms and outlines the possibility of modular algorithm creation.

As a second task, algorithm management must be able to integrate different algorithms and clustering results into a final consensus solution. To accomplish this, we took the concept of ensemble clustering, and developed it from a passive post-processing method to a controllable focal point for the execution of clustering. We introduced two novel ensemble techniques. Flexible clustering aggregation is based on soft cluster assignments and the evaluation of pairwise assignments. Frequent groupings are derived from frequent itemset mining and create multiple alternative consensus solutions by combining groups of objects that often occur together. Both techniques offer equivalent control options that allow the adjustment of clustering results.

Offering a general interface for control is the third and final task of the algorithm management. It is realized by using the capabilities of the previous two tasks. The proposed interface contains three levels. On the primary level the user interacts with the ensemble techniques from integration. By using the universally valid set of three control options that they provide, users can directly control the clustering result and refine it iteratively. The second level of control deals with the configuration of the clustering ensemble that builds the foundation for the creation of the consensus clustering. Despite the technical nature of this level, our approaches for algorithm description and integration can be used to support the user in working on this level. This is also true for the expert driven tertiary control level that addresses algorithm customization.

With all our proposed methods and solutions, we can successfully realize the algorithm management and thus implement the first of our principles for a versatile clustering process. Now that the handling of algorithms is covered, we move on to the next chapter, in which we will discuss the interaction with the user. For this, we will create a visual-interactive interface that will work in unison with the proposed algorithm management.

VISUAL-INTERACTIVE INTERFACE

5.1 High-Level Feedback

5.2 Hybrid Visualization Concept

5.3 Visualization I: Large Screen

5.4 Visualization II: Small Screen

5.5 Summary

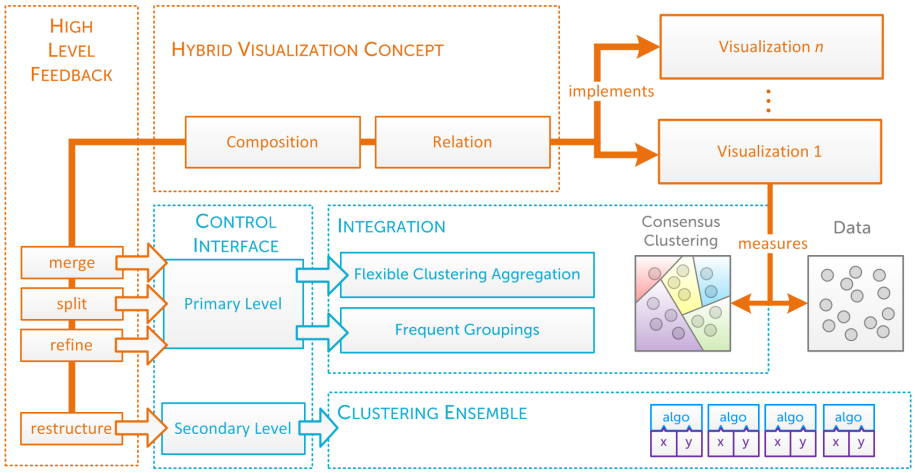


Figure 5.1: Overview of the visual-interactive interface.

Working with a clustering means to interact with the algorithms that produce it. We already outlined this at certain points during this thesis. While the algorithm management takes care of all aspects of clustering creation, the visual-interactive interface provides a solution for the communication and interaction between users and the clustering. In doing so, it handles the steps of result interpretation and adjustment, which makes it the second essential part of our versatile clustering process. Although the technical and methodical approaches of the algorithm management play an important role, the usability and applicability of our proposed process strongly depend on this interface. While clustering creation is mostly hidden from users, the solutions for interpretation and adjustment are in plain sight and present the focal point for working with the clustering.

An overview of our visual-interactive interface is given in Figure 5.1. Basically, it has to fulfill two main tasks. The first one is to provide an option for interaction that allows direct result adjustments. We realize this with a universal set of 4 high-level feedback operations: merge, split, refine, and restructure. These define an easy-to-use way for the user to address the underlying control interface of the algorithm management. From a user's view, feedback operations are fixed and independent from the techniques employed, as the control interface realizes the algorithm-specific mapping. The second task of our interface is to inform the user about the clustering itself and its relevant characteristics, which is a prerequisite for interaction and adjustment. To communicate this information, we introduce a hybrid visualization concept that is based on the visual information seeking mantra [60] and combines the benefits of traditional data-driven visualizations and quality measures. By focusing on the cluster level and the two basic characteristics of composition and relations, our approach provides clear and simple presentations of information. We designed the hybrid visualization concept as a template that can be realized in different visualizations to improve versatility. Both components—visualization as well as feedback—are tightly integrated and attuned to each other, to allow users the holistic handling of result interpretation and adjustment. In the following sections, we give a detailed description of our high-level feedback before we introduce the hybrid visualization concept and two of its implementations.

Parts of the material in this chapter have been developed jointly with Wolfgang Lehner and Dirk Habich and were published in [31], [30] and [33].

5.1 HIGH-LEVEL FEEDBACK

Although the control interface of the algorithm management offers three levels that can be used for the adjustment of the clustering result, it is not directly accessed by the user. The control options on each level are still technical and algorithm-specific to a certain degree, which makes them unsuitable for inexperienced users. For our clustering process we aim to establish a set of easy-to-understand feedback operations that is universally valid. These feedback options are abstract commands rather than specific parameters and are a fixed part of the visual-interactive interface. While a user issues a command, it should be of no concern to him, which actual algorithms are working underneath. Therefore, we must not only define our abstract commands, but also describe how they are mapped to the different levels of control offered by the algorithmic platform.

The idea of control via parametrization is deeply ingrained in the area of clustering and alternative approaches to interaction are not very common. An interesting, yet very theoretical approach was proposed in [7]. The authors show that a clustering can be transformed into any target solution by splitting and merging pairs of objects. Although their setting is only one-dimensional and provides the desired solution as guidance, the interaction approach itself is compelling due to its abstractness and expressiveness. The basic idea of this approach also builds the foundation for clustering with constraints. With methods like [15], the user provides 'must-link' and 'cannot-link' constraints for certain pairs of objects as additional parameters. These influence clustering creation and ensure that the respective objects are part of the same or different clusters, respectively. In summary, both approaches utilize the most basic decision of clustering to realize control and provide an option to group similar objects and separate dissimilar ones.

Based on this, we start the definition of our high-level feedback. In contrast to existing approaches, our feedback is aimed at clusters instead of objects, which makes its application more convenient for the user. We already mentioned that the general definition of clustering calls for high intra-cluster similarity in combination with high inter-cluster dissimilarity. Thus, a cluster must be adjusted if its members are too heterogeneous or if it is too similar to another cluster. This leads to our first two feedback commands *merge* and *split* that embody the base decision of clustering mentioned earlier. While these two options already allow many adjustments to a clustering result, we complement them with the feedback commands *refine* and *restructure* to provide the user with more convenience and versatility. In the following, we define each of our 4 universal commands and describe their integration with the different methods offered by the algorithm management.

merge If clusters are not well separated or if the created partitioning wrongly divides a natural accumulation of points in the data, the *merge* command is used for adjustment. With it, two separate clusters can be united into a single cluster. By choosing this option, a user reduces the number of clusters and states that the two merged clusters are too similar to justify a stand-alone existence. The goal of this adjustment is to create a new cluster with improved separation and homogeneity.

split This command is used for the adjustment of clusters with high internal heterogeneity. These occur if a clustering algorithm concentrates points with too many dissimilarities in the same cluster. By applying the *split* command to a cluster, a user decides that its members are not similar enough to be grouped together and breaks it up into more groups. The goal of this further division is to create a set of smaller clusters that better fit the data and feature an improved intra-cluster similarity.

refine Typically, clustering suffers from outliers in the dataset that manifest as loosely distributed points, slim chains of objects between large accumulations of points, small isolated object groups far away from the main cluster structures, etc. With the *refine* command, users can remove such outlying points from the scope of the clustering and relocate them to a global noise cluster. Basically, refine is a more specialized form of the split command. While the split command *splits a cluster up* and creates multiple new clusters, refine *splits certain points off* a cluster and labels them as noise, which excludes them from further adjustment steps.

restructure The three feedback commands merge, split and refine can be used to adjust the clusters of the consensus solution. Thus, they are implemented via the primary level of the algorithm management's control interface and established by using the information stored in the clustering ensemble. While this is a direct and convenient way to enact control, the potential for adjustments is dictated by the clusterings and clusters of the underlying ensemble. This can become a problem in some situations, when the ensemble does not contain the structural information necessary for the realization of the selected adjustments. Assume a consensus clustering, where result interpretation indicates that a certain cluster c should be split. Now assume that due to an inappropriate selection of clustering algorithms and parameters, each clustering of the ensemble locates all members of c inside a single cluster. This effectively prevents splitting via the primary level, as we cannot establish a split if not even one clustering of the ensemble sub-divides c. Such a case poses a dead end for the user as it is impossible to make adjustments with the given commands.

In order to escape such a situation, we have to adjust the clustering ensemble, i.e. address the secondary level of control. For this, we introduce the *restructure* command. With restructure the user can trigger a recursion of the clustering process on a selected cluster. This means that a new clustering ensemble and consensus solution are created for the members of the selected cluster and that this setting becomes the new scope of the clustering process. All remaining clusters from the parent clustering process are omitted and become inaccessible for the user. By creating a new ensemble for a subset of the data, new potential for adjustments on the primary control level become available and can be utilized by applying merge, split or refine. When the user is satisfied with the result for this part of the dataset, he can reintegrate this recursion back into the parent clustering. This is done by replacing the original cluster with the clusters created during the recursion. The reassembled clustering can then be adjusted like before, although merging between original clusters and restructured clusters is not allowed anymore, as these originate from disjoint ensembles. Besides escaping the mentioned trap situations, restructure can be used to execute a focused analysis of interesting parts of the dataset, for which the initial ensemble does not contain enough detail. As restructure addresses the clustering ensemble, it is implemented via the second control level. After selecting the target cluster, the user can select a particular ensemble configuration/recipe or apply the initial configuration to the subset of the data again.

This last command completes our universal set of high-level feedback commands. Our visual-interactive interface now provides the user with an easy-to-use facility for interacting with the clustering result that offers a wide range of adjustments. In the following, we will describe the implementation of our feedback commands in the algorithm management in more detail. This is necessary because merge, split and refine address the primary level of control, i.e our methods for integration. As both of the methods we proposed for integration work on different granularities, we have to provide individual mappings. The restructure command is not affected by this and is only realized in the described way.

5.1.1 Mapping to the Flexible Clustering Aggregation

The mapping of our abstract commands to the parameters of our flexible clustering aggregation can be made in a straightforward fashion. To realize a merge, the user has to provide the two clusters that should be combined, by selecting them in the visualization. With this, the relevant object pairs are identified and $a_? \mapsto a_+$ is chosen as handling strategy for undecidable pairwise assignments. Although complete parameterization requires the specification of a reliability threshold, we omit this input as it is not necessary in this context. When we apply these parameters to the whole clustering, the threshold makes sense as it can be used to merge only certain clusters. As mentioned in the description of the control interface, the scope of the primary level is local and, thus, affects only certain clusters, which partly nullifies the threshold's necessity. Even if the user provides a threshold, it is possible to ultimately force the merge of two clusters by setting $t = 1.0$, which effectively makes all pairwise assignments undecidable. Because of those reasons, we relieve the user from specifying t and use it at its maximum internally.

The mapping of split is done in a similar way. The user has to provide the target cluster for splitting and a value for the threshold t. While the selected cluster is used to determine the relevant pairs, t is used for reliability filtering. The actual splitting is done by choosing $a_? \mapsto a_-$ for the handling of undecidable pairwise assignments. Depending on the value of t, the results of split may vary. Although the splitting behavior depends on the individual characteristics of the underlying ensemble, we can assume a general rule for choosing t. Typically, satisfying splits are obtained with values for t from the range $[0.2 - 0.8]$ which represents the middle part of the full range for t. Splits with smaller t have a lower chance to actually divide the cluster, while higher values often produce a substantial amount of noise i.e. objects that cannot be assigned to any cluster.

To complete the mapping, we consider refine, which can also be easily mapped to the flexible clustering aggregation. The user selects the cluster he wants to refine and again specifies the familiar value between 0.0 and 1.0. Based on the soft assignments created during flexible clustering aggregation, all objects whose assignment to the current cluster does not exceed the user-specified threshold are relocated to the global noise cluster.

5.1.2 Mapping to Frequent Groupings

The implementation of feedback via our frequent groupings approach is a little more complicated in general, due to its more coarse granularity and the fg-graph. This already begins with the realization of the merge command. While flexible clustering aggregation works on the pair level and offers almost complete freedom when it comes to cluster merges, frequent groupings substitute whole clusters and are thus bound by certain constraints. To merge two cluster, i.e. frequent groupings, the fg-graph is searched for a superset that contains both clusters, which is used to substitute the two original frequent groupings. In contrast to the pairwise approach that can merge any pair of clusters, cluster substitution is bound by the fg-graph and has three possible outcomes: (i) a superset grouping that contains exactly both original clusters exists, (ii) a superset grouping that contains both original clusters and additional clusters exists, and (iii) there is no common superset. This means that not all clusters can be merged and that the merge result can be more than the simple unification of the two supplied clusters. To address this, we have to modify the user input. In order to prevent the user from selecting two clusters that cannot be merged, the fg-graph is checked in advance and only those clusters that are covered by a common superset are made available as choices in the visual-interactive interface. In addition, the user has to provide a value between 0.0 and 1.0 for the range budget f. This value is used to select a merge result if more than one superset, containing both original clusters,

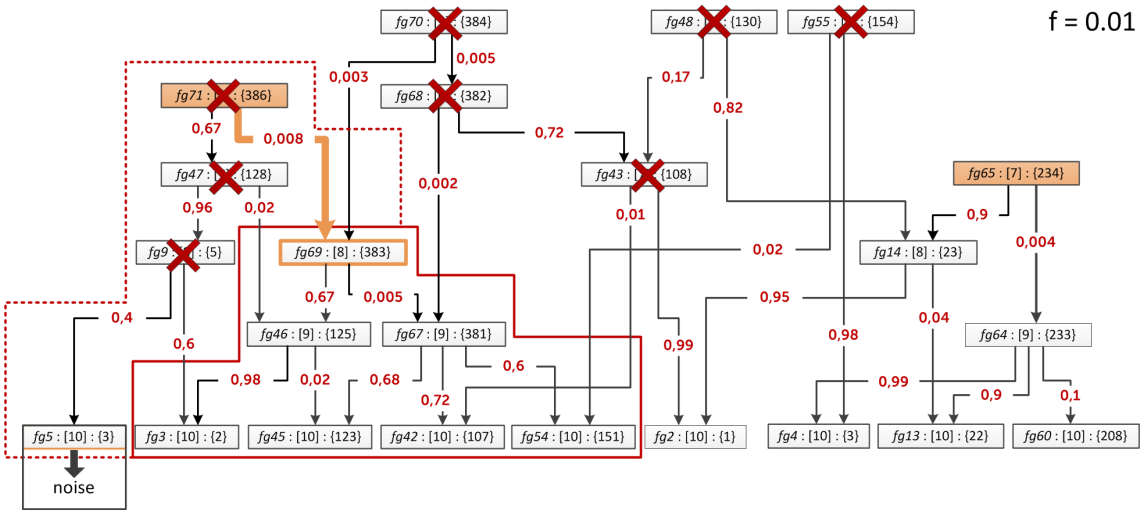


Figure 5.2: fg-graph for a refine implementation.

exist. A higher value for f translates into a farther upward movement in the fg-graph and will typically result in a merge with larger extent as bigger supersets are selected for substitution.

The implementation of split with frequent groupings is quite similar. The user-selected cluster is further divided by substitution with its subsets. Again, a value for f must be supplied to define the range of the downward motion in the fg-graph that is used to select the subsets for substitution. In contrast to the flexible clustering aggregation, points can always be assigned to a cluster regardless of the value chosen for f. Nonetheless, an equivalent to noise can occur in the form of very small frequent groupings/clusters with member counts in the single digits. With flexible clustering aggregation, noise occurs in the form of single objects that have no associations with other points, which makes automatic identification and relocation to a global noise cluster possible. This is mainly due to the fact that t represents a threshold for filtering on a given scale, while f defines the maximal weight/length of a path inside the fg-graph. Although this may seem like a noticeable difference, it has little to no impact on the interaction/behavior of the adjustments made. From a user's point of view, f and t are identical as they have the same domain and express the force with which the chosen adjustment is enforced.

Mapping the refine command to frequent groupings is complicated, as the introduction of noise disagrees with our goal of exact coverage. We handle this problem by assuming that noise forms a global cluster itself and adds to the coverage of objects from an algorithmic point of view. Regarding the implementation of refine, we want to keep things consistent and let the user provide a target cluster and a measure of force between 0.0 and 1.0. While soft cluster assignments express the strength of association between object and cluster and, thus, allow the straightforward removal of noise with low degrees of association, the frequent groupings concept is limited to the information contained in the fg-graph. We already know that outliers/noise manifest as very small clusters in this approach, which means refine is aimed towards the removal of these small clusters. Since refine is a variant of split, we implement it similar to the split operation and extend it with an option for the removal of noise. For exemplification, we recall the example from Section 4.2.2 where we introduced directed extraction. The fg-graph shown in Figure 5.2 is based on said example and illustrates how a refine is carried out for the cluster represented by *fg71*. We already described the character of f as the range for substitute selection, which expresses the amount of difference that can be expected after the split is carried out.

This means, a split carried out with $f = 0.5$ tries to split up the original cluster and assign at most 50% of its members to new clusters, according to the available frequent groupings. For refine, we

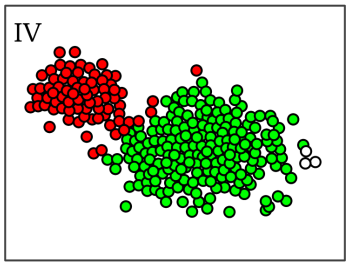


Figure 5.3: Result of the refine adjustment.

assume that f indicates the percentage of points that should be split off the cluster as noise. Basically, refine substitutes a frequent grouping with only one of its subsets to get a refined version of the original cluster. Again, the visualization must provide information regarding the estimated amount of noise in the cluster, in order to determine f. In our example, the current clustering contains two clusters represented by *fg71* and *fg65*. The user decides to refine *fg71* and remove around 1% of its members as noise by specifiying $f = 0.01$. The substitution algorithm starts and selects *fg69* as a replacement. Due to the local scope we desire on the primary control level, no further root nodes are processed and the algorithm starts to complete the clustering by establishing exact coverage. This is where our changes for refine take effect. After executing all necessary steps, the algorithm would normally select *fg5* as an additional cluster to cover the remaining members of the original *fg71*. But during a refine, only the direct substitute of the original cluster is kept, while all remaining subsets from its successor set that could be used for exact coverage are relocated to the noise cluster.

The result of the refine command is shown as a scatterplot in Figure 5.3. The two clusters are shown with colored circles, while the three members of *fg5* are left blank to mark them as noise. Please note that the amount of noise that can be removed with frequent groupings via refine, strongly depends on the underlying ensemble and the fg-graph created from it.

5.2 HYBRID VISUALIZATION CONCEPT

Besides interaction, communication of information about the clustering is a main task of our visual-interactive interface. Only if the user comprehends the structures that where identified in the data, result interpretation and evaluation are possible. This is also the basic requirement for the derivation of appropriate feedback and interaction with the clustering result via our proposed high-level feedback. Although there is a strong dependence between these two tasks, they are generally addressed as separate issues. In contrast, our hybrid visualization concept approaches them in an integrated way, which means that our employed form of communication is attuned to our feedback commands. In doing so, we are able to support result interpretation as well as the enactment of adjustments.

During our description of contemporary clustering practice in Chapter 2, we introduced visualizations and quality metrics as common approaches for result interpretation. Typically, visualizations for clustering are data-driven and create a visual element for each object of the dataset. The object's individual attributes are expressed via its position and/or shape on the screen, while its color represents cluster assignment. Using this approach, all raw information provided by the data and the clustering is communicated to the user as a whole. This offers an unparalleled richness of details and cluster structures that the user can utilize for the precise interpretation of the result. However, there are also inherent limitations that can lead to problems. First, data-driven visualizations maintain a $1:1$ mapping between data points and visual elements. For high volume datasets, this leads to an increase

of displayed information and can cause an 'overload' for the user. This effect is also known as visual clutter and puts strain on the user that can severely impair interpretation. If the human visual system encounters multiple objects, visual brain activity is actually reduced [17] as perception capacity is limited and objects compete for detailed examination. Although this problem is known and solution attempts were proposed for certain visualizations [4, 19], it remains a serious challenge. The second big problem of data-driven visualizations is dimensionality. Screens as the medium for display offer two dimensions, while human senses and thinking are designed for three-dimensional space. These limits strongly conflict with the large number of dimensions that datasets can feature. Most approaches try to scale down the information to fit the available means by reducing dimensions and/or employing complex visual metaphors for displaying the data. As a consequence, result interpretation becomes more challenging for the user.

The opposite of this situation is given with the result-driven quality metrics. Instead of depicting all elements and their assignment, these approaches use the available information as input for a specific function, whose calculated result represents a certain aspect of the clustering. By aggregating the cluster information into a single value, datasets of arbitrary volume and dimensionality can be compressed into a simple representation. However, due to this, large amounts of details are omitted. Furthermore, the usefulness of these measures is limited as there is no universal definition of clustering quality.

With our hybrid-visualization approach we want to establish a middle way between the two existing approaches to result interpretation. For this, we adopt the best of both worlds. On the one hand, we desire the compressed, aspect-oriented character of quality measures that ignores volume and dimensionality. On the other hand, we want a reasonable level of detail regarding the cluster structure, similar to what data-driven visualizations offer. To achieve this goal, we first have to decide on the granularity of visualization, i.e. the entities that should be displayed. Regarding this property, both existing approaches are located at the borders of the scale. While single data objects form the most fine-grained level, the whole clustering states the choice with maximum coarseness. Between these extremes, we decide to use the cluster level as our subject for visualization. Clusters are a reasonable compromise as they allow the communication of a respectable degree of detail without overstraining the user with too much information. Using clusters as visual elements is also a convenient way to prevent clutter. Typically, the number of identified clusters is orders of magnitude smaller than the number of objects in the dataset and can be easily managed by humans. With this, our visualization concept is not directly affected by the volume of the processed data.

Now that we have defined the subject of our visualization approach, we have to decide what kind of information we want to display. Due to the mentioned issues regarding dimensionality and volume, raw information per point is not an option. In order to obtain a clearly arranged and understandable visualization we use the aspect-orientation from quality measures as a starting point and expand this concept. Users must be enabled to interpret the result, identify clusters that need adjustment, and derive appropriate feedback options like split or merge. Traditional approaches do not support this complete spectrum of tasks as they focus purely on description. Typically, mathematical or statistical properties are measured, combined and finally presented to the user as an expression of quality. In general, the underlying notion/model of quality is not directly disclosed to the user as it is often very complex. For example, the Davis Bouldin Index [16] uses a function to measure scatter within a cluster and a distance metric to express cluster separation. Both aspects are measured for the whole clustering before their ratio is incorporated into the final result. In doing so, the user is left with an obscure score that expresses 'what' level of quality is present but does not offer a way to reason 'why' it is like this.

In order to support interpretation as well as adjustment, we need to identify understandable aspects that do not only describe structures but also can be used in a systematic way to decide, whether these structures are satisfying or not and why. To find such aspects, we regard the general definition of

clustering and derive two basic abstract characteristics. The first one is *composition*, which is used to describe the homogeneity and thus expresses how similar the members of each cluster are. The second aspect is *relations* which represents the separation of clusters, i.e. their similarity. Using both aspects in concert creates a small and easy-to-understand system for the evaluation of structure and the identification of appropriate feedback, e.g. if relations are satisfying but composition is not, the split or refine command could be used to improve intra-cluster homogeneity. In addition, these two aspects can be realized with a variety of measures. As long as it is clear what aspect the measure represents, the user will know how to incorporate it into his decision process.

With this setup, our hybrid visualization concept allows us to introduce more detail by unraveling the traditional single-display visualization into multiple views. Each of these views covers a certain facet of the clustering and is linked to the remaining views. A facet can illustrate composition and relations in an individual way by employing variable sets of measures, as long as these measures express our basic properties and are calculated on the granularity of clusters. This means that values captured for each point must be aggregated while values calculated for the whole clustering must be disaggregated.

The utilization of multiple views brings up the question of meaningful design and especially connections. To solve it, we use the visual information-seeking mantra: *"Overview first, zoom and filter, then details on demand."* proposed by Shneiderman [60] as a guideline. The mantra already indicates that views can express different aspects of a clustering on different scopes. By adhering to this approach, the character of our hybrid visualization changes from a traditional single-view that passively displays all information, to an interactive interface where interpretation is done by actively browsing a composite of multiple views. In summary, our concept provides not the actual visual-interactive interface, but a template for its implementation. In the following two sections we describe how this approach can be applied in a versatile fashion. We introduce two different realizations of our template, that are both able to work with the algorithm management but are aimed at different platforms and offer a different feature set.

5.3 VISUALIZATION I: LARGE SCREEN

Our first attempt at implementing the hybrid visualization concept is designed with desktop computers in mind. For the display of information, large stationary screens are used, while user interaction is realized with keyboard and mouse commands. This platform allows the utilization of multiple windows and provides enough resources to execute the algorithm management and the visual-interactive interface on one machine.

In order to visualize the composition and relations of clusters, we need sources that provide the necessary information. Our visual-interactive interface attaches directly to the integration part of the algorithm management, which makes the consensus clustering and the dataset natural candidates to fill this role. The values displayed by our visualization are derived by refining the raw data provided by these sources. Refinement starts with the calculation of centroids for the clusters of the consensus solution that are necessary to obtain inter-centroid distances and further values. Based on those centroids, the points of the dataset and their cluster assignment, we can subsequently calculate the soft cluster assignments. These assignments are used to express composition and relations of clusters. On their basis we calculate the reliability score, introduced with flexible clustering aggregation, that contributes to the description of cluster composition. The set of derived information is completed with histograms for each cluster and dimension of the dataset and with the minimal object-object distances between each pair of clusters.

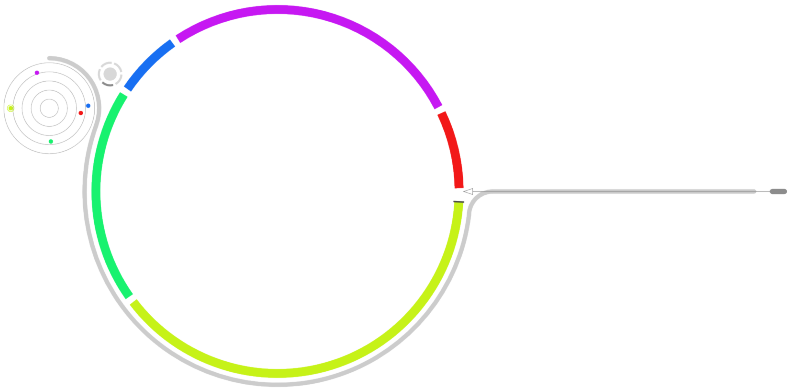


Figure 5.4: Overview showing clusters and inter-cluster distances.

Our visualization displays the derived information with simple visual components that allow users to easily interpret a clustering result and derive the appropriate feedback operations for iterative refinement. Our visualization consists of two windows: a primary window that displays the main visualization and a secondary one that keeps track of iterative refinement by listing the issued feedback operations and the resulting clusterings. In the following, we focus on the description of the primary window and its three views, as the secondary windows is only a visual listing.

5.3.1 Overview

The entry point of our visualization is defined by the overview. An exemplary depiction of it is shown in Figure 5.4. The subject of this presentation are the two most general characteristics of the consensus clustering: its composition, i.e. the partitioning of data into clusters, and the relations between these clusters. For the illustration of composition we use a circle, which forms the dominant element at the center of the view. It consist of differently colored segments that represent the clusters. The angle of each circle segment is proportionate to the size of the corresponding cluster. All in all, this resembles a pie-chart, which is a natural way of illustrating the partitioning of a whole and should be familiar to all users.

The relations between the clusters are expressed through the distances between them. As clusters are sets of objects, we must define the distance similar to the linkage constraint from hierarchical clustering. For the overview, distance between two clusters will be measured as the distance between their centroids. To depict these relations, we use a circular radar-like gauge located on the left of the main circle. It depicts each centroid as a dot whose color matches its respective circle segment. The illustration of distances is based on a distance graph, whose vertices are given by the centroids, and whose edges represent the Euclidean distance between centroids in the full-dimensional data space. The edges are not drawn in order to prevent visual cluttering. With this, the overview provides the user with a visual summary of the clustering result and allows a first evaluation of the partitioning and the relations between clusters. Since the overview works with clusters instead of points, it can handle high-volume, high-dimensional datasets without problems.

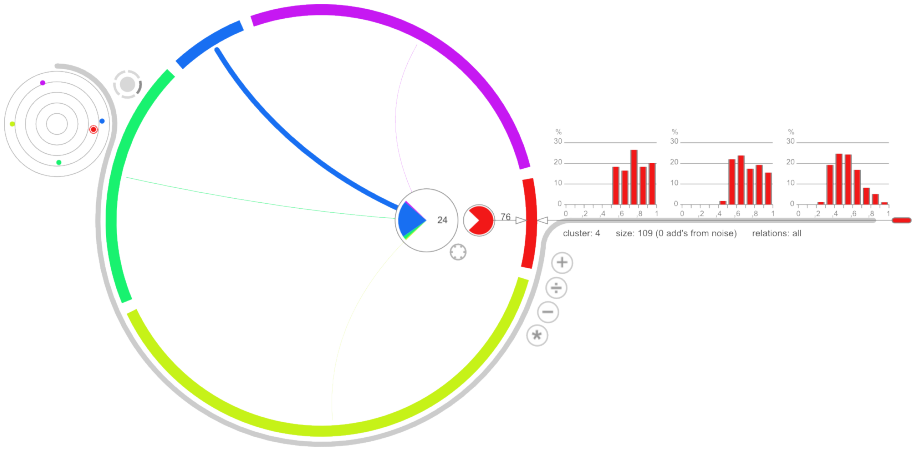


Figure 5.5: Cluster description view of the red cluster.

5.3.2 Cluster Description

More information on a specific cluster can be obtained by selecting it, which implements the '*zoom and filter*' step of the visualization mantra. Selection is done by using the mouse to rotate the main circle until the desired cluster segment intersects the small arrow on the horizontal axis in the right part of the view. As soon as a cluster is selected, the cluster description view is depicted and additional information regarding the cluster is displayed. An example of this view is depicted in Figure 5.5, where the red cluster from our running example is selected.

The selected cluster's composition is shown by the row of histograms on the right. All histograms feature the interval $[0.0 - 1.0]$ with ten bins of equal width. From the left to the right, they show the distribution of: (i) soft assignment values, (ii) reliability scores for all object-centroid pairs, and (iii) reliability scores for all object-object pairs of the selected cluster. The signatures of these histograms indicate certain cluster states. If a cluster is homogeneous and very compact, the soft assignments of its member to its centroid approach 1. This is also the case for both reliability scores. This manifests in histograms, where the number of objects per bin rises with increasing bin value, and the highest value bin on the right has the highest population. Let us regard the signature of the example depicted in Figure 5.5. The histograms show that many of the object-centroid and object-object pairs have only an average reliability. This can either indicate that the objects of the cluster are not very similar or that there are other clusters nearby that influence the selected cluster objects.

To complete such assumptions, the relations between clusters have to be examined. For this purpose, the two 'pie-chart' gauges and the arcs inside the main circle are used. The smaller gauge shows the degree of 'self-assignment' of the selected cluster, while the other one displays the degree of 'shared assignment' and its distribution among the remaining clusters. These degrees are calculated from the soft cluster assignments as follows: each soft assignment is a vector with a sum of 1.0, consisting of components ranged between 0.0 and 1.0 that indicate the relative degree of assignment to a certain cluster. As each vector-dimension corresponds to a cluster, the degree of self-assignment is calculated by summing up all components in the dimension corresponding to the selected cluster. This sum is then normalized to get the percentage of the total possible assignment that goes to the cluster itself. Accordingly, the shared assignment is generated in the same fashion for each remaining cluster/dimension. The target and strength of these outgoing relations between the selected cluster and others is described by the color and size of the shared-assignment slices. To better illustrate these pairwise relations, they are also displayed by arcs that connect the respective clusters and express the strength of the relation via their stroke width. If a cluster is well separated and not influenced by

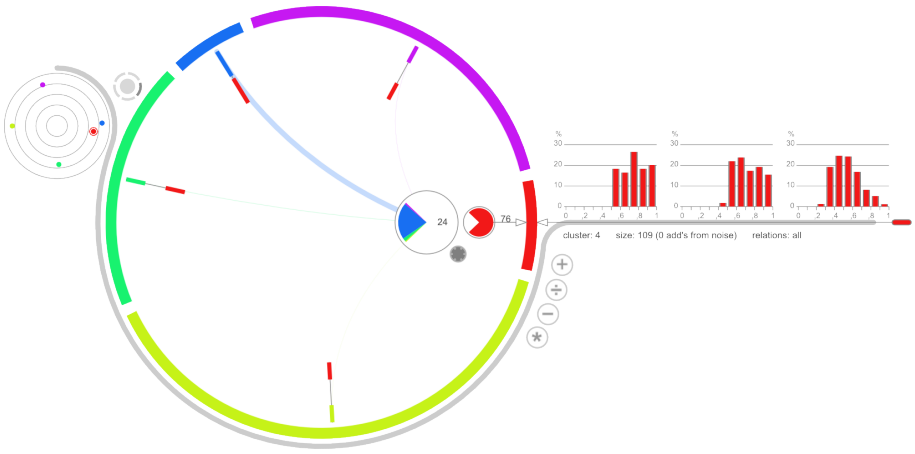


Figure 5.6: Cluster description with activated distance indicators.

others, it shows a very high degree of self-assignment with no outstanding relations to other clusters. In contrast, the example in Figure 5.5 shows that the selected cluster has a noticeable relation to the blue cluster. This supports the merge assumption and furthermore indicates the other cluster that should be part of a possible merge.

Further insight of relations can be obtained by examination of the inter-cluster distances. The radar-like gauge introduced in the overview only shows the distance between the centroids of clusters. Clusters can be pictured as a cloud of points in multi-dimensional space. Only considering the distance between centroids does not take their spacial extent into account. Although two centroids seem to be separated by a significant distance, their corresponding clusters can still be connected if they are stretched in space. To clear this issue, the user can overlay the cluster description view with additional distance indicators. These express the ratio of centroid-to-centroid distance and the minimum object-to-object distances between the selected and the remaining clusters. An example of this '*details-on-demand*' feature is shown in Figure 5.6. If the distance ratio approaches 1, the respective clusters are well separated and the colored bars of the indicator are distant. In our example, this is the case for all clusters except the blue one, where both bars nearly touch each other. This shows that the minimal object distance between the clusters is much smaller than the centroid distance which hints that both clusters are somehow connected. By utilizing the trinity of centroid distances, foreign assignments and distance indicators, the relations between clusters can be comprehensively analyzed. In our example, it is save to state, that the red and blue clusters could be combined. To double-check, the blue cluster is selected and shows similar relations to its red counterpart.

The cluster description view also allows to evaluate whether or not a cluster should be partitioned further. Such clusters can be identified on the basis of their histograms. If their most populated bin is located in one of the medium-significance bins, this indicates a strong heterogeneity in the cluster. This is emphasized by a reduced degree of self-assignment. Since these characteristics can also indicate a strong influence by nearby clusters, relations must be examined. If these show no prevalent relation to a specific cluster and indicate a clear separation in terms of inter-centroid distances and distance indicators, the cluster becomes a candidate for splitting. However, there is still a chance that these characteristics are caused by other reasons, e.g. non-spherical clusters. To gain more certainty, an examination of the cluster in the attribute view is advised.

Besides the communication of cluster characteristics, this view also provides the facilities for issuing feedback to the algorithm management. In Figures 5.5 and 5.6 an array of four buttons is placed to the right of the main circle. Each of these buttons triggers a feedback operation for the selected cluster. A merge is issued by pressing the $(+)$ button and selecting the second cluster for the merge.

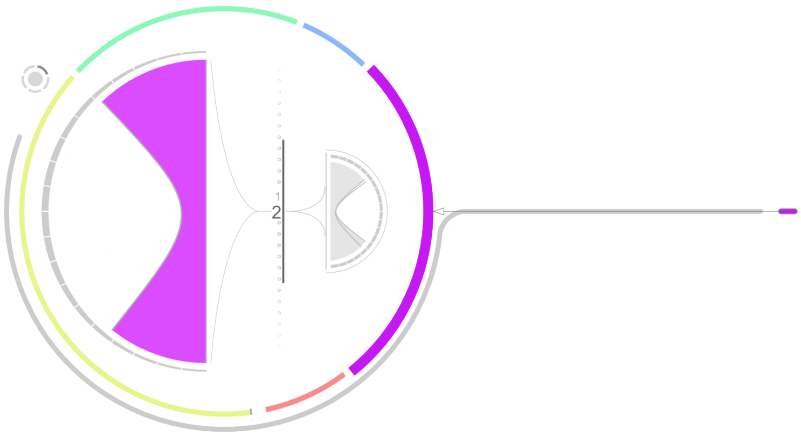


Figure 5.7: Attribute view indicating a split for the violet cluster

Splitting is done with the $(\div)$ button and refine via $(-)$. Both buttons require the specification of a threshold/force value between 0.0 and 1.0, which is realized with a dial displayed after pressing one of the buttons. The last feedback option restructure is issued for the selected cluster by pressing $(*)$.

5.3.3 Attribute View

The attribute view is a part of the 'details on demand' level and shows information on the distribution of cluster members in the different dimensions of the dataset. In contrast to the previous moves, the user must explicitly change to this view for a particular cluster. The attribute view occupies the center of the main circle and masks all other information like histograms or the different relation displays. Furthermore, it locks the selected cluster and must be left in order to select a different cluster.

In order to design this view, we looked at attributes in terms of clustering. We can state that an attribute with unimodal distribution—denoted as Φ—is not desired for clustering, because the objects of the dataset cannot be clearly separated in this dimension. In contrast, bi- or multi-modal distributions are interesting, since they can be used for the separation of objects. When we look at attributes on the cluster level, this is inverted. Regarding a cluster, it is desirable that all of its attributes have unimodal distributions, since this shows high intra-cluster homogeneity. A multimodal distribution of an attribute would imply that the cluster can be further separated in this dimension. Generally, we desire the following: On the dataset level, attributes should be dissimilar to Φ, while on the cluster level, they should resemble it as closely as possible. These are the basics for our attribute view.

To calculate the similarity to Φ, we use a straightforward approach. We generate histograms for each attribute on the dataset and cluster level. In each histogram, the bins that are local maxima are selected. Starting at these bins, we iterate over the neighboring bins to the left and right. Each neighboring bin that contains a smaller or equal number of objects than the current one is counted and becomes the reference bin for comparison with its next neighbor. The examination stops for each direction, if a neighboring bin has more elements than its predecessor. In doing so, we assume each local maximum as the mode of an unimodal distribution and collect all objects that belong to it by assuming a bell-shaped curve. From all unimodal distributions, we select the one that covers the maximum number of objects and display the points and bins *not* covered by this distribution in the attribute view.

Figure 5.7 depicts the attribute view for the violet cluster of our example. There are two hemispheres and a band of numbers between them. The band shows the attributes of the dataset, ordered by

our computed values, and is used to select an attribute for examination. The small hemisphere on the right shows the global behavior of attributes. Each curve represents an attribute, while for the selected attribute, the area under its curve is colored. The hemisphere itself consists of two 90-degree scales: the upper one displays the percentage of objects and the lower one the percentage of bins that can be fitted under Φ. The start and end point of each curve show the respective values for the attribute on these scales. If all objects and bins fit under Φ, a simple vertical line is drawn and no color is displayed in the hemisphere. The larger hemispherical display on the left works in the same fashion, but shows the attribute limited to the selected cluster.

In Figure 5.7 the second attribute, i.e. the y-axis of our dataset, is selected. We can see a large colored area, showing that roughly 50% of the objects and bins do not fit under Φ for this cluster. If the selected cluster also shows split characteristics in the cluster description view, the user may consider to split this cluster. The attribute view was designed with the goal of fast and easy interpretability of dimensional data. More color in the left hemisphere indicates a higher potential for further partitioning of the cluster in this attribute. In addition, the amount of color in the right hemisphere shows the usefulness of the selected attribute for clustering on the dataset level.

5.3.4 Summary

We introduced and described a first materialization of our hybrid visualization concept. In accordance with the visualization mantra, our approach features an overview, a 'zoom and filter' cluster description view, and 'details on demand' in the form of distance indicators and the attribute view. By focusing on the cluster-level, we can visualize clusterings regardless of volume and number of dimensions of the underlying dataset. To express the characteristics of composition and relation of clusters, we use different distances, soft assignments and the reliability scores from our flexible clustering aggregation. Our approach allows users to derive appropriate feedback operations for each cluster, while using understandable and familiar visual components like pie charts and histograms for the realization of all views.

5.4 VISUALIZATION II: SMALL SCREEN

After extensive testing of our desktop-based visualization, we came to several conclusions. First, we realized that the distribution of the time invested in the execution of clustering analysis is unbalanced. User interaction and the issuing of adjustments typically takes a short amount of time, while the computations necessary for their implementation usually take much longer. Depending on the size of the underlying dataset, the computational part can require hours to finish. During this time users are effectively condemned to idleness, which is why we reconsidered the used platform. While the size of desktop monitors is beneficial for visualizations, it also is a factor that makes work stationary. In order to improve accessibility and convenience of use, we selected smart devices as the platform for a mobile implementation of our visualization. This choice brings two major benefits: First, smart devices offer touch-screens and various sensors, that can be used to make interaction much more direct and intuitive. Second, smart devices are small and nearly omnipresent, which allows users to conveniently access the visual-interactive interface wherever and whenever they want. Unfortunately, these features come at the price of a greatly reduced display area. In order to exploit the benefits and adapt to the small screen of this new platform, we need to design a completely new visualization. The limited processing power of hand-held devices also requires the implementation of the main process components in a client-server architecture. While a high-performance server runs the algorithm management, the smart device only handles the visual-interactive interface.

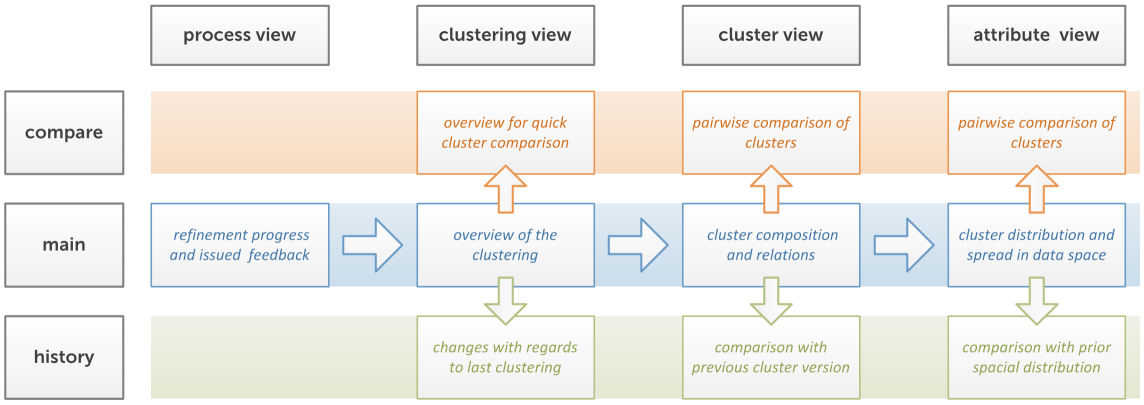


Figure 5.8: View architecture of the mobile visualization.

As smart devices offer only a small image area and are not intended to run multiple windows, we first have to re-think our view structure. In adherence to the Shneidermann mantra, we build a sequence of four views for: the process, the clustering, the cluster, and the attributes. With each view, the level of detail that is displayed gets finer. In contrast to our large screen visualization, these views are clearly separated in order to emphasize their different contexts. Furthermore, our first visualization attempt neglected the process character and only allowed the comparison of clusters by successive selection. To improve this, we introduce two tracks that run parallel to the main view sequence. The compare track allows comparisons on each level of detail, while the history track provides the user with information regarding the changes in comparison with the previous clustering result. Our view structure and the possible transitions between views are shown in Figure 5.8.

Subsequently we will describe each view and the information it displays in detail. We basically keep the information sources that were already used for our large screen visualization. At certain points we will complement them by introducing additional derived values.

5.4.1 Process View

This view acts as the entry view of the visualization and depicts how the iterative refinement progresses by showing issued feedback and the adjusted results. An example of this view is shown in Figure 5.9 and shows a sequence of three clustering results. Each clustering is represented as a small segmented circle that represents the number and relative size of all identified clusters. In addition, information regarding the executed adjustments is displayed. The leftmost clustering represents the initial consensus clustering. The two colored lines and the "+" symbol next to it show that the user decided to merge the blue and purple clusters. The next clustering of the sequence illustrates the result of this adjustment and has a reduced number of clusters. An arrow in its center indicates the new cluster created by the merge. At this point of the iterative refinement, the user decides to divide the green cluster and issues a split. Its result forms the last clustering of the depicted example sequence. Again, arrows indicate the new clusters that result from the split. With each issued feedback operation, the sequence is expanded and thus the whole refinement process is recorded. Users can trace back their adjustments and review the information for each step. Furthermore, this view allows to undo a particular chain of adjustments. Assuming a sequence of n steps, the user can enter an arbitrary step k with $0 \leq k \leq n$, and issue a different adjustment. In doing so, all steps that follow k are undone and replaced with the newly created $k + 1$. The process view is the only view that does not use our two additional tracks, as we have not considered comparison and a change-history on the process level in our implementation. Although, these could be used to support collaborative analysis in future versions.

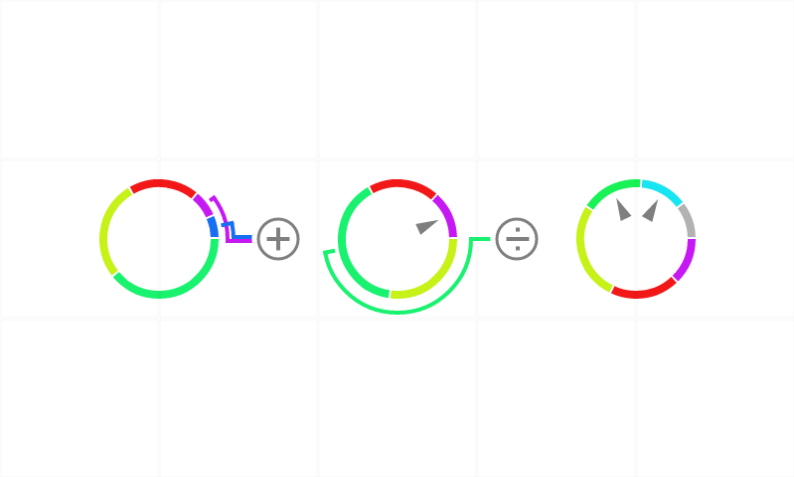
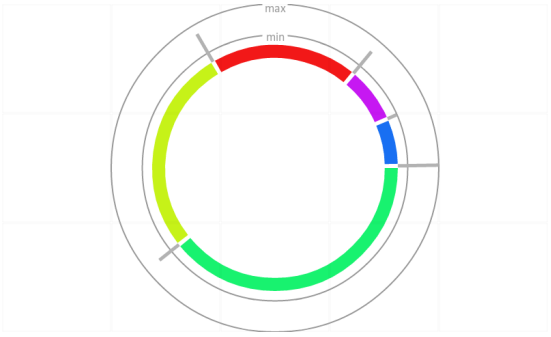


Figure 5.9: Process view showing a sequence of three steps.

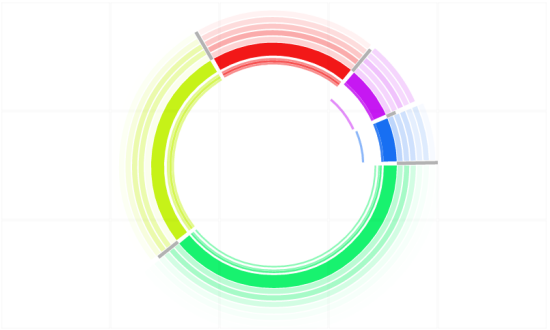
5.4.2 Clustering View

The clustering view forms the next level of detail after the process view. It is used as an entry point for accessing individual clusterings created during the process of iterative refinement. Examples for this view and its versions on the compare and history track are shown in Figure 5.10. We start with the description of the main clustering view depicted in Figure 5.10(a). Like the overview of our desktop-centric visualization, all clusters and their sizes are represented as circle segments with different sizes and colors. For the display of the relations between clusters, we use a modified approach. In our previous visualization, we used a distance graph to show the distances between centroids. Unfortunately, this approach becomes inaccurate if the number of dimensions of the underlying dataset increases. This is due to the fact that the graph is projected into two-dimensional space for display. To exemplify this problem, assume a dataset with 5 dimensions and 4 clusters, all having the same distance to each other. For correct representation of this setting on a plane, it would be necessary to draw 4 points and connect each of them with all other points, using only straight lines of the same length, which is not possible. In order to solve this issue, we compute all distances between the cluster centroids. We then calculate a minimum spanning tree from the resulting distance graph and use it to order the segments of the circle. With this, adjacent segments express that the distance to the corresponding clusters is relatively small. In addition to this ordering, the actual distances between adjacent clusters are displayed with lines of different length that emanate from the main circle. Two thin additional rings are added to represent the minimum/maximum distance between centroids and act as reference for the evaluation of relations between the clusters.

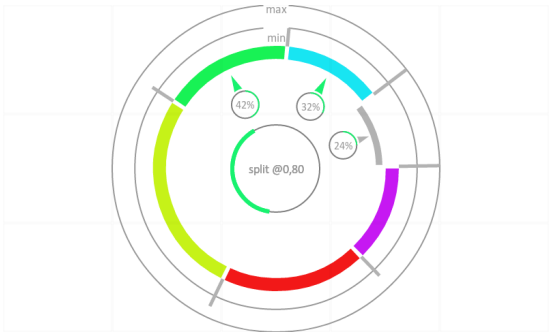
From the main view, the user can switch to the compare track by performing a downward slide gesture. This brings up the quick compare view depicted in Figure 5.10(b). We can see that an outer and inner halo is added to the main circle. For each segment, the outer halo summarizes cluster composition, while the inner halo shows a glimpse of the relations towards other clusters. The measures necessary for this representations are part of the cluster view and will be described in detail in the following section. These halos allow a quick assessment of composition and relations for all clusters. This is done by evaluating the extent of the halo of each segment. Large halos indicate that the respective characteristic is unsatisfactory and adjustments should be applied to the corresponding cluster. In addition to providing an overview on cluster character and providing hints regarding feedback application, this view allows the easy visual communication of refinement progress. A clustering that needs heavy adjustment is represented with extensive halos and gets a wide and blurred silhouette. If



(a) main view



(b) quick compare



(c) structural changes

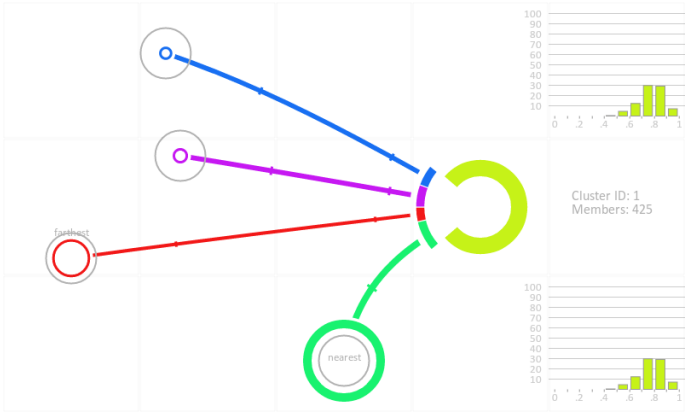
Figure 5.10: Examples for the clustering view and its track-variants.

appropriate adjustments are applied, the halos are reduced and the circle silhouette becomes slim and sharp.

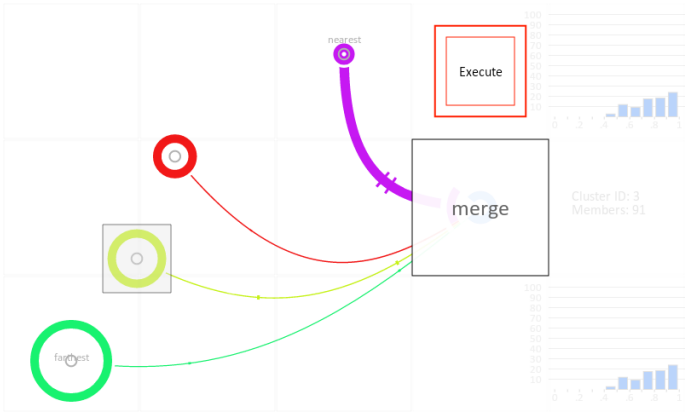
By performing an upward slide gesture, the user leaves the compare track and gets back to the main view, from where the history track can be accessed with an additional upward slide gesture. An example for this view is depicted in Figure 5.10(c) and shows the changes between the current and the previous version of the clustering that where induced by the applied adjustment. In our example, a split was applied with a force of 0.8 to increase the number of clusters in the current clustering. The issued feedback operation is displayed at the center of the main circle. A smaller inner circle depicts the size and position of the originating cluster in the former clustering. Arrow indicators show that 42% and 32% of the former green cluster's members were relocated to two new clusters, while the remaining objects were assigned to the noise cluster. With this easy-to-understand overview, users can track the structural changes caused by each adjustment.

5.4.3 Cluster View

After getting a general overview of the clustering, users can select an individual cluster to get more detailed information on it. Touching a circle segment in the clustering view triggers an animation that brings up the corresponding cluster view. This view realizes the "zoom and filter" level of Shneidermanns mantra and is illustrated in Figure 5.11(a). In our example, the yellow-green cluster is selected and displayed as a circle on the right. The right side of the view provides the cluster id, the number of cluster members, and information on cluster composition in form of two histograms. In the main cluster view, both histograms are identical and express the self-assignment of the selected cluster by showing the distribution of its respective soft assignments. Relations are shown in the left part of the



(a) main view



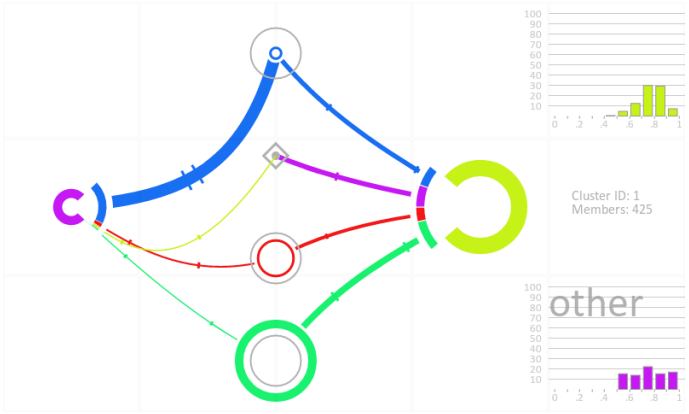
(b) feedback menu

Figure 5.11: Example cluster view and adjustment application.

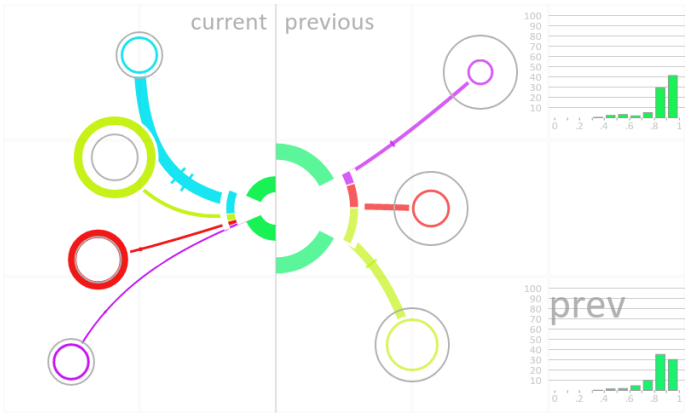
view, where each remaining cluster is represented by a circle. Similar to our large screen visualization we use soft assignments and distances to express them. Again, soft assignment values towards each cluster are aggregated and visualized with circle segments of different size and connecting lines with different stroke weight. Centroid-to-centroid distances are reflected in the placement of circles either near or far from the selected cluster's circle. The distance indicators we introduced in our former implementation were integrated in the connecting lines and are displayed as orthogonal lines that become more prominent as clusters become closer. In addition, the circle diameter is used to express the size of each cluster in comparison with the selected cluster, whose size is added as a thin gray reference circle. Clusters that are bigger than this reference are also emphasized with an increased stroke weight. All in all this enables users to easily appraise cluster characteristics and derive the appropriate feedback/adjustment.

The cluster view also acts as a point of interaction for the application of feedback operations. When a user decides to adjust the selected cluster, a long tap is used to bring up a menu with our four high-level feedback commands. The user picks the command of his choice, provides additional parameters, like force, and finally executes the specified adjustment. Figure 5.11(b) illustrates an example for the issuing of a merge. The respective command is shown as a square button overlaying the selected cluster, the merge partner was already selected and is also marked with a square overlay. All necessary selections are made and the execute button that triggers the adjustment is unlocked.

In addition to the main view, the compare and history track views exist and can be accessed with the same slide gestures used in the clustering view. An example for the compare track of the cluster view



(a) pairwise comparison



(b) cluster changelog

Figure 5.12: Examples for the cluster view and its track variants.

is shown in Figure 5.12(a). This view is used to make pairwise comparisons between clusters. For this, we basically mirror the main view and place a second cluster representation face-to-face with the selected cluster. To allow comparison of both cluster compositions, one of the histograms is switched to show the self-assignment of the other cluster. Relations between both clusters and all remaining clusters are visualized like in the main view, with the exception of centroid-to-centroid distances. As all clusters that are not part of the chosen pair must be aligned between both selected clusters, variable distances cannot be used.

The history track of the cluster view is depicted in Figure 5.12(b) and shows the transformation of the selected cluster. For this view, the presentations of the current and previous cluster versions are joined at their corresponding circles and create a janus-faced display. A vertical line separates a major part of the display area and marks the corresponding cluster versions. Changes in cluster size can be read from the difference in size between the two semicircle components at the center. The composition of both cluster versions is again shown with the two histograms on the right. The presentation of relations to other clusters does not change. In our example we can see the aftermath of a split operation. The former large green cluster was split into two new clusters. The current green cluster is roughly half the size of its former self, because a large portion of its members was reassigned to its new light blue sibling. Changes in relations occur mostly due to the introduction of an additional cluster. We can also see that the newly created siblings are very close to each other which indicates that further adjustment might be necessary.

In the former section, we introduced the compare track for the clustering view and its halos. These halos are created from parts of the information represented in the cluster view. The outer halo that

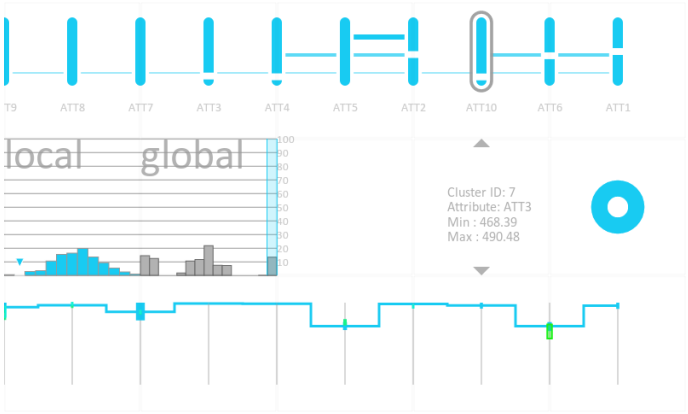
represents composition is made up of concentric circle segments that correspond to the bins of the soft assignment histogram. The segment for the highest value bin is closest to the main circle segment, while the radii of the remaining segments increase as the bin value decreases. The population of each bin determines the transparency of the corresponding segment, i.e. an empty bin results in an invisible circle segment. All this leads to the described character of the outer halo, where populated low value bins lead to a wide and blurry silhouette. The inner halo is created using a similar approach based on the shared assignment of each cluster. The degree of assignment to each other cluster is again visualized as a concentric circle segment. This time, the value of a relation is used to determine the distance to the main circle, i.e. strong relations become distant segments that widen the silhouette and thus indicate a need for adjustment.

5.4.4 Attribute View

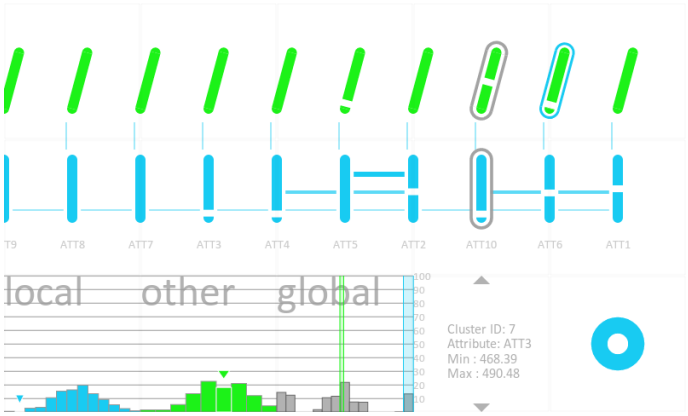
The attribute level constitutes the last station of our view sequence and offers the most fine-grained information in our visualization. It is accessed by dragging the selected cluster's circle to the right edge of the cluster view. In contrast to our previous views the attribute view itself is more complex as information for every dimension must be displayed for the cluster. For this reason, the attribute view contains several facets that act as overview, zoom-and-filter, and details-on-demand levels for its own scope. However, the attribute view as a whole takes the details-on-demand position in the context of our visualization. An example of this view is show in Figure 5.13(a).

In general, the view can be divided into three horizontal ribbons. The top ribbon acts as overview for all dimensions of the selected cluster, and shows an array of vertical bars that are connected with horizontal lines. Each vertical bar represents a dimension and summarizes its distribution by adding gaps to the bar. For this we use an approach similar to the attribute view of our large screen visualization. Again, we identify local maxima in the histograms of each dimension and look for the largest unimodal distribution. If all members of a cluster fit under a single distribution, the corresponding bar is complete. If the distribution is bimodal, a gap is created to split the bar according to the ratio of objects covered by the two unimodal distributions. Distributions, where more than two unimodal distributions can be found, result in only one additional gap. With this, the bar is divided according to the ratio of objects covered by the largest, second largest, and remaining unimodal distributions. This allows to quickly evaluate if the cluster can be split in certain dimensions.

The horizontal lines between the bars visualize the relations between dimensions. As a measure for relations, we use the statistical independence, which we measure between all pairs of dimensions. The calculated values are normalized to a range from 0 to 1, which is divided into three equal parts that are each represented with one horizontal line. The number of lines between two bars expresses the degree of independence between both dimensions. One line corresponds to a small value v with $0 \leq v \leq 0.33$, while three lines show strong relations with $0.66 < v \leq 1$. These values are also used to sort the bars in a way that places dimensions with strong independence next to each other. Groups of statistically independent dimensions are of interest as they typically present promissing subspaces for clustering [54]. In order to get more information on one of the dimensions from the top ribbon, users can scroll the array of bars using a slide gesture and select a single dimension by placing it under the gray selection frame. Details on the selected dimension are shown on the middle bar and contain dimension name, minimum and maximum values as well as two histograms. The first one shows the distribution of cluster members in the selected dimension, while the second one shows the global distribution and overlays the range occupied by the selected cluster. In our example we can see that the cluster has nearly an unimodal distribution in the selected dimension and is located at the right edge of the global dimension's range. The bottom ribbon, again, provides an overview regarding the cluster's location in multidimensional space. We use a presentation similar to parallel coordinates



(a) Main view.

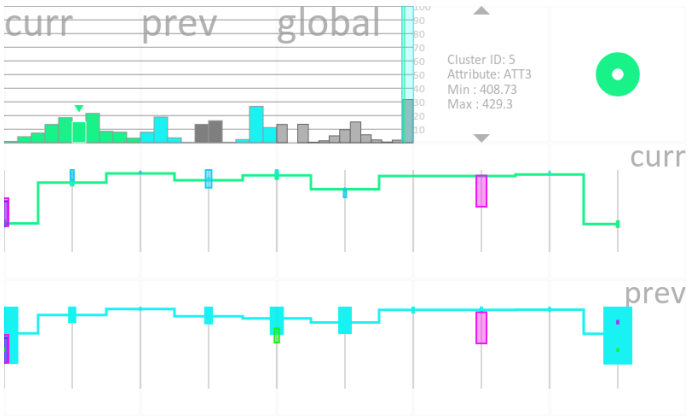


(b) Compare-track

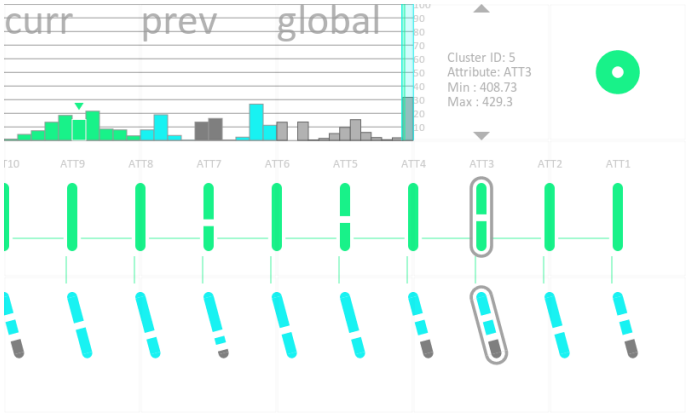
Figure 5.13: Examples for the attribute view.

and depict the cluster as a horizontal line that cuts the vertical lines, i.e. dimensions, at a point that corresponds to the cluster centroid's location in the range of the particular dimension. In addition, we use colored boxes on each vertical line to show the range a cluster occupies. The presence of multiple colored boxes indicates that other clusters intersect with the selected one in this dimension, which can be interpreted as a hint towards further adjustment.

Like in all our views, the compare track is accessed with a downward slide and is shown in Figure 5.13(b). We can see that the ribbon configuration has changed. An additional bar ribbon has been added at the top and displays the dimensions of another cluster for comparison. Interpretation is similar to the main view, although some components were added. The dimensions of both clusters are connected with vertical lines that express the overlap of strongly connected groups of dimensions between clusters. Assume a set of dimensions $[D6, D8, D10]$ that is connected with three lines in the selected cluster and a set $[D4, D5, D6, D8, D9]$ with equally strong relations in the other cluster. Both sets intersect in $[D6, D8]$ which would be visualized with three vertical lines on top of the bars for $D6$ and $D8$, illustrating that this subspace is shared by and has equally strong relations in both clusters. Like in the main view, a single dimension can be selected by scrolling. Details for this dimension are shown at the bottom ribbon with three histograms, of which the first two depict the local distribution for both clusters, while the third shows the location of the cluster pair on a global scale. This information can be used to find overlaps between clusters and derive appropriate actions. Cluster overlap is also shown in the bar arrays. Each dimension of the other cluster which overlaps with the selected cluster, is framed in the selected cluster's color.



(a) changes in spacial extent



(b) changes in local distribution

Figure 5.14: History track of the attribute view.

We complete our description of the attribute view by regarding its history track that is depicted in Figure 5.14. In contrast to the views introduced so far, this history track has two display modes. This is necessary because adjustments cause changes in a clusters position and extent in global dimensional space, as well as changes in the local composition and relations of dimensions. Figure 5.14(a) shows the first global mode. Like on the compare track, the ribbon configuration has changed. The two ribbons at the bottom use our parallel coordinates display to illustrate the location of the current cluster and its previous version in the dimensional space of the whole dataset. In our example we can see that the current cluster is created by splitting up the former light blue cluster, which of course leads to a decrease in the range size occupied in each dimension. Furthermore this resolved some overlaps with other clusters and, thus, made a contribution to separation. Similar to the compare track, we use the top ribbon to display histograms for the currently selected dimension. Our example shows that the applied adjustment sharpened the cluster profile. While data distribution in the previous version was trimodal with distinguished peaks, the current distribution only has two very similar peaks. Thus, the change also improved the overall cluster composition. Using a slide gesture, the user can switch to the local mode of the view, shown in Figure 5.14(b). There, we replaced the two bottom ribbons with our familiar bar arrays that give an overview of the composition and relations of all dimensions. We can easily see that the split operation results in a general improvement of the composition. Local distributions have become unimodal or have reduced their number of peaks, which indicates an increase in intra-cluster similarity. No changes in relations can be observed, which means no new interesting subspaces have emerged.

5.5 SUMMARY

The visual-interactive interface is the vital component of our clustering process, as it constitutes the main point of interaction for the user. Its control and communication facilities are always in plain sight, while the workings of the algorithm management are mostly hidden. A clustering analysis with our process can only be successful, if users can easily interpret and adjust the provided clustering results. For this purpose, we introduced a compact and universally applicable set of four high-level feedback options. Despite their small number, these options provide users with a mighty tool for cluster refinement. Even if the underlying algorithms change, feedback stays fixed from the user's point of view, as all necessary mappings to the algorithm management are made internally.

In order to communicate the result itself and enable users to interpret it, we proposed a novel hybrid visualization concept. Designed as a middle way between data-driven visualizations and result-driven quality measures, it combines the best traits of both existing approaches. The core of our concept is the focus on the cluster level as the subject of visualization and the communication of the two abstract aspects of composition and relation. This aspect orientation and the focus on clusters makes our approach immune to high dimensionality and clutter induced by high-volume data. To ensure a reasonable level of detail for the communicated information, our concept unravels traditional single-display visualization into a multi-view structure. Each view uses several measures to cover certain aspects of the clustering result and users browse these views to interpret it.

Per definition, our concept does not specify a visualization technique but provides a template for its construction. We described two implementations of this template and illustrated the adaptability of our concept. Our first visualization is aimed at stationary desktop platforms that offer large screens and conventional input. It offers three simple and clear views that allow result interpretation and fully support all of our feedback commands. For our second visualization, we choose smart devices as a platform. Besides convenient access at any time, these offer touchscreens as a very intuitive tool for interaction. To adapt to this platform, we redesigned our view structure and added features for cluster comparison and tracking of changes, which better accommodate the process character of our iterative refinement. Furthermore, the application of feedback was directly integrated into the visualization. With all this, we provide users with a powerful tool that is still convenient to use and easy to understand.

PROCESS APPLICATION

6.1 A Guided Tour to Iterative Refinement

6.2 Learning by Doing - Process Adaptation

6.3 Summary

At this point, we posses all the parts that are necessary to realize the template of our versatile clustering process, introduced in Chapter 3, and put it to use. For this, we start by assembling an implementation of our process concept. Because of its modular character and the fact that we described different options that can fulfill the necessary tasks, we can already create different process variants. For this chapter, we choose a process configuration aimed at inexperienced users.

Starting with the algorithm management, we select the flexible clustering aggregation to perform integration, as it is a little more approachable than frequent groupings. As we address inexperienced users, we also limit control options to the primary level. This means, users have no access to ensemble configuration or the creation/adaptation of clustering algorithms, as these actions require a higher level of experience. All high-level feedback commands are mapped in accordance to these regulations. However, we must provide the restructure command that requires access to the secondary level of control. This is done by fixing the ensemble configuration, i.e. if a user applies the restructure command on a certain cluster, the initial ensemble configuration is applied to the respective subset of the data. In doing so, the full set of feedback options is provided without user access to the secondary level. For the realization of the visual-interactive interface, we use our small screen visualization for smart devices.

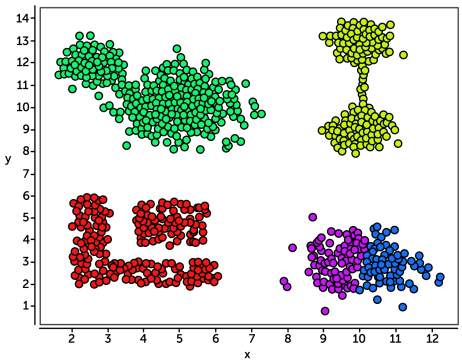
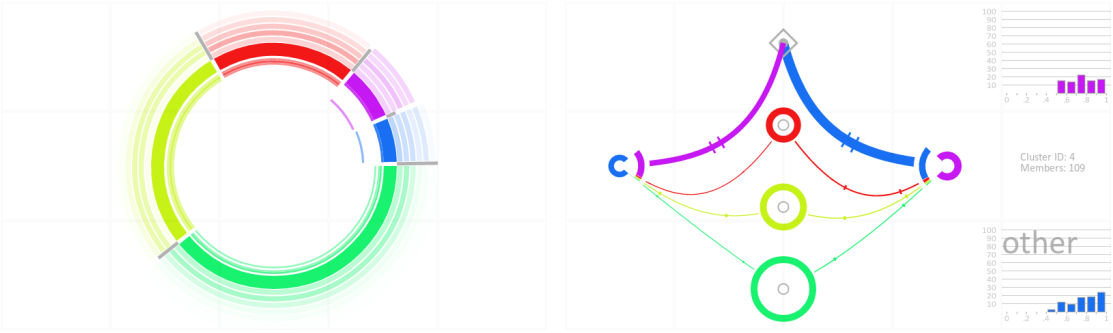


Figure 6.1: Initial clustering for the example dataset.

Our actual application setting uses a synthetic dataset comprising 1500 points, distributed in 2 dimensions. Such a small setting is ideal for explanatory purposes, as we can still use scatterplots to display the clustering results for reference. The employed ensemble contains 10 clusterings that were created with k-means. Values for k range from 2 to 15 and each clustering features a different seed for centroid initialization. A scatterplot of the resulting initial clustering is shown in Figure 6.1. In the following sections, we give an exemplary description of our process's execution and illustrate how it can be easily adapted to cope with a variety of scenarios.

6.1 A GUIDED TOUR TO ITERATIVE REFINEMENT

We illustrate the execution of our process in form of a tour, by describing the individual actions a user takes during the iterative refinement of a clustering. This illustration is arranged into a sequence of steps and focuses on images that show what the user sees during the course of the process. Each step features figures of the views that are crucial for interpretation and derivation of appropriate feedback, as well as complementary remarks.

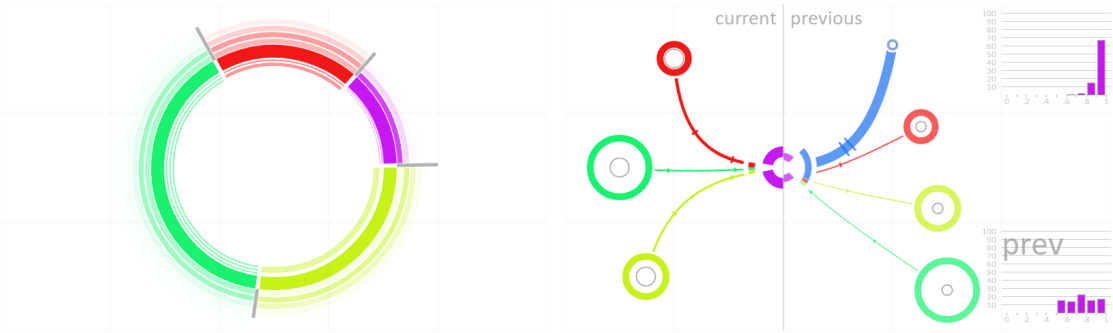


(a) Compare track of the clustering view. (b) Comparison of merge candidates.

Figure 6.2: Starting point of the clustering process.

Step 1: This is the initial result provided by algorithm management and presented to the user at the start of the clustering process. To get a quick overview and appraise the situation, the compare track of the clustering view shown in Figure 6.2(a) is used. There, our attention is drawn towards the purple and blue clusters, where the inner and outer halos have their peak extension. Both clusters are relatively small and have a low centroid-to-centroid distance. This is also depicted in the lower right quadrant of Figure 6.1. We select the purple cluster and bring up its cluster view that shows a dominant relation to the blue cluster, as well as an unsatisfactory composition. All of this strongly indicates a merge adjustment, but before we issue it, we bring up the compare track shown in Figure 6.2(b) to double-check with the potential blue merge partner. The view confirms our assumption. Both clusters share a dominant relation with each other and express close proximity via the distance indicators. Simultaneously, they are clearly separated from the remaining clusters. In addition, their composition histograms indicate that their self-assignment is not very strong. Thus, we issue the merge command that leads us to the next step.

Step 2: As can be seen in Figure 6.3(a), the merge operation fused both small original clusters into a new purple one. The slimmed down halos already show the success of the adjustment. To get more details on the change, we check the history track of the corresponding cluster view shown in Figure 6.3(b). The new cluster has doubled in size due to the assimilation of all former blue members. Histograms illustrate that composition has improved greatly, while relations are now balanced and show a clear separation. Now, we move on to adjust the yellow-green cluster, which is located in the upper right of Figure 6.1. Its cluster view depicted in Figure 6.4(a) shows good separation but only mediocre composition, which makes this cluster a typical split candidate. To confirm this, we switch



(a) Compare-track of the clustering-view. (b) Changes induced by the merge.

Figure 6.3: Results of the merge adjustment.

(a) Cluster-view.

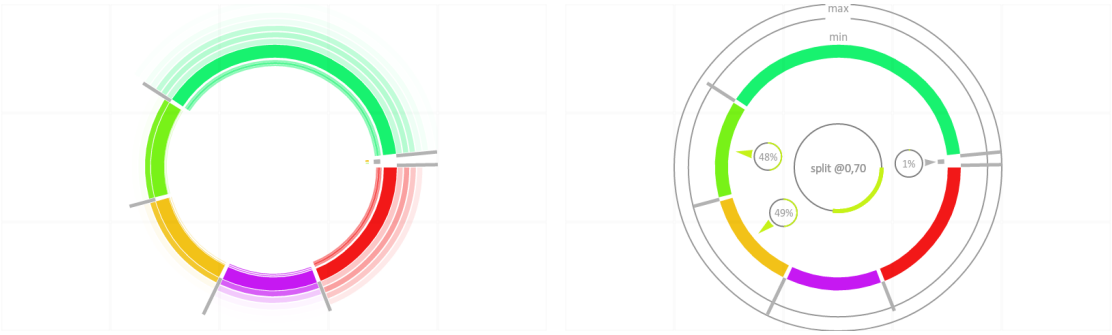
(b) Attribute-view showing the distribution along dimension y.

Figure 6.4: Views indicating a split adjustment.

to its attribute view shown in Figure 6.4(b). There we find an unimodal distribution in dimension x, but a clearly bimodal distribution in dimension y, which indicates that the cluster can be divided further along this dimension. We issue the split and derive a value for its force from the composition histogram, using a rule of thumb. During our work on the flexible clustering aggregation, we found that a good force value can often be obtained by searching for a point of steep rise in bin size. In our example this point is situated between the 7th and 8th bin, so we set force to 0.7 and execute the split.

Step 3: The views shown in Figure 6.5, depict two new clusters. On the compare track of the clustering view, slim halos illustrate improvements to composition and relations. The history track tells us, that the former cluster was nearly divided in half and that a small amount of noise was created during the process. Satisfied by this result, we examine the green cluster next. In the scatterplot of Figure 6.1, this cluster occupies the upper left quadrant. Judging by the views in Figure 6.6, it has a character similar to the yellow-green cluster we just split. Relations to other clusters are weak, which indicates clear separation. Composition is only mediocre and a detailed check of the attribute view reveals bimodal tendencies in both dimensions. Although, these are less pronounced in comparison to the previous split candidate, we decide to issue a split. Following our rule of thumb, we again choose a force of 0.7.

Step 4: This time, our adjustment did not have the desired effect. Figure 6.7(a) shows, that the former cluster remains intact, with roughly a tenth of its points being relocated to the noise cluster.



(a) Compare track of the clustering view.

(b) Changes induced by the split.

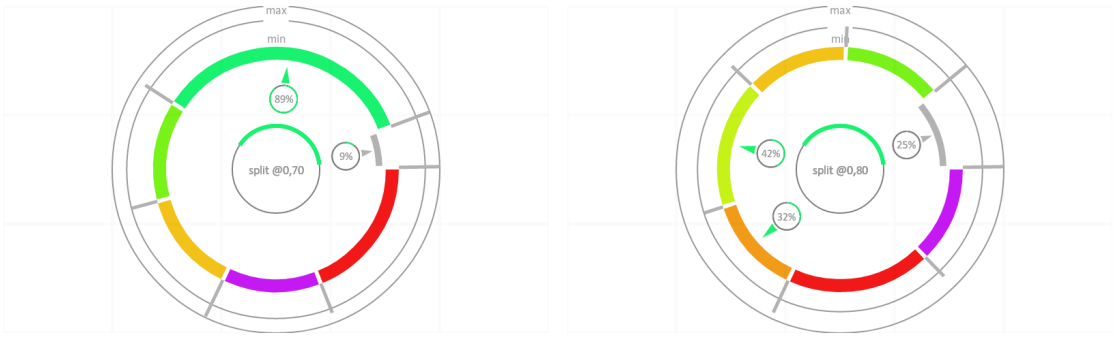
Figure 6.5: Results of the split adjustment.

(a) Cluster view. (b) Attribute view.

Figure 6.6: Views indicating a split adjustment.

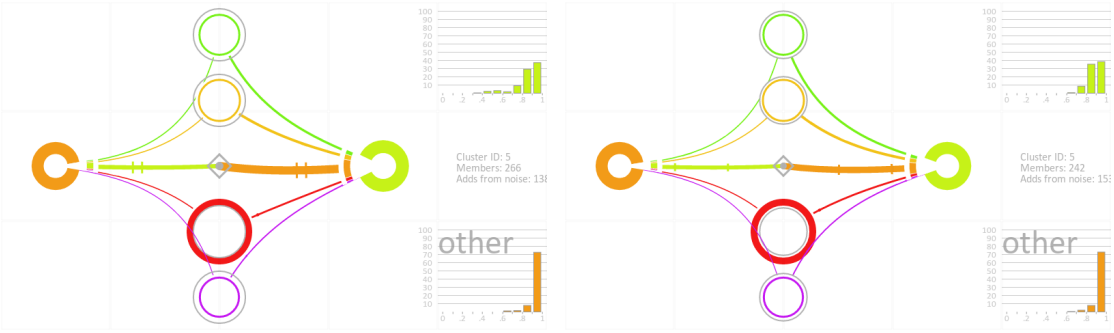
Halos on the compare track still indicate the original unfavorable characteristics, so we decide to redo this step. We switch back to the process view, select the previous clustering and head to the cluster view of the original green cluster, where we issue the split again, but this time use an increased force of 0.8. With this, the existing step 4 is removed and replaced with the newly calculated step 5.

Step 5: The repetition of the split adjustment works as desired and two new clusters are created. The proportional distribution of original cluster members is depicted in Figure 6.7(b). By looking at the halos on the compare track, we find that both new clusters have improved in comparison with the original green cluster. However, the still considerably wide inner halo of the new yellow-green cluster indicates a need for further adjustment. The main cluster view reveals a pretty strong relation to its sibling, which is why we compare both clusters as depicted in Figure 6.8(a). A certain proximity between clusters that result from a split is normal, since they descend from the same parent. However, in our example the relation is noticeable and furthermore accompanied by distance indicators showing very little distance between the members of both clusters. The reason for this can be found by closer examination of the composition histograms. While the orange one is nearly ideal, the yellow-green one shows a group of 4 scarcely populated bins below the assignment value of 0.7. These objects with weak affiliation to the cluster are most likely small noise structures, e.g. thin object chains that disturb the expression of relations. To solve this issue, we apply the refine command with a force of 0.7 to remove said structures, i.e. all objects located below the 8th bin.



(a) Compare track showing a failed split. (b) Successive split with increased force.

Figure 6.7: Results of a repeated split adjustment.



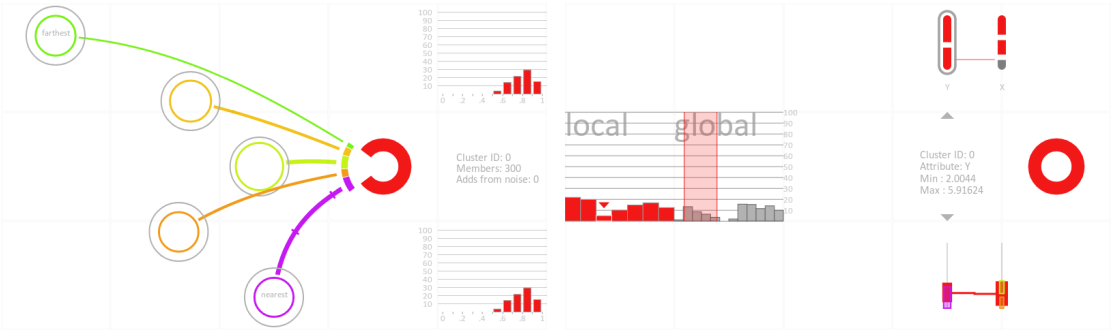
(a) Comparison of the two newly created clusters. (b) Comparison after refine adjustment.

Figure 6.8: Improvement of cluster relation with the refine command.

Step 6: After the completion of the refine adjustment, the history track of the clustering view shows that about 10 percent of the original members where split off the cluster. Regarding our updated compare track shown in Figure 6.8(b), we see that the tail of bins was removed as desired. The strength of the relation between the sibling clusters decreased and the distance indicators show a better separation. Satisfied with our results, we move on to the remaining red cluster shown in the bottom right of Figure 6.1 that is still in need of adjustment. Its cluster and attribute view, depicted in Figure 6.9, show the familiar characteristics of a split candidate. Similar to the green cluster, the distribution of members in each dimension is bimodal but not very pronounced. Taking our previous experiences into account, we issue a split with an increased force of 0.7, although our rule of thumb would recommend only 0.6.

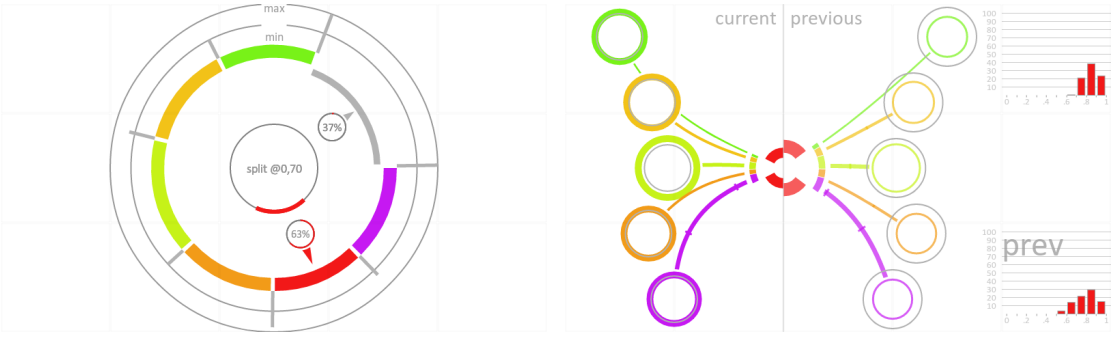
Step 7: Despite the increased force, no new clusters were created. Looking at Figure 6.10, we can see that more than a third of the cluster members are relocated to the noise cluster, but no meaningful partitioning is established. Checking the history track of the cluster view reveals only minimal changes in composition and relations but no real improvements. Like before, we redo this step and increase force to 0.8.

Step 8: Again the cluster is not split up. This time, nearly 80% of the cluster cannot be assigned to any cluster and are relocated to noise. The inability to establish a propoer split originates from the ensemble. In this case, the majority of clusterings identifies only one cluster in this particular area of the dataset. Thus, no additional structural information that could be accessed through integration



(a) Cluster view. (b) Attribute view.

Figure 6.9: Views expressing the characteristics of a typical split candidate.



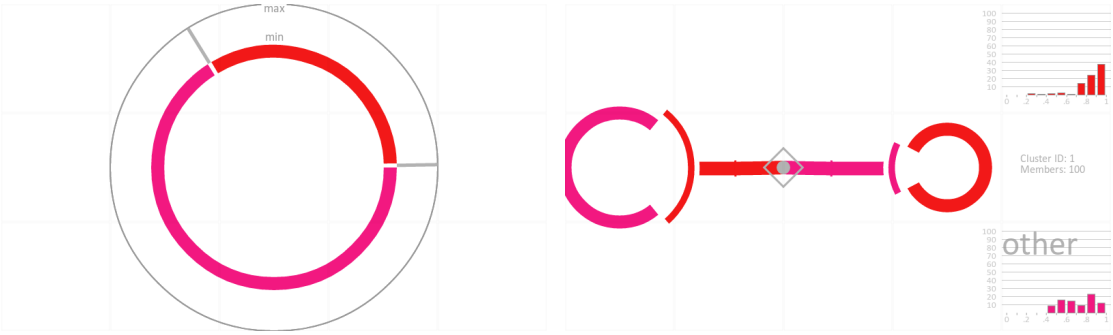
(a) Clustering after the split. (b) Cluster changes after the split.

Figure 6.10: Failed split attempt.

is provided. To get out of this dead end, we have to use the restructure commmand on the cluster. With this, a new set of clusterings is created for the corresponding subset of the data. As stated in the beginning, the same ensemble configuration that was already used for the initial solution is applied again. This frees the user from working on the secondary level of control and allows the adjustment to be made by simply pressing a button. Due to the range of k and the reduced data range, we can expect more structural detail from the new ensemble.

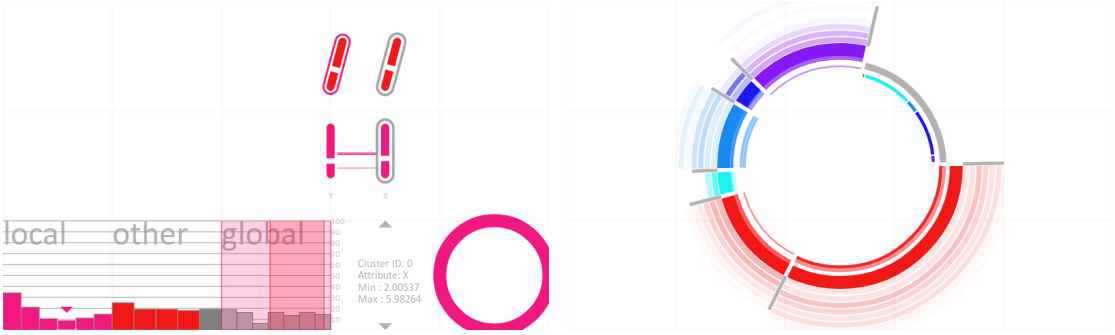
Step 9: The operation results in a new clustering with two clusters, which is depicted in Figure 6.11(a). Using the compare track of the cluster view shown in Figure 6.11(b), we can see that one has a pretty good composition, while the other shows an unsatisfying histogram. Unfortunately, relations do not contribute much to the interpretation in this case, because there are only two clusters. Thus, there is only one bilateral relation, that we cannot evaluate as no further relations exist for reference. We are now faced with the question of how to proceed. Since we only have four feedback options, we can use the exclusion principle. A merge is out of the question due to two main reasons. First, it would get us back to where we came from, i.e. the original red cluster, which needed adjustment. Second, one of the obtained clusters has a good composition and the distance indicators—the only meaningful measure for relation in this two-cluster setting—shows reasonable separation. Using refine would only be appropriate for the already well defined smaller cluster but not the other, which basically leaves split.

Although this seems an option, the attribute view kind of contradicts this idea. Figure 6.12(a) shows its compare track, where we can see that the bigger pink cluster overlaps with the red cluster in each



(a) Newly created clustering. (b) Comparison of the two new clusters.

Figure 6.11: Results of the restructuring.



(a) Cluster comparison on the attribute level.

(b) Result of further split adjustments.

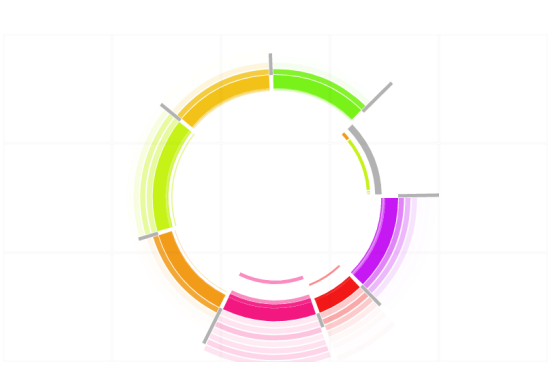
Figure 6.12: Adjustment of the restructured clusters.

dimension. Furthermore, it covers the full range of the global dimension in contrast to its more local sibling. The histogram shows a nearly uniform distribution which makes further division unlikely. To illustrate what happens if this adjustment is applied nonetheless, we issued a sequence of two splits. The first yields a larger and a smaller child, of which the large is divided further due to its characteristics. The resulting clustering features six clusters and is illustrated in Figure 6.12(b). All in all, it is not a satisfying result: nearly a quarter of all objects is assigned to the noise cluster, half of the clusters are very small and probably insignificant due to the already reduced size of the dataset. Despite all adjustments, halos show that all clusters need adjustment. Even the well formed cluster that was left untouched is compromised due to the increased number of clusters in its proximity. As no improvement is gained we discard the steps and go back to the initial clustering of step 9. The failure of split also rules out the option of another restructure, which means we have to accept the current state as we cannot improve it. With this, we move on to the last step of the tour.

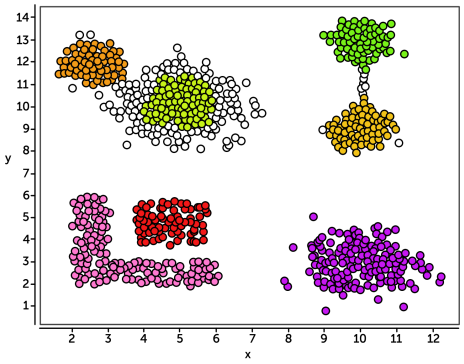
Step 10: With restructure, we created a new branch of our process for a subset of the original data. In order to interpret the structures found in the context of the whole clustering, we have to get them back to the trunk of the process. For this we use the integrate command, which is the counterpart of restructure. At the moment, we just allow a branching depth of one, i.e. after a restructure is issued, it cannot be applied again before the created branch is incorporated into the trunk. As integrate replaces restructure in the feedback menu, the number of available commands remains constant.

After integration is finished, our process comes to an end with the final clustering result shown in Figure 6.13(a). Most of the clustering is in good shape, with the two newly integrated clusters being the exception. As both are located near the purple cluster, they influence it, which results in an increased outer halo. However, regarding our available options and the experience gained during process execution, we can assume that the current clustering states the best result possible under the circumstances.

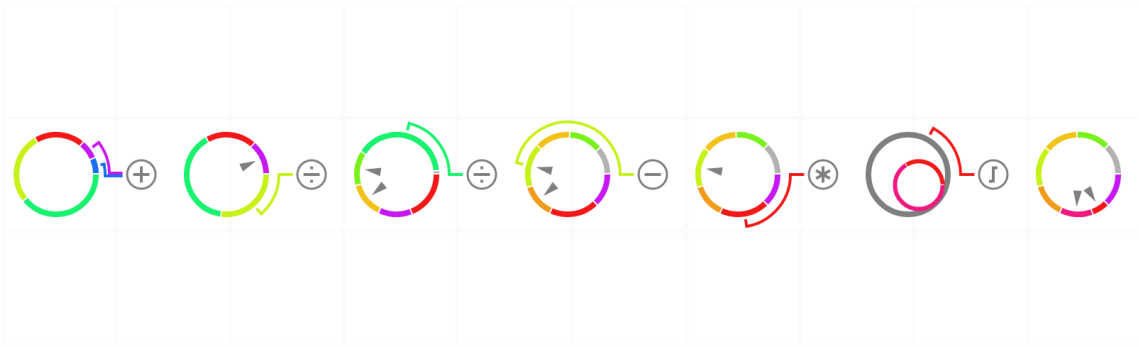
To get a better idea of said circumstances, we depict the final result with a scatterplot in Figure 6.13(b). There we can see how our final clusters are located in the dataset. The three clusters on the right side are the result of the first two steps of our guided tour. The purple one was created with the merge in step 1, while the green and yellow ones were obtained via the split in step 2. All these clusters are spherical and thus can be identified with k-means. In contrast, the clusters on the top left that made up the original green cluster we split during steps 3 and 4, have a more problematic structure. Although both clusters can be distinguished as separate groups, their direct adjacency makes it hard for algorithms like k-means to separate them. However, with our approach we were able to separate them, although the yellow-green cluster was reduced down to a compact core. The reason for this is our employed relation measure, from whose point of view both clusters are not clearly separated due



(a) Clustering view of the final result.



(b) Scatterplot of the final result.



(c) Complete process sequence.

Figure 6.13: Final results of our exemplary process application.

to their proximity. Due to this, the yellow-green cluster was reduced to its compact core, which improved its composition and separation. Regarding the clusters in the lower left, it is easy to understand that restructure had to be applied there. Due to their shape, k-means is effectively unable to separate them. Only the combination of multiple clusterings via integration, allowed their identification. This also shows that our decision to keep a solution that was not perfect but could not be improved further under the given conditions, was correct. All adjustments taken during our exemplary application are summarized in Figure 6.13(c), using the process view. This view allows the involved user to retrace his actions and furthermore enables other users to understand the adjustments that lead to the final result.

6.2 LEARNING BY DOING - PROCESS ADAPTATION

The compact and direct nature of our feedback operations allows users to apply appropriate adjustments and resolve unclear situations, using the principle of exclusion. However, as seen in our example, we can run into dead ends that we cannot escape in a satisfying way, using only the primary level of control. In such cases, we typically have to accept certain compromises regarding the result. Although this lets our process look limited at first sight, it is actually not a drawback. During iterative refinement, we do not only gain information from the clustering result, but also from the course of action itself. The way in which adjustments are applied and results are produced, provides us with knowledge that can be used to adapt and improve our process. So every time we run into a dead end, i.e. conclude the clustering process with an unsatisfying result, we can follow a train of thought that is defined by the modular character of our process and its components. Subsequently, we present a short guide on how this is done.

If we hit a dead end on the primary level of control, there are three possibilities to escape it. The first one is ensemble configuration, i.e. the secondary level of control for the algorithm platform. Our example from the previous section shows that certain structures inside the data cannot be identified by certain algorithms. This prevents satisfactory result adjustments and can only be resolved by employing a different ensemble configuration. Inevitably, we are confronted with the question of algorithm and parameter selection. These tasks can be done more or less sophisticated, depending on the available level of experience. However, if user experience is low and no expert is available, we suggest the following essential configurations. We know that there are certain algorithm families that share certain traits, which designate them for certain data characteristics. Unfortunately, this classification is extensive and variable. Thus, maintaining an ensemble configuration for each class is not really feasible. For this reason, we used our building blocks approach to find the most basic difference between clustering algorithms that can be used to create a taxonomy. Our examination starts with the different phases of an algorithm. Optimization and association are too individual and, thus, cannot be used for our classification. Selection is also sorted out as there are unlimited options for filters and conditions, which leaves evaluation. The defining parts of this phase are the distance measure and its inputs. While the function itself is too variable, we find our desired basic difference in the relation between its inputs, i.e. points and references. There are two options: either points and references are identical or not. Based on this, we can create two base classes that divide all existing clustering algorithms. Now we can create one ensemble configuration for each class, which should provide enough versatility to tackle most datasets. Our proposed configuration for the class in which references and points are not equal is k-means. For the second class we suggest DBSCAN. The selection of parameters for both configurations is greatly eased by the structure of our process. As we use ensembles, single parameterizations do not have a critical influence. As long as a parameterization creates a non-trivial clustering, i.e. a result with more than one and less than n clusters, it contributes useful structural information that can be exploited during integration and iterative refinement.

The second way to escape a dead end is to switch the method of integration. Changing the way in which the structural information from the ensemble is combined, also changes how adjustments are realized. Thus, previously impossible improvements can be enabled. Due to the modular character of our process, this switch—e.g. from flexible clustering aggregation to frequent groupings—is easy to perform and does not even require a re-calculation of the ensemble.

The third and last escape option is located in the hybrid-visualization. Result interpretation strongly depends on the measures used to present cluster composition and relations. In our small screen visualization we employed soft assignments for this task. These were designed with spherical clusters in mind, which means from their point of view freeform structures like the L-shaped cluster from our example will never be considered as decent clusters. This issue can cause phantom dead ends, where a valid cluster exists but the user is not able to recognize it as such, due to biased measures. As our visualization concept is also modular, this issue can be fixed by introducing a different point of view by employing alternative measures to present composition and relations. We suggest a measure aimed at freeform clusters to complement the already used soft assignments.

These three options provide a large potential for adaptation of our process. When a user encounters a dead-end situation they should be used in the following order: First, the point of view in the visualization should be changed to clear potential mismatches during interpretation. Second, the method of integration should be changed, as this can enable new options without changing the existing ensemble. Third, the ensemble configuration should be switched. These options should be able to resolve almost any issue a user may encounter.

6.3 SUMMARY

The components described in the previous chapters enabled us to put together a realization of our clustering process. It was used on a small example dataset to exemplify the procedure of iterative refinement in the form of a guided tour. In the course of our description we regarded every action taken by the user and illustrated each step with the corresponding view from our mobile visualization. Particular attention was given on the derivation of appropriate feedback operations and the resolution of unclear situations. In addition to this, we considered the aspect of process adaptation to different settings. For the event of iterative refinement being stuck in a dead-end situation, we proposed a simple course of action that utilizes the modular character of our process to adapt it, and escape the consequences. Options for changing ensemble configuration, switching integration methods, and introducing alternative points of view into the hybrid-visualization make our clustering process highly versatile.

EVALUATION

7.1 User Study Setup

7.2 Application Results

7.3 Usability Results

7.4 Summary

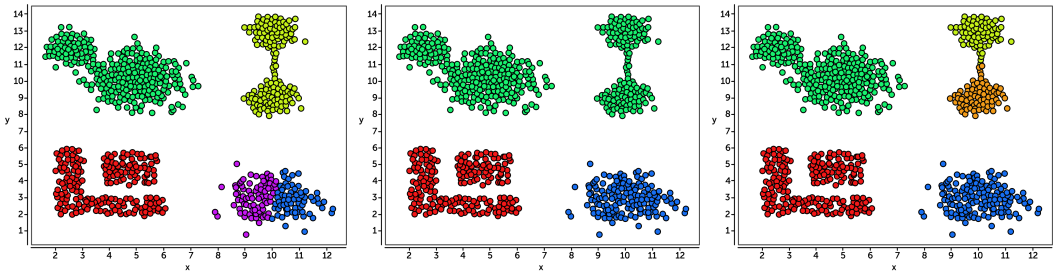
Objective and meaningful evaluation is the Gordian knot of clustering. Typically, it is aimed at result quality or runtime performance to rate how good and/or fast an algorithm performs. In contrast to other data mining techniques like classification or frequent itemset mining, clustering knows no ground truth that can be used to assess and compare performance. Basically, clustering tries to discover previously unknown knowledge by identifying structures in the data. With this, a clustering result must be considered more as a hypothesis rather than a fixed statement. Therefore, result evaluation and validation must be done separately and in accordance with the application domain from which the processed dataset originated. Despite these characteristics, there exists an understandable desire for quality measurement in order to emphasize the benefits of a newly proposed approach in comparison to existing work. This lead to the establishment of two particular evaluation procedures for traditional clustering algorithms that are widely used in the contemporary clustering practice. The first one involves a dataset with a known gold standard regarding its partitioning. This gold standard acts as ground truth for comparison and can be generated by either creating synthetic data with 'handmade' clusters, or using realistic data that is partitioned by a domain expert. Based on this, the performance of different algorithms is rated by comparing the results they produce with the gold standard solution. The second approach does not create such a gold standard, but employs quality measures like Dunn's Index [18] to score the performance of different algorithms in a relative fashion.

Unfortunately, both approaches do not always produce fully reliable results. As mentioned earlier, quality measures are scenario-specific and are based on a specific model of cluster quality. If this model does not fit the distribution of the underlying data or the cluster model of the used algorithm, quality measures become unreliable [35]. Working with a dataset and a gold standard can introduce a bias to evaluation because algorithm parameterization can be optimized with the optimal solution in mind. This does not represent a realistic application setting, in which the ground truth is unknown and cannot be utilized for modifications. Although they lack objectivity and resilience, these approaches still constitute the most common way of evaluation in the area of clustering.

Regarding the evaluation of our clustering process, we have to rethink the existing procedures. Although our process ultimately produces a clustering result, whose quality is of interest, it is not its single main goal. The focus of our process is to provide a versatile and easy-to-use way to create and modify clusterings. In doing so, it does not define new clustering algorithms, but provides a platform for the application of existing ones. Due to this situation, it is more suitable to assess the journey than the reward, i.e. we focus our evaluation on the production of the clustering. An optimal way to do this, would be to make an implementation of our process available to users in various application domains and let them work with it. The performance of our approach could then be measured via feedback regarding usability, cross-validation of obtained results by domain experts, and the knowledge or publications that emerge during its application. Unfortunately, this is not possible in the timeframe of one thesis, as such a deployment requires the release of an implementation that is nearly a market ready product. Development of the necessary methods and their prototypical implementations took several years. Realization of this evaluation approach would at least require an additional two years for completing the implementation and giving the community time to work with it. Since these time requirements cannot be satisfied, we use a scaled down version of this evaluation approach and execute a small user study.

7.1 USER STUDY SETUP

As already mentioned, the goal of our process is to provide an easy and usable way for producing clusterings. In order to measure how well this goal is achieved we execute a user study that consists of



(a) Starting solution A. (b) Starting solution B. (c) Starting solution C.

Figure 7.1: Starting solutions for the user study.

two main tasks, which the probands have to complete. The first one is to actually work with an implementation of our process. For this we use the setting from Chapter 6 again. Our mobile visualization is used as visual-interactive interface, while the flexible clustering aggregation handles integration in the algorithm management. We also use the same dataset, as it is synthetic and we know its optimal partitioning, which allows us to rate the quality of results to a certain degree. With the first task we want to show that our approach of interaction and iterative refinement enables users to create a satisfying clustering solution from a variety of situations, even if the starting point is not optimal.

For this, three different ensembles based on k-means were configured to create different starting solutions for iterative refinement. Scatterplots of the resulting initial clusterings are shown in Figure 7.1. From left to right, the three ensembles were designed as follows. Solution A originates from the same ensemble already used as an example in the previous chapter. For solution B, the underlying ensemble is configured with small values for k. This not only reduces the number of clusters, but also the degree of structural information available for split operations. Assuming solution A as a baseline, solution B constitutes a more difficult starting point. It requires more adjustment steps, which also include multiple restructure operations to get additional information for the clusters at the top. In contrast, solution C offers an easier starting point as it only requires users to adjust the red and green clusters on the left. We divided the 15 probands that took part in our study in three groups by randomly assigning 5 users to each ensemble/starting solution. To assess the performance of our process we examine the obtained results and also monitor the number of issued adjustments and their sequence in order to assess how users were able to solve the task.

Before the test persons actually started working with the system, they were given a short tutorial on its application. This included a 15 minute demonstration and a handout illustrating the gestures used to control the visual-interactive interface. All probands were monitored during their work and could request help on controlling the system.

In the second task, our test users were asked to provide feedback regarding the usability of our process. To measure the perceived usability we use the widely popular *System Usability Scale(SUS)* [12]. The SUS is a simple questionnaire with ten questions. Each one is represented by a five-point Likert item that allows answers on a scale ranging from *strongly disagree* to *strongly agree*. Assessment of the provided answers results in a single score that can be used to rate the usability of the evaluated system. Figure 7.2 shows the SUS questionnaire.

7.2 APPLICATION RESULTS

Subsequently, we present the final clustering results that were created by the probands of each of our three groups. For the purpose of illustration we provide scatterplots of all solutions and present some statistics regarding the number of applied adjustments and how often feedback options were revoked using the redo feature.

System Usability Scale

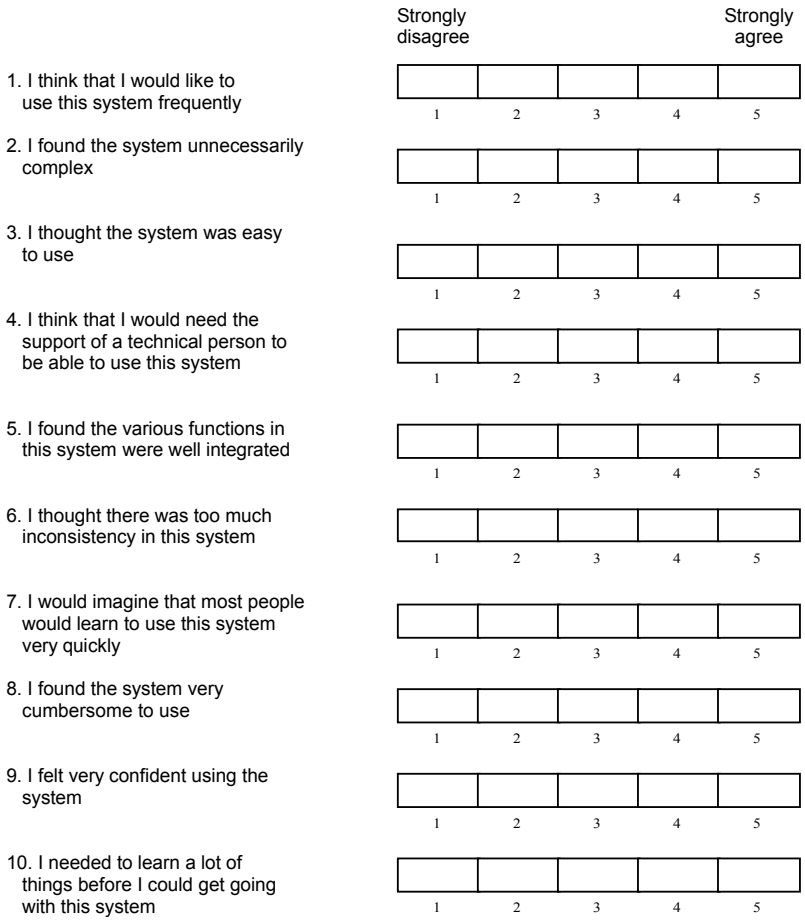
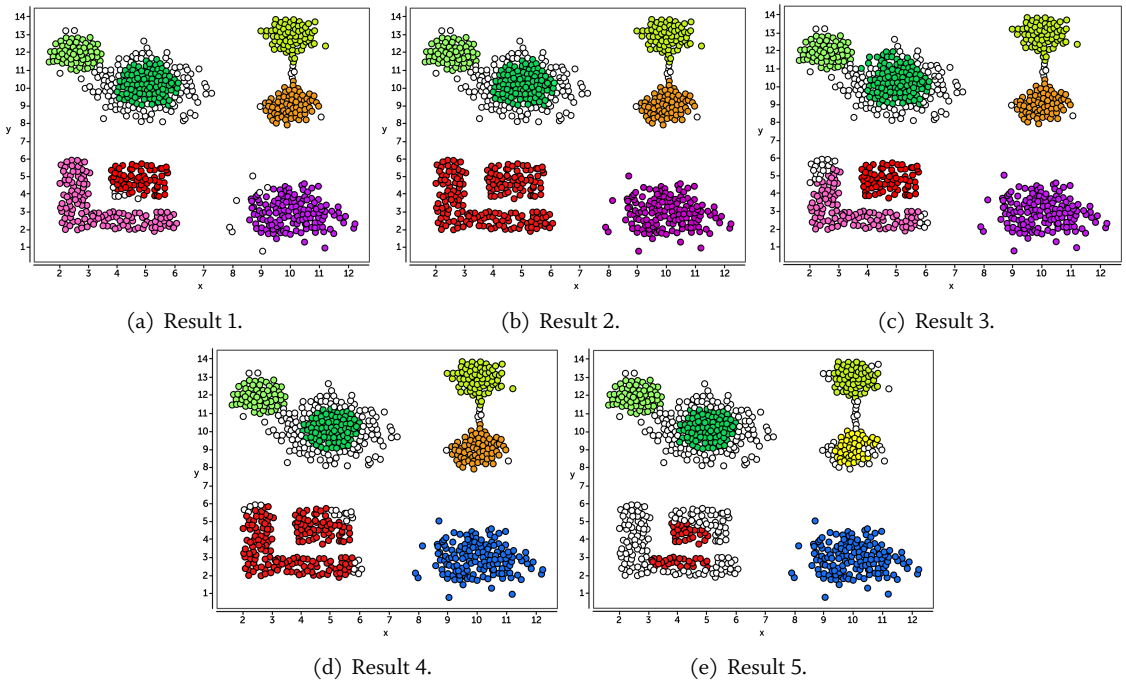


Figure 7.2: SUS questionnaire.



(a) Result 1. (b) Result 2. (c) Result 3.

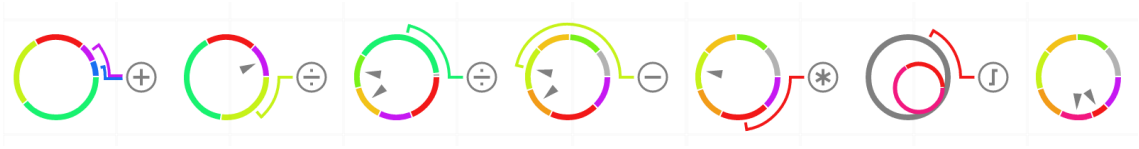
(d) Result 4. (e) Result 5.

Figure 7.3: Refined clusterings for group A.

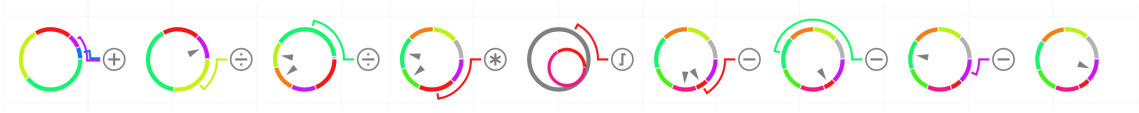
7.2.1 Group A

Figure 7.3 depicts the results obtained from starting solution A. We can see that the adjusted clusterings are in general pretty good. All users correctly used merge to combine the two initial clusters in the lower right quadrant. Furthermore, the split feedback was used to divide the clusters at the top. Regarding the adjustment of the red cluster in the bottom left users had different opinions, which was expected. As mentioned in Chapter 6, this cluster cannot be adjusted perfectly with the employed process configuration. By applying restructure, the cluster is divided into two new clusters. From the user's point of view, one of these shows an improved composition while the other's composition and relation get worse. In this situation the test users made different decisions. Two regarded the improvement of one cluster as sufficient and kept the adjustment which leads to results 1 and 3. The other two judged that no improvement was achieved and decided to go back to the previous mediocre version of the cluster. Regarding the adjustment of said cluster, result 5 shown in Figure 7.3(e) constitutes an outlier. The test person also used restructure and decided to take it back. In contrast to the remaining users, the proband did not accept the reverted cluster state and applied a split with strong force that relocated more than 70% of its members to noise. Due to this, the cluster was reduced to a small core which naturally lead to an improvement composition and relation. Although all participants were informed that major amounts of noise disprove a successful adjustment, the proband stayed with his decision.

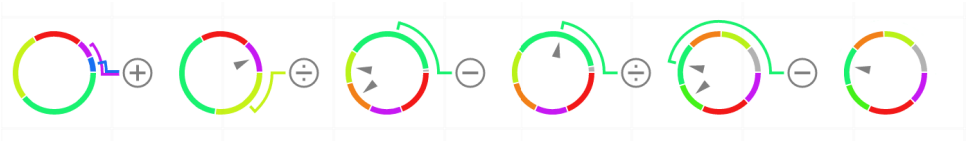
Next, we examine the number of adjustments that were applied to reach the presented solution. We already ran through this example in the previous chapter and introduced a corresponding adjustment sequence, which is used here as reference. Figure 7.4 illustrates the adjustment sequences for each clustering created in group A. The sequences that lead to the results 1 to 4 are pretty close to the reference and range from 5 to 8 adjustments. In general, they contain the mandatory merge and split operations, but differ regarding the restructure feedback. Although each user issued this operation, only two kept it for its improvements. Again, result 5 states an outlier as it contains 13 adjustments. However, the difference mostly lies in the number of applied refine operations—6 in this case—which represents the individual degree of fine-tuning users are willing to employ. The number of redo's, i.e. adjustments that were taken back, ranges from 1 to 6 with an average of 3.



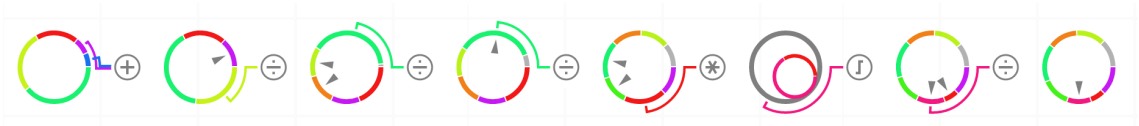
(a) Reference from Chapter 6.



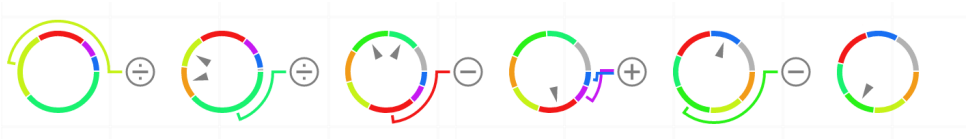
(b) Adjustments leading to result 1.



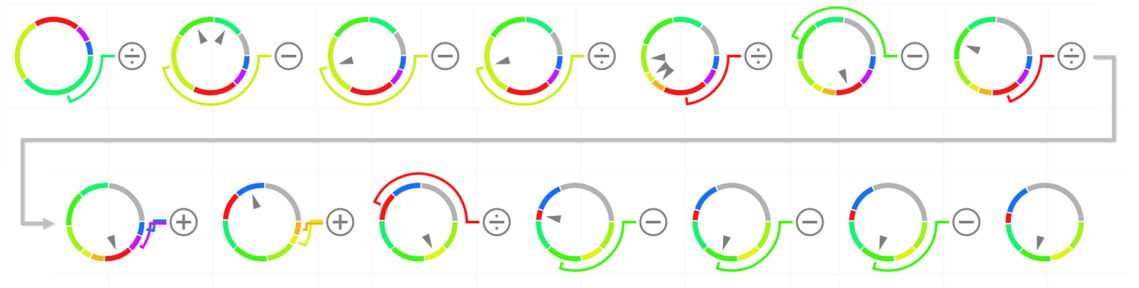
(c) Adjustments leading to result 2.



(d) Adjustments leading to result 3.



(e) Adjustments leading to result 4.



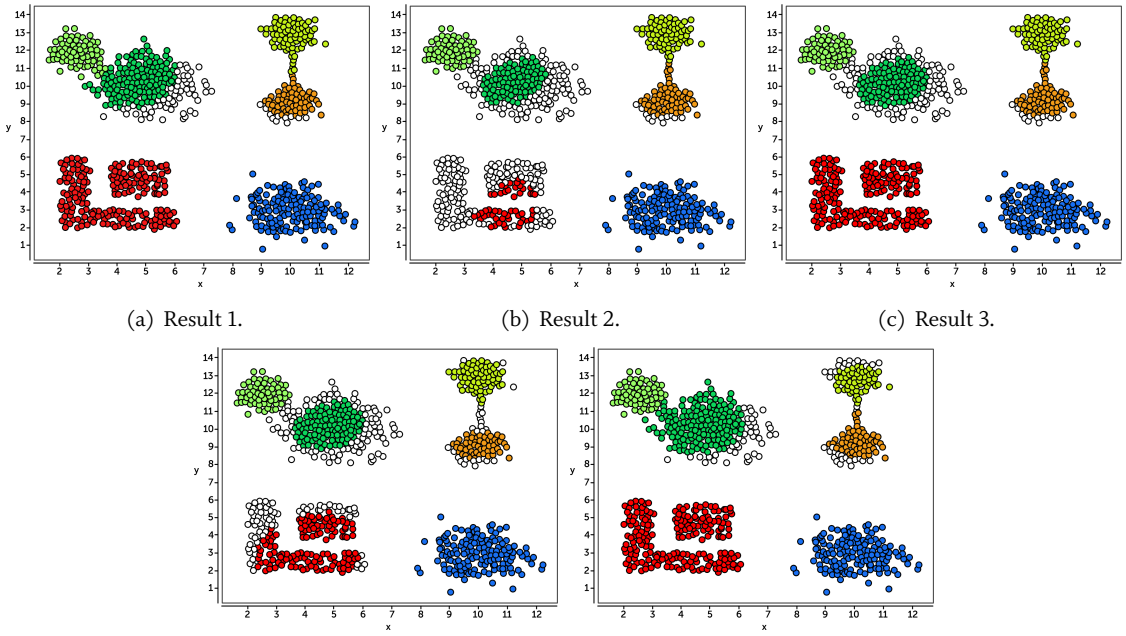
(f) Adjustments leading to result 5.

Figure 7.4: Adjustment sequences for group A.

7.2.2 Group B

Now we examine the results from the 'difficult' group. As before, scatterplots of all obtained clusterings are shown in Figure 7.5. Like with group A before, we can see that the major adjustments were applied, despite the disadvantageous starting situation. All of the test users correctly identified the need to split and restructure the top clusters and achieved similar results. Like before, the adjustment of the red cluster in the lower left proved challenging. In contrast to the previous group, all users decided to keep the cluster as one. Some applied the restructure command, but took it back. Again, one user applied split/refine commands to greatly reduce cluster size as can be seen in Figure 7.5(b).

As expected, our test users needed to apply more feedback operations to reach a satisfying solution than in the previous group. Due to this, adjustment sequences become very long, which is why we



(a) Result 1. (b) Result 2. (c) Result 3.

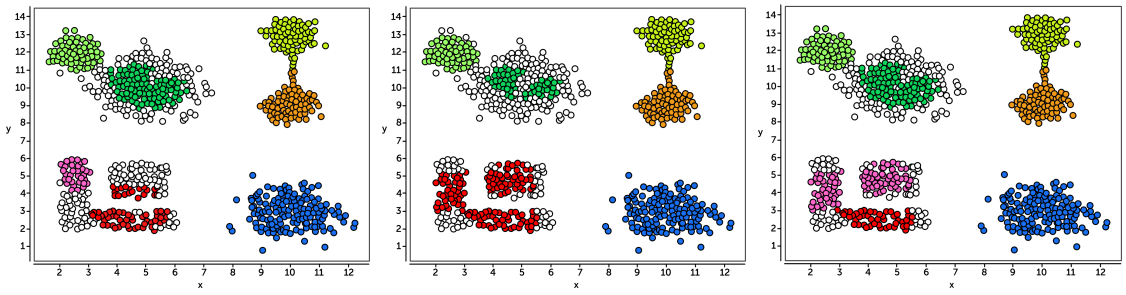
(d) Result 4. (e) Result 5.

Figure 7.5: Refined clusterings for group B.

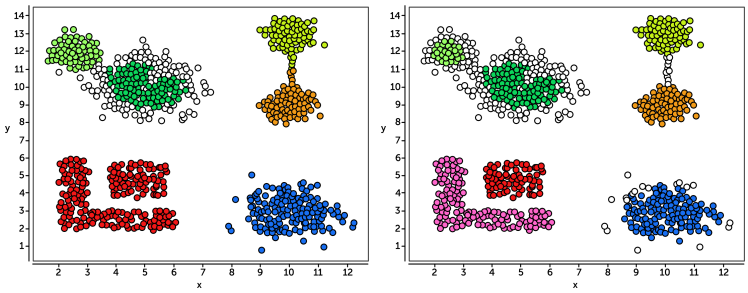
result	adjustments	redo	merge	split	refine	restructure	integrate
1	8	7	3	1	0	2	2
2	14	1	4	4	2	2	2
3	14	1	7	2	1	2	2
4	18	0	8	3	3	2	2
5	16	3	8	3	1	2	2

Figure 7.6: Adjustment sequences for group B.

use a shortened notation for their description. The table in Figure 7.6 shows the number and type of adjustments issued for each result. Furthermore, it lists how many redo operations were used to withdraw unsuccessful adjustments. Please note that the adjustment column contains only the number of kept adjustments and does not account for the number of withdrawn steps. All test persons started by applying a split to the large initial cluster at the top. With this, two new clusters were created in the top left and right quadrant. In order two refine these clusters further, it was necessary to apply the restructure operation to each of them. While all users did this, a notion of randomness was introduced during restructuring of the top left green cluster. This randomness originates from a safeguard that we implemented in our restructure operation. When the initial ensemble configuration is applied to a smaller subset of the data, there is a small chance that the different resulting clusterings overlap in a way that leads to a clustering solution with only one cluster. In this situation, the focus on pairwise assignments during flexible clustering aggregation causes transitive effects that connect all clusters of the ensemble. Should this case occur, our safeguard starts a loop that randomizes initial centroid positions and reduces the values for k until a solution with more than one cluster is obtained. In our example this safeguard is activated during processing of the green cluster and causes restructure to produce different numbers of clusters. This requires users to issue a different amount of merges to refine this region of the dataset. Despite this randomness, the final results show that all probands were able to handle the situation with the tools provided by our process and arrived at the same conclusions. Restructuring of the top left cluster was not affected by this issue and was identical for all users. Please note that this issue is not present in our frequent groupings approach.



(a) Result 1. (b) Result 2. (c) Result 3.



(d) Result 4. (e) Result 5.

Figure 7.7: Refined clusterings for group C.

result	adjustments	redo	merge	split	refine	restructure	integrate
1	11	0	4	2	1	2	2
2	4	5	2	2	0	0	0
3	2	7	0	2	0	0	0
4	1	1	0	1	0	0	0
5	15	4	5	3	4	2	2

Figure 7.8: Adjustment sequences for group C.

In general the performance of group B was quite good. Due to the mentioned reasons the number of applied adjustments ranges from 8 - 18, which is considerably higher than group A. This was expected and originates from the more difficult starting solution. Again, all test persons performed the same major adjustments—split of the top cluster and two restructures on its descendants—and differed only in the treatment of the red cluster and the number of issued refine commands. Interestingly, the number of redo's was a little lower, ranging from 0 to 7 with an average of 2.4.

7.2.3 Group C

The starting solution for group C was intended as an easy starting point that only requires the adjustment of the two clusters on the left. To our surprise, the test users of this group struggled quite a bit during the refinement of the starting solution, which resulted in considerably different adjustment sequences. The final results for this group are shown in Figure 7.7. Again, we can see that all users correctly adjusted the top left cluster while the red cluster was treated differently. Like before, three ways for its treatment could be observed. In the first three results, users issued split or refine operations to improve the cluster. Due to the configuration of ensemble C, this actually leads to the creation of two new clusters in result 1 and 3. The remaining two users choose to leave the cluster as one—Result 4—or applied the restructure command—Result 5—to further improve it.

While users of the previous two groups naturally created very similar adjustment sequences, the members of group C worked in a very individual fashion as can be seen in Figure 7.8. One user started with an unnecessary merge that was not indicated by the visualization at all. Subsequently, he applied merge operations to basically restore the initial cluster configuration, before he applied the obvious adjustments. This resulted in a high adjustment and redo count. Another user issued the two necessary splits, but employed too much force. This lead to some poorly separated child clusters that were merged again, finally producing result 2. The third user only kept the two necessary split adjustments, but played around with the force value which lead to a high redo count. User four was the most efficient and only used one split to improve the result. Finally, the fifth user took a totally different approach and used the whole palette of feedback operations. His adjustments produced result 5, which is shown in Figure 7.7(e) and constitutes the most accurate partitioning of group C. All in all, this group showed the most variability in the refinement procedure with an adjustment count ranging from 1 to 15 and a redo count ranging from 0 to 7 with an average of 3.4. Although we do not know what caused the diversity in this group, we were pleased to see that, regardless of the way they took, our process guided all users to similar final results.

7.3 USABILITY RESULTS

After each proband was finished working with the clustering, he/she filled out the provided SUS questionnaire. In order to calculate the total usability measure, each answer is evaluated and provides a score from 0 to 4. For each sheet, these scores are summed up and multiplied with 2.5 to get a final score that ranges from 0 (negative) to 100 (positive). This scale is very intuitive and can be used for the relative comparison between different approaches, stating which one is better in terms of usability. However, an absolute judgment on usability cannot be made with this score. In order to solve this issue, several approaches to establish an absolute scale have been made. Mostly, these analyze a large number of SUS surveys, assume the most frequent score as average and build an absolute scale around it. In this section, we use the absolute scale proposed in [9] as a reference for our scores. The authors added a seven-point Likert item as the eleventh question to nearly 1000 SUS surveys. The additional question 'Overall, I would rate the user-friendliness of this product as' offers the choices: Worst Imaginable, Awful, Poor, OK, Good, Excellent, and Best Imaginable. Based on the survey results, the authors assigned a score range to each of these classes.

Average usability scores from our user study are shown in Figure 7.9. We can see that the overall score for our approach is 72.7. According to the absolute scale from [9], this puts out approach in the 'Good' class, which is defined with a mean score of 71.4 and a standard deviation of 11.6. On closer inspection, we found that scores vary considerably between the different groups. As shown in Figure 7.9 group B has the lowest average score of 68. A reason for this could be the higher difficulty of the corresponding starting solution. Users had to perform a high number of adjustments during iterative refinement, i.e. reaching the intended goal was more tiresome than in the other groups. Maybe this had a negative impact on the perceived usability. The highest score of 78.5 was obtained from group A and could be placed in the 'Excellent' class of the absolute scale which has a mean of 85.5 and a standard deviation of 10.4. Again, the reasons for this could be located in the ensemble/starting solution. The lower average number of adjustments and the general conformity of adjustment sequences in this group show that users could easily deal with the given setting. Located between these extremes, we find group C with an average score of 71.5 that fits the diverse character of this group.

In order to identify the areas of our approach that still need improvement we closely examined the provided answers. Figure 7.10 shows the average scores for each of the 10 questions. We can see that most of the scores are larger or equal 3, which is pretty good considering a maximal value of

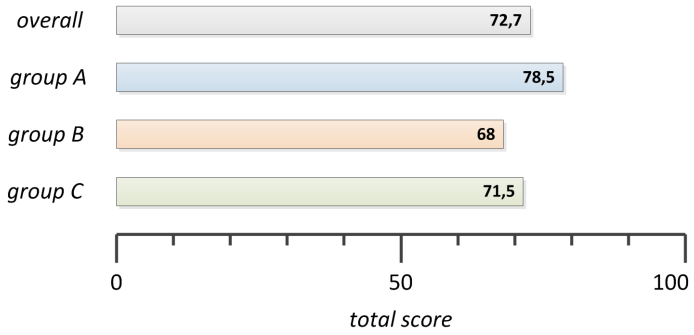


Figure 7.9: Total SUS scores from our user study.

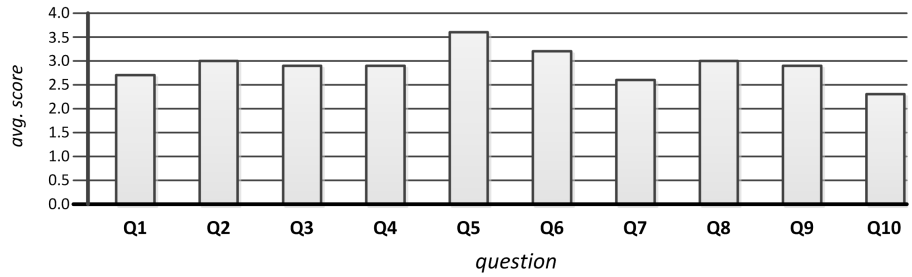


Figure 7.10: Average score per question.

4. However, questions Q1, Q7, and Q10 fall short of this mark and provide the three weakest scores. Question 1 is given as 'I think that I would like to use this system frequently', which was hard to answer as nearly all of our probands do not work in the area of clustering. The two remaining questions consider how much people need to learn in order to use the system. The lower scores for these points show that our short verbal introduction did not communicate the operating instructions for our process in an optimal way. This issue could be improved by adding some sort of on-screen help that would provide user support.

All in all, the obtained SUS scores indicate a very good usability for our process. However, many of our test persons noted that the SUS questionnaire was lacking some features. For example, many users liked the overall interaction concept, but were not fully satisfied with the implementation of some gestures. These users criticized that they had no option to justify their ratings. Other probands missed a rating option for 'Fun of use'. Interestingly, we observed that several users treated working with our process as a game. Successful cluster improvements made them happy and motivated them to further adjust the clustering. They were also eager to obtain the optimum of each cluster and were disappointed when they could not reach this goal. This characteristic has significant potential for the further development of our process. Gamification of our process could be used to increase user engagement and motivation which could lead to better usability and result quality.

7.4 SUMMARY

Due to the character of clustering, meaningful evaluation is a big challenge. Traditional clustering algorithms are typically evaluated solely with regard to result quality. This is problematic as clustering aims to uncover unknown knowledge and, thus, knows no ground truth. Because of this, evaluation results are often unreliable. Furthermore, this approach is unfitting as our process does not define a

new clustering algorithm, but a versatile and easy-to-use clustering procedure. For an optimal evaluation we would have to release an implementation of our process to the community. Users from different application domains would then work with our approach and provide feedback regarding usability and their obtained results. As such an evaluation would be too complex and time-consuming for the scope of this thesis, we carried out a small user study in order to assess the core features of our approach. In a first task, we let users apply our process to an exemplary scenario. Test persons were divided into three groups and each was given a different clustering ensemble and starting solution for the same dataset. The obtained clustering results show that our process enables users to produce correct clusterings even from different disadvantageous situations. As a second task we used the SUS questionnaire to rate the overall usability of our process. Although it was rated as good, we closely examined the provided answers and identified minor flaws as well as ways to clear them out.

CONCLUSION AND OUTLOOK

The abundance of data in modern life and its continuous collection have made data-mining and knowledge discovery challenging, but mandatory tasks. Out of all data-analysis approaches, clustering is one of the most fundamental techniques for the extraction of unknown knowledge. By partitioning a set of objects into groups according to their similarity, structures that represent possible semantic connections are identified. Although the concept of clustering is clearly defined, its solution is extremely challenging due to the variable character of similarity, its subjective perception, and the absence of a ground truth, which is inherent when extracting new knowledge. To deal with these uncertainties, specialization has been a formative trend in clustering-algorithm design for the last years. Narrowing down the application scenario allows to incorporate domain specifics into the clustering algorithm which often leads to improved performance, but greatly reduces versatility and increases development costs.

For these reasons, contemporary clustering offers a multitude of specialized tools, whose application requires a high-level of expertise. As large amounts of data emerge in more application domains, clustering has become a widespread analysis technique and new users must utilize it. With this development, characteristics like usability and applicability have gained importance. The application of clustering in its current state requires amateur users to either hire experts or try building a fitting solution by trial-and-error. This thesis aims to solve this dilemma by structuring and integrating the necessary steps of clustering into a guided and feedback-driven process.

To reach this goal, we first gave a brief overview of contemporary clustering and its basic steps. We described different available approaches to clustering and some of their associated methods. During an assessment of these existing techniques, we outlined their limitations regarding applicability and usability for non-expert users. We came to the conclusion that a lack of background-knowledge and the vast amount of necessary critical decisions are the main problems emerging during the creation of a clustering. These issues also lead to an iterative mode of operation in which more or less random variations are introduced during each step with a negative influence on result quality and user satisfaction. To tackle these challenges, we completely reconsidered the current practice of clustering. Individual steps were tightly coupled and evolved into a basic concept of a versatile clustering process. Its template contains two major components: the algorithm management that handles clustering creation and the visual-interactive interface for user interaction. Both work in concert to allow users the iterative refinement of a clustering. These defined requirement were realized by proposing novel methods and techniques.

For the algorithm management three major tasks were defined. First, a modular approach for the description and design of clustering was proposed. We defined the characteristic phases of each clustering algorithms and its basic building blocks. These were refined into a vocabulary, based on functions and matrices. Its mathematical syntax provides a consistent way of description that is formal, but not too specific and still allows the recognition of the abstract tasks an algorithm performs.

Second, we proposed integration techniques to make our process versatile and enable it to incorporate a variety of clustering algorithms. We dismissed the traditional single-execution paradigm and choose the concept of ensemble clustering, which works with sets of multiple clustering configurations. We evolved it from a passive post-processing technique into an active and controllable method and described two novel integration methods: Flexible clustering aggregation works on the fine-grained level of object pairs and incorporates soft cluster assignments, an extended pairwise similarity concept, as well as a scoring function for reliability. With this, a way of direct control over the aggregation procedure was established and made it the focal point of action for working with the clustering. The frequent groupings approach was proposed as a more coarse-grained technique to realize integration. It constructs a consensus solution by identifying and combining prevalent sets of objects. This allows the construction of multiple robust clustering alternatives and tackles some drawbacks of pairwise

methods. Similar to the flexible clustering aggregation it allows direct adjustments of the consensus clustering and does not require users to deal with the underlying clustering ensemble.

As clustering creation is a vital part of our process, users must be able to control it. To relieve users from complex interactions and keep away most technical details, an understandable and easy-to-use control interface was proposed. It structures all available control options into multiple levels and acts as connection point for the visual-interactive interface, which is the second major component of our process. It constitutes the main point of interaction between the user and our clustering process and consists of two parts: high-level feedback for result adjustment and a hybrid visualization concept for result interpretation. The high-level feedback contains four commands: merge, split, refine, and restructure. In contrast to the variable technical parameters of traditional clustering, our feedback is universally valid and describes direct effects in the clustering result. To achieve independence from the underlying algorithms, specific mappings between our feedback and the methods of the algorithm management were described.

In order to enable users to interpret the clustering result and derive appropriate adjustments, a hybrid visualization concept was proposed. It was designed as a middle way between overly detailed data-driven visualizations and too general result-driven quality measures. The concept focuses on the cluster level and communicates two abstract aspects: composition and relations. While composition represents the homogeneity of clusters, relations show their separation. In addition, traditional single-display visualization was unraveled into multiple views that communicate a reasonable level of detail without overburdening users. These characteristics make our approach immune to high dimensionality and clutter induced by high volume data.

As the hybrid visualization concept itself is not an actual visualization, but a template its construction, we developed two implementations of it to illustrate the capabilities of our concept. The first visualization was aimed at stationary desktop platforms with large screens and conventional input. It provides users with three simple and clear views for result interpretation and fully supports our high-level feedback. The second implementation was aimed at the small screens of smart devices and takes advantage of the intuitive touch-based input they provide. The view structure was redesigned, while features for cluster comparison and the tracking of changes were added. With all this, users are provided with a tool that is powerful, but still convenient to use and easy to understand.

Finally, all proposed components were assembled into a realization of our clustering process. A small example dataset was used to exemplify the application of iterative refinement. During this guided tour, every action and its corresponding visualization views were examined and described. Furthermore, the derivation of appropriate feedback operations and the resolution of unclear situations were discussed. In case the iterative refinement gets stuck in a dead-end situation, a simple adaptation approach can be utilized to modify our approach. Taking advantage of the modular character of our clustering process, it contains options like changing ensemble configuration or switching integration methods to ensure a high degree of versatility.

To evaluate the overall performance and usability of our process, a user study was designed and carried out. It consisted of two tasks that were performed in succession. First, users had to apply the proposed process in one of three exemplary scenarios. While each scenario used the same dataset, different clustering ensembles were employed to create varying starting solutions. Although some of these initial situations were more challenging than others, our clustering process enabled users to produce correct clusterings. In the subsequent second task, users had to fill out a SUS questionnaire to rate the perceived usability of our process. The obtained results showed a very good overall usability and pointed out some minor issues that can be easily fixed.

In this thesis we introduced a template for a feedback-driven clustering process and proposed the necessary components to realize it. With it, amateur users are enabled to easily create and adjust

clustering results. Our process-centric approach to data clustering is still a novelty and this thesis only describes possible options to implement it. For these reasons, there is a lot of potential for further research regarding optimization and the addition/expansion of features. In this section, we will name a few research questions for the whole process and its core components.

We begin with the algorithm management and its building blocks approach for algorithm design. At the moment its main focus is description and specification. In the future we also want to consider the efficient execution of these descriptions on different platforms. For this, our unique matrix data model and matrix functions offer various interesting opportunities. Already, a lot of usable related research exists in different domains. For example, efficient large matrix computation has a long tradition as an area of research in high-performance computing. Furthermore, graphic cards and CUDA are strongly geared to matrix processing and seem to be an ideal target architecture for our approach. Besides these hardware-centric approaches, research activities in the database community try to enhance database systems by integrating mathematical constructs. For example, SciDB [13] is a native array DBMS that combines data management and mathematical operations. With this, it mixes statistical and linear algebra operations with data management operations. As this work demonstrates, such an approach brings several advantages from the perspective of *application* as well as *performance*.

The visual-interactive interface also holds different options for future research. In Chapter 6 we already mentioned that result interpretation strongly depends on the measures used to present cluster composition and relations. Until now, we employed soft assignments for this task, which were designed with spherical clusters in mind. Our example has shown that this technique has trouble working with freeform clusters. Therefore, the development of measures aimed at this kind of cluster, states an interesting research question. As density based clustering methods like DBSCAN [21] and DENCLUE [37] already contain ways to model freeform clusters, these techniques could be an interesting starting point for this research.

While we do not see a need to expand our high-level feedback options, researching different scopes for their application is an interesting goal. Limiting the application of feedback to a certain set of dimensions could provide a way for integrating functionality from the area of subspace clustering and improve the analysis of high-dimensional datasets.

When looking at the process itself, collaboration is an obvious area for further research. With our proposed approaches, the clustering process itself has become traceable and comparable. This means users that work on the same dataset can share and compare their adjustment sequences in order to get feedback and new ideas for further refinement. To support this it is necessary to develop concepts for the comparison of processes, the annotation of certain steps, and the creation of different process branches to track alternative courses of action.

BIBLIOGRAPHY

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD '98)*, pages 94–105, 1998.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*, pages 487–499, 1994.

[3] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD '99)*, pages 49–60, 1999.

[4] Almir Olivette Artero. Uncovering clusters in crowded parallel coordinates visualizations. In *In Proceedings of the 10th IEEE Symposium on Information Visualization (InfoVis 2004)*, pages 81–88, 2004.

[5] Eric Bae and James Bailey. Coala: A novel approach for the extraction of an alternate clustering of high quality and high dissimilarity. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006)*, pages 53–62, 2006.

[6] Xue Bai, Zezhen Lin, Yun Xiong, and Yangyong Zhu. Clustering based on yukawa potential. In *Proceedings of the 12th SIAM International Conference on Data Mining (SDM12)*, pages 672–683, 2012.

[7] Maria-Florina Balcan and Avrim Blum. Clustering with interactive feedback. In *Proceedings of the 19th International Conference on Algorithmic Learning Theory (ALT 2008)*, pages 316–328, 2008.

[8] G. Ball and D. Hall. Isodata: A novel method of data analysis and pattern classification. Technical report, Stanford Research Institute, Menlo Park, 1965.

[9] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of Usability Studies*, 4:114–123, 2009.

[10] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.

[11] Constantinos Boulis and Mari Ostendorf. Combining multiple clustering systems. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2004)*, pages 63–74, 2004.

[12] John Brooke. *Usability evaluation in industry*, chapter SUS: A "quick and dirty" usability scale, pages 189–194. Taylor & Francis, 1996.

[13] Philippe Cudré-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Roman Simakov, Emad Soroush, Pavel Velikhov, Daniel L. Wang, Magdalena Balazinska, Jacek Becla, David J. DeWitt, Bobbi Heath, David Maier, Samuel Madden, Jignesh M. Patel, Michael Stonebraker, and Stanley B. Zdonik. A demonstration of scidb: A science-oriented dbms. *Proceedings of the VLDB Endowment*, 2(2):1534–1537, 2009.

[14] Xuan Hong Dang and James Bailey. Generation of alternative clusterings using the cami approach. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM10)*, pages 118–129, 2010.

[15] Ian Davidson, S. S. Ravi, and Martin Ester. Efficient incremental constrained clustering. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)*, pages 240–249, 2007.

[16] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1:224–227, 1979.

[17] R. Desimone and J. Duncan. Neural mechanisms of selective visual attention. *Annual Review of Neuroscience*, 18:193–222, 1995.

[18] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics and Systems*, 3(3):32–57, 1973.

[19] G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1216–1223, 2007.

[20] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 3rd SIAM International Conference on Data Mining (SDM03)*, pages 47–58, 2003.

[21] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD '96)*, pages 226–231, 1996.

[22] Yu Feng and Greg Hamerly. Pg-means: learning the number of clusters in data. In *Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS '06)*, pages 393–400, 2006.

[23] Xiaoli Z. Fern and Wei Lin. Cluster ensemble selection. In *Proceedings of the 2008 SIAM International Conference on Data Mining (SDM08)*, pages 787–797, 2008.

[24] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

[25] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, pages 341–352, 2005.

[26] David Gondek and Thomas Hofmann. Conditional information bottleneck clustering. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), Workshop on Clustering Large Data Sets*, pages 36–42, 2003.

[27] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD '98)*, pages 73–84, 1998.

[28] Dirk Habich, Thomas Wächter, Wolfgang Lehner, and Christian Pilarsky. Two-phase clustering strategy for gene expression data sets. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 06)*, pages 145–150, 2006.

[29] Martin Hahmann, Markus Dumat, Dirk Habich, and Wolfgang Lehner. Explorative multi-view clustering using frequent-groupings. In *Proceedings of the 3rd MultiClust Workshop: Discovering, Summarizing and Using Multiple Clusterings (MultiClust 2012)*, 2012.

[30] Martin Hahmann, Dirk Habich, and Wolfgang Lehner. Evolving ensemble-clustering to a feedback-driven process. In *Proceedings of the IEEE ICDM Workshop on Visual Analytics and Knowledge Discovery (VAKD '10)*, pages 401–408, 2010.

[31] Martin Hahmann, Dirk Habich, and Wolfgang Lehner. Visual decision support for ensemble-clustering. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management (SSDBM '10)*, pages 279–287, 2010.

[32] Martin Hahmann, Dirk Habich, and Wolfgang Lehner. Browsing robust clustering-alternatives. In *Proceedings of the 2nd MultiClust Workshop: Discovering, Summarizing and Using Multiple Clusterings (MultiClust 2011)*, 2011.

[33] Martin Hahmann, Dirk Habich, and Wolfgang Lehner. Touch it, mine it, view it, shape it. In *Proceedings der 14. GI-Fachtagung für Datenbanksysteme in Business, Technology und Web (BTW 2011)*, pages 746–749, 2011.

[34] Martin Hahmann, Peter Volk, Frank Rosenthal, Dirk Habich, and Wolfgang Lehner. How to control clustering results? flexible clustering aggregation. In *Advances in Intelligent Data Analysis VIII*, pages 59–70, 2009.

[35] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, December 2001.

[36] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Proceedings of the 17th Annual Conference on Neural Information Processing Systems (NIPS '03)*, pages 281–288, 2003.

[37] Alexander Hinneburg and Daniel A. Keim. A general approach to clustering in large databases with noise. *Knowledge and Information Systems*, 5(4):387–415, 2003.

[38] P. Hore, L. Hall, and D. Goldgof. A cluster ensemble framework for large data sets. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC '06)*, volume 4, pages 3342–3347, 2006.

[39] Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2):69–91, 1985.

[40] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[41] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In *Proceedings of the 34th Design Automation Conference (DAC '97)*, pages 526–529, 1997.

[42] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[43] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[44] George Karypis and Vipin Kumar. Metis: Unstructured graph partitioning and sparse matrix ordering system, version 4.0. Technical report, University of Minnesota, Minneapolis, MN, 2009.

[45] Leonard Kaufman and Peter J. Rousseeuw. *Clustering by Means of Medoids*. Reports of the Faculty of Mathematics and Informatics. Delft University of Technology. Fac., Univ., 1987.

[46] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 9th edition, March 1990.

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

[48] Donald E. Knuth. Dancing links. In *Millenial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposion in Honour of Sir Tony Hoare*, pages 187–214. Palgrave, 2000.

[49] Hans-Peter Kriegel, Peer Kroger, Matthias Renz, and Sebastian Wurst. A generic framework for efficient subspace clustering of high-dimensional data. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pages 250–257, 2005.

[50] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[51] Bo Long, Zhongfei (Mark) Zhang, and Philip S. Yu. Combining multiple clusterings by soft correspondence. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pages 282–289, 2005.

[52] Shuai Ma, Tengjiao Wang, Shiwei Tang, Dongqing Yang, and Jun Gao. A new fast clustering algorithm based on reference and density. In *Advances in Web-Age Information Management*, pages 214–225, 2003.

[53] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[54] Emmanuel Müller, Ira Assent, Ralph Krieger, Stephan Günnemann, and Thomas Seidl. Densest: Density estimation for data mining in high dimensional spaces. In *Proceedings of the 2009 SIAM International Conference on Data Mining (SDM09)*, pages 173–184, 2009.

[55] Raymond T. Ng, Jiawei Han, and Ieee Computer Society. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1017, 2005.

[56] Feng Pan, Gao Cong, Anthony K. H. Tung, Jiong Yang, and Mohammed Javeed Zaki. Carpenter: Finding closed patterns in long biological datasets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003)*, pages 637–642, 2003.

[57] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 727–734, 2000.

[58] W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[59] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of 24th International Conference on Very Large Data Bases (VLDB '98)*, pages 428–439, 1998.

[60] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages (VL '96)*, pages 336–343, 1996.

[61] H. Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci.*, C1. III vol IV:801–804, 1956.

[62] Alexander Strehl and Joydeep Ghosh. Cluster ensembles a knowledge reuse framework for combining partitionings. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 02)*, pages 93–98, 2002.

[63] Andrada Tatu, Fabian MaaSS, Ines Fäber, Enrico Bertini, Tobias Schreck, Thomas Seidl, and Daniel A. Keim. Subspace search and visualization to make sense of alternative clusterings in high-dimensional data. In *Procedings of IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 63–72, 2012.

[64] Soujanya Vadapalli and Kamalakar Karlapalem. Heidi matrix: nearest neighbor driven high dimensional data visualization. In *Proceedings of the ACM SIGKDD Workshop on Visual Analytics and Knowledge Discovery: Integrating Automated Analysis with Interactive Exploration (VAKD '09)*, pages 83–92, 2009.

[65] Wei Wang, Jiong Yang, and Richard R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '97)*, pages 186–195, 1997.

[66] Xiaoru Yuan, Peihong Guo, He Xiao, Hong Zhou, and Huamin Qu. Scattering points in parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1001–1008, 2009.

[67] Mohammed J. Zaki and Ching-Jiu Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2nd SIAM International Conference on Data Mining (SDM02)*, pages 457–473, 2002.

[68] Yujing Zeng, Jianshan Tang, Javier Garcia-Frias, and Guang R. Gao. An adaptive meta-clustering approach: Combining the information from different clustering results. In *Proceedings of the 1st IEEE Computer Society Conference on Bioinformatics (CSB 2002)*, pages 276–287, 2002.

CONFIRMATION

I confirm that I independently prepared the thesis and that I used only the references and auxiliary means indicated in the thesis.

Dresden, February 27, 2014