Cluster Flow — an Advanced Concept for Ensemble-Enabling, Interactive Clustering

Sandra Obermeier[1], Anna Beer[1], Florian Wahl[1], Thomas Seidl[1]

Abstract: Even though most clustering algorithms serve knowledge discovery in fields other than computer science, most of them still require users to be familiar with programming or data mining to some extent. As that often prevents efficient research, we developed an easy to use, highly explainable clustering method accompanied by an interactive tool for clustering. It is based on intuitively understandable kNN graphs and the subsequent application of adaptable filters, which can be combined ensemble-like and iteratively and prune unnecessary or misleading edges. For a first overview of the data, fully automatic predefined filter cascades deliver robust results. A selection of simple filters and combination methods that can be chosen interactively yield very good results on benchmark datasets compared to various algorithms.

Keywords: Clustering; Interactive; kNN; Ensemble; Explainability

1 Introduction

Researchers in virtually all areas can benefit from clustering their data at some point. From natural sciences over social studies to economics — data is gathered everywhere. Clustering provides many advantages: while the main goal is to find groups of similar objects, it can also help gather valuable hidden information from the data or identify essential attributes. While researchers are experts in their field, they often do not have sufficient background knowledge about clustering methods and, for the sake of simplicity, use old traditional algorithms that may not even fit their data.

As datasets from different research areas contain different types of clusters, ensemble methods proved themselves as suitable for users without profound knowledge in data science. Nevertheless, ensemble methods can be even less understandable "black boxes" than only one algorithm, as they combine different clustering algorithms. Interactive and visual approaches offer great possibilities to make these black boxes more accessible and embody a good solution for the desired balance: To make the powerful tool of clustering accessible to researchers from all fields so that they can create the most meaningful and transparent clusterings with as little effort and background knowledge as possible.

[1] LMU Munich, Institut für Informatik, Oettingenstr. 67, 80538 München, Germany, [obermeier,beer,seidl]@dbs.ifi.lmu.de

doi:10.18420/btw2021-09

A high explainability is thus equally important as public availability. Due to the lack of those in current methods, surprisingly, many researchers still group their data manually, which is disastrous regarding the reproducibility of results and the whole research's objectivity.

Especially complex clustering methods such as ensemble methods are hard to visualize, and accordingly, it is tough to create interaction possibilities on intermediate levels that allow the user to intervene, guide and better understand the process.

To solve these problems, we developed a concept with a prototypical implementation called Cluster Flow. It combines kNN-based approaches for clustering on graphs with a modular and easy to understand architecture. It is simple but simultaneously provides enough flexibility to accomplish difficult clustering tasks. We publish our code at `https://github.com/sobermeier/cluster-flow`. Cluster Flow combines the advantages of ensemble clustering with interactive clustering: users of all areas can easily apply and compose various intuitively understandable cluster improvement steps iteratively to explore and cluster their data. Our method, which we describe in detail in Section 3, is based on kNN graphs as they represent one of the best foundations for clustering and offer several advantages: they are highly explainable, suitable for anytime changes, and interactive approaches, and they can enable finding non-convex shaped clusters as well as clusters of different densities. We developed multiple intuitively understandable filter methods partially based on existing methods to prune edges connecting clusters. They can be combined sequentially or in parallel (ensemble-like). Users can fine-tune parameters, change filters, and explore the dataset at any time, guided by a well-structured prototypical user interface as explained in Section 4. Extensive experiments in Section 5 show that our concept, applying pruning-strategies on kNN graphs, achieves better clusterings than several other algorithms with their best parameter settings. Simultaneously, the method is robust and suitable for exploring data: with fixed parameters for all tested datasets, our fully automatic predefined filter cascades yield better results than comparative methods. Our main contributions can be summarized as follows.

- We propose Cluster Flow, an advanced clustering concept based on kNN graphs by deleting edges through filters.
- Due to its well-thought-out modular design, interactions can be easily integrated on intermediate stages while also providing step-by-step visualizations of the intermediate clustering results.
- Even beyond this, we provide predefined filter cascades that achieve competitive results fully automatically.
- A prototypical implementation serves as a proof-of-concept and demonstrates the power of our proposed approach.

2 Related Work

We address here the three main components that comprise our Cluster Flow concept. As most filter methods are based on diverse graphs describing the data, and we need to evaluate our clustering results objectively, we introduce some graph-based methods in Subsection 2.1, which we also use as a baseline for our experiments in Section 5. Combining the filters can be seen as an ensemble approach; thus, we introduce the most relevant ensemble-based clustering methods in Subsection 2.2. Subsection 2.3 concludes this section by setting our concept into context regarding existing interactive approaches.

2.1 Graph-Based Clustering

Clustering algorithms can rely on different graphs extracted from the original data, e.g., ε-range graphs or diverse variants of kNN graphs, where we focus on the latter. Some approaches rely on a mutual kNN graph (MkNN, see Section 3.1). Existing works include taking the plain MkNN graph, where a connected component with two or more points form a cluster and otherwise are considered outliers [Br97], or slightly advanced ones where a weighted MkNN graph is used to capture clique-like structures [SB14]. Choosing the optimal value for k is especially difficult for MkNN, which are inherently sparser than, e.g., symmetric kNN graphs (see Section 3.1).

The hierarchical clustering algorithm *CHAMELEON* [KHK99] is based on symmetric kNN graphs and consists of two phases. First, the kNN graph is partitioned into small sub-clusters by repeatedly splitting the currently largest sub-cluster, such that the edge cut is minimized until the largest sub-cluster contains fewer nodes than a user-given parameter *MinSize*. Secondly, these sub-clusters are recombined using an agglomerative hierarchical clustering algorithm concerning their relative inter-connectivity and closeness. The merging algorithm terminates when only one cluster remains, or no pair of clusters satisfies the condition of having high enough relative inter-connectivity and relative closeness.

Girvan-Newman Algorithm [GN02] is an approach for detecting community structures in graphs. The authors introduce a measure called *edge betweenness* which corresponds to the number of shortest paths that run along this edge. All shortest paths between nodes of different communities go along at least one edge that connects the communities. Thus, the *edge betweenness* score of such an inter-community edge is higher. The algorithm iteratively removes the edge with the highest *edge betweenness* and recalculates it for the remaining edges until there are no more edges in the graph. The result of the algorithm is a dendrogram that reveals the community structure of the underlying graph. However, this method has a relatively high runtime with $O(m^2 \cdot n)$ for graphs with m edges and n nodes.

Spectral Clustering [SM00] is based on a similarity graph and its weighted adjacency matrix, the first k eigenvectors are calculated. A kNN graph can be used as a similarity graph with distances between points as weights. Clustering is then performed with k-means

on the matrix's rows, which contains the previously calculated eigenvectors as columns. The resulting clustering assignment of k-means corresponds to the clustering alignment of the original points. Spectral clustering also inherits the disadvantages from the additional partitioning step, such as k-means to a certain degree.

2.2 Ensemble based Clustering

Our approach is closely related to ensemble clustering methods since our filter strategy allows combinations to find a consensus. Cluster ensembles, sometimes also referred to as clustering aggregations or consensus clustering, combine several cluster algorithms to obtain a single result of better quality than each cluster individually. Usually, they are based on two steps, namely the *generation*, where different partitions are obtained, and the *consensus*, where these partitions are integrated into one final partition. Ensemble methods should at least meet the following four criteria [VPRS11]:

- Robustness: The average performance must be better than the single clustering algorithms.
- Consistency: The combined result should be very similar to all combined single clustering algorithm results.
- Novelty: The ensemble must allow finding solutions unattainable by single clustering algorithms.
- Stability: The results must be less sensitive to noise and outliers.

However, according to the same authors, identifying the best result is hard, but the general idea behind ensemble methods is that several algorithms' combined decisions should be more reliable than any individual one. Several existing works focus on the clustering techniques [Li15, FJ05, Wu13] while others focus on finding the right consensus [SG02], and allow for using different clustering methods.

However, in these approaches, determining the consensus functions is only applicable for experts and integrating different techniques is even more complex and challenging to understand to scientists from other domains. In contrast to that, our edge-deletion concept allows for a smooth integration of establishing consensus across filters while at the same time the intermediate results are always visible and understandable. Our approach differs from existing approaches since it is possible to apply the ensemble clustering paradigm as an intermediate step. The user can access the result at a fine-granular level and directly see how different filters agree upon activities to identify crucial spots or steer the end product into a more conservative or progressive direction. These advantages come because our approach combines the power of ensemble clustering and interactive clustering. We discuss the latter in the next section.

2.3 Interactive Clustering

The overall goal of interactive clustering is to engage the user as far as possible in the clustering process, not only to allow the user to make the result fit their preferences but also to make it understandable. In our context, we utterly differentiate from methods that solely enable visual exploration of the result and are only considering methods that allow interaction within the algorithmic loop. [Ba20] provide a thorough survey of interactive clustering. Over 100 papers related to interactive clustering are analyzed regarding the stage and type of interaction, user feedback, evaluation criteria, data, and clustering methods. The authors distinguish three groups of stages in which interaction occurs. (1) Interaction on clustering results, (2) interaction on model/algorithm level, and (3) machine-initiated interaction. Our concept belongs to the second group since the user's interactions directly happen at the algorithm level by tweaking parameters rather than at the clustering results. We evaluate the clustering result on an objective basis rather than conducting a user study for subjective evaluation. A user study might help develop an appealing and intuitive graphical user interface but is not within this work scope. Also, visualization methods for supporting interactive ensemble clustering like AUGUR [HHL10] could be incorporated for future work.

To conclude this section and put our work into the context of related works, our concept combines three main components that fit together enormously well. First, kNN-graphs are inherently well understandable for humans. Second, based on the kNN-graph, filter strategies are applied. The filters focus on different hidden structures in the data, but all result in the same action, namely deleting edges. Therefore, finding a consensus among them in an ensemble-like manner does not require complex mathematical functions but rather comparison on edge level and can thus be well visualized and understood. Third, the modular design allows interaction on each stage and the continuous visualization of intermediate results.

3 Cluster Flow

Cluster Flow works on a kNN graph of the input, for which we present multiple options in Section 3.1. Elaborated filters, which we introduce in Section 3.2, delete edges between clusters, and thus the graph decomposes into several smaller graphs representing one cluster each.

3.1 Build kNN Graphs

Cluster Flow allows to choose between different variants of the kNN graph, as they can have a severe impact on the clustering result [MLH09]: basic kNN graphs, symmetric kNN graphs, and mutual kNN graphs.

Within this paper, we use the following notation: Let U be an arbitrary set and $DB \subseteq U, |U| < \infty$ be a dataset. A **kNN** graph consists of nodes corresponding to the points of a dataset and directed edges from every node to its k nearest neighbors (kNN). The kNN graph is a directed graph where an edge (p_i, p_j) from point p_i to point p_j exists if and only if p_j belongs to the k-neighborhood of p_i [HKF04]:

$$edges = \{(p_i, p_j) \mid \forall p_i, p_j \in \text{DB}: i \neq j \wedge p_j \in \text{kNN}(p_i)\} \quad (1)$$

A **symmetric kNN** graph has a higher connectivity than the kNN graph: it is an undirected graph where an edge (p_i, p_j) from point p_i to point p_j exists if p_j is part of k-neighborhood of p_i or vice versa [HKF04, MHVL07]:

$$edges = \{(p_i, p_j) \mid \forall p_i, p_j \in \text{DB}: i \neq j \wedge (p_j \in \text{kNN}(p_i) \vee p_i \in \text{kNN}(p_j)\}. \quad (2)$$

A **Mutual k-Nearest Neighbor (MkNN)** graph is an undirected graph, where edges exist between two points p_i and p_j if both points belong to each other's k-neighborhood [Br97, HKF04]:

$$edges = \{(p_i, p_j) \mid \forall p_i, p_j \in \text{DB}: i \neq j \wedge p_i \in \text{kNN}(p_j) \wedge p_j \in \text{kNN}(p_i)\}. \quad (3)$$

The **RkNN** graph connects points to their reverse nearest neighbors, i.e., its adjacency matrix is the transpose of the adjacency matrix of the corresponding kNN graph. A point p is a reverse nearest neighbor of a point q, iff q is a nearest neighbor of p.

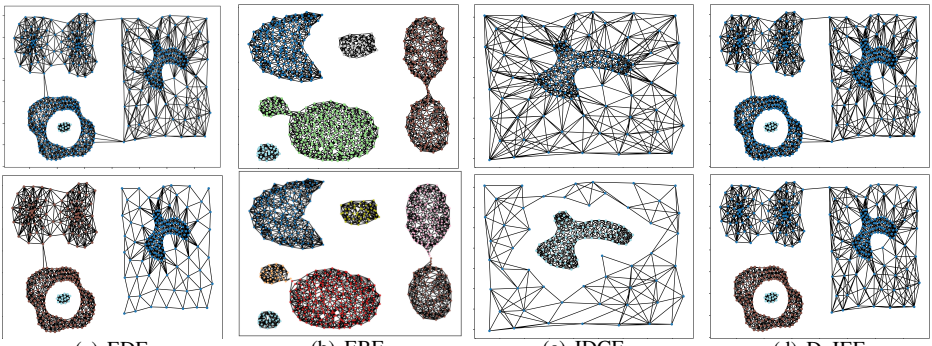
As we later only delete edges and never add edges, points of a cluster must have a connection in the graph. Thus, a symmetric kNN graph with its inherent rather high connectivity is usually suitable for our approach. If users are interested in the most significant clusters or the core points of a cluster, an MkNN graph can be a good choice [MHVL07]. Note that k for MkNN graphs should be higher than for symmetric kNN graphs to ensure a certain degree of connectivity. Even though we use unweighted graphs for all filters, we save and reuse the distances calculated in this step if needed.

3.2 Filters

In the following, we introduce filters that can be applied to the kNN graph. Filters are easily accessible for users of diverse domains and delete different edges depending on the graph's properties, as we illustrate with selected example graphs shown in Figure 1.

3.2.1 Edge-Distance Filter (EDF)

Since points that are close to each other and connected by a short edge in the kNN graph are likely to be in the same cluster, EDF deletes edges longer than a threshold t,



(a) EDF (b) EBF (c) IDCF (d) DoIEF

Fig. 1: Application of the Filters. Top: Symmetric kNN graph with k=10 of the original datasets. Bottom: After filter application. Each connected component, i.e. cluster, is indicated by a different color.

granting certain reachability for the clusters. The filter considers each connected component individually, which has two advantages: (1) We can run the filter in parallel on several connected components to save computation time, and (2) t is not global but can be chosen for each connected component individually, depending on its edges. This enables maintaining connected components with different densities, which will later result in clusters. The threshold t depends on the mean μ and standard deviation σ of the edge distances in a connected component, on which the filter is applied, where σ is weighted by a parameter $p \in \mathbf{R}$: $t = \mu + p \cdot \sigma$. An empirically good value for p is between 1 and 3. Figure 1(a) shows an example application of this filter on the Compound dataset. The top image displays the symmetric kNN graph with k=10 on the original dataset, which is the input for this filter. The bottom figure displays the result after applying EDF with p=2 where several unwanted edges have been removed correctly. The filter runs with a complexity of $O(m)$ where m is the number of edges. Note that the distances between all points can be reused from the kNN graph generation.

3.2.2 Edge-Betweenness Filter (EBF)

This filter is based on the Girvan-Newman algorithm and uses the edge betweenness measure to identify and reduce inter-community connections. It works on the assumption that loosely connected components belong to different clusters. This filter iteratively removes the edges with the highest edge betweenness, where i is the number of iterations and p is the number of edges to delete. As smaller clusters have fewer paths connecting all nodes than larger clusters, misclassification, i.e., deleting wrong edges, has a higher impact on them. To overcome this, we make our filter more restrictive, i.e., we scale the number of removed

edges per iteration by the total number of edges $|edges|$ within a connected component. We also constrain that in each iteration, a minimum of one edge is deleted. Parameter r is then defined by: $r = max(|edges| \cdot p, 1)$.

Figure 1(b) shows an example application of the edge betweenness filter on the Aggregation dataset. The input graph on top is generated with $k = 10$. We set the filter parameters as follows: $i = 5$; i.e., five iterations were performed, and $p = 0.0075$; i.e., 0.75% of the edges were removed from each connected component in each round. The bottom figure shows the result after all iterations. The inter-community connections have been detected correctly, and the initial five connected components have been divided into seven, which fit the ground truth clusters and are highlighted by different colors.

The original Girvan-Newman algorithm has a worst-case complexity of $O(m^2 \cdot n)$, where n is the number of nodes and m is the number of edges. In our modified setting, we can decrease this complexity. First, instead of performing a full hierarchical clustering down to every node, we only run the algorithm for i iterations to identify the top inter-community connections, where $i << m$. Secondly, originally only one edge is removed in each iteration. In our case, we increase the number of edges that are deleted in each round to the value of a parameter r. This way, the user can decide how precise the final result should be. Since the complexity for one round of the original algorithm is $O(m \cdot n)$, the filter's application with i rounds has the complexity of $O(m \cdot n \cdot i)$. It is advisable not to apply the filter in the beginning but rather when the complete dataset is already segmented in several connected components to reduce the complexity (For example, by first applying a filter with lower complexity, e.g., the *EDF*).

3.2.3 Inter-Density Connection Filter (IDCF)

This filter assumes that two points located in regions of different densities also belong to two different clusters. The sparseness of the neighborhood of a point is defined by the average distance to its k-nearest-neighbors, which equals to the sparseness estimation presented in [SRS00] and is used for outlier detection. We call an edge between two nodes with very different dense neighborhoods **inter-density connection**. The inter-density connection filter aims at detecting these edges to classify them as unwanted. The **density difference of an edge** is defined as the absolute difference of the sparseness of the two nodes it connects. If this density difference is higher than a threshold t, the edge is classified as unwanted. Given μ as the average of the density difference of all edges, σ as its standard derivation and p as a user-defined parameter to regulate the sensitivity of the filter, the threshold t is defined as: $t = \mu + p \cdot \sigma$. Again, the filter is applied to each connected component separately. The sparseness estimation is made on the basis of the directed kNN graph, but only the edges which also exist in the current state of the undirected graph will be considered so that each point has 0 to k considered nearest neighbors. We need the directed graph for this filter, since edges to outliers, which more likely exist in an undirected graph where

the outlier only needs to belong either to the kNN *or* the reverse kNN set, may lead to misclassification. A connected outlier forces an increase in the sparseness estimation for the whole connected component and may lead to the deletion of edges between actual cluster nodes. Nevertheless, we do not have to recompute all kNN relationships since we have created and saved a directed kNN graph at the graph generation step that contains all kNN relationships and distances. Figure 1(c) shows an example application of the inter-density connection filter on a sample from the Compound dataset. On the top, the original input graph is depicted (symmetric kNN graph with $k = 10$). On the bottom, the resulting graph after applying the filter two times with $p = 1.5$ is shown. The filter can separate the inner, more dense cluster from the outer data points.

3.2.4 Distance of Incoming Edges Filter (DoIEF)

Similar to the *EDF*, this filter considers the length of edges. However, unlike the aforementioned, this filter focuses on each node separately and uses the directed kNN graph. The filter considers all incoming edges of a node and therefore requires a calculation of the reverse kNN relationships based on the directed kNN graph in advance. We classify the incoming edges of a node, i.e., edges connecting nodes of the reverse kNN set of the node, as unwanted if their length exceeds a threshold t. The threshold t is defined as: $t = \mu + p \cdot \sigma$, where μ is the average edge length, and σ is the standard derivation of the edge lengths. This filter is used for separating outliers or boundary nodes from other connected components. The intuition behind this filter is that nodes with long edges to their k-nearest neighbor are likely to be outliers or at least probably not part of the cluster to which that neighbor is currently connected. We use the incoming edges for this filter because this is more restrictive than using the outgoing edges. The RkNN and the kNN relationship are not symmetric, i.e., every node has the same number of outgoing edges, but not every node has incoming edges. If we took the outgoing edges, every node in the whole graph would be considered, including those that do not belong to the kNN set of any other node. However, by taking the incoming edges, we limit the considered nodes to those that are part of the kNN set of at least one other node and thus decrease the computational costs and the probability of deleting wanted edges. Figure 1(d) shows a symmetric kNN graph as input on the top, which was generated with $k = 10$ on the Compound dataset. The result after the first application of the *DoIEF* with $p = 2$ is shown on the bottom. The illustration shows that the *DoIEF* deletes edges that connect boundary points of different clusters.

3.3 Combination of Filter Results

We present two different filter strategies: using filters *sequentially* and using filters in *parallel*. The former applies filters consecutively, using results from the previous filter as input for the next filter. Filters can be applied multiple times, and different types of filters can be concatenated. As filters are applied on each connected component separately, an

iterative application of the same filter often leads to better results than a one-time application with relatively high respectively low threshold values. In the beginning, there might be only one component connecting all points. Iterative filters can consider different cluster structures, and the individual clusters can develop selectively. The second strategy embodies the classical cluster ensemble approach's central idea, where different clustering algorithms are performed on the same input, and afterward, a consensus between the different results is determined. We adopt this idea for our concept by applying different filters in parallel on the same input graph. Each filter determines independently which edges should be deleted. Afterward, a common consensus is determined, which can be chosen conservatively, e.g., all filters must agree to the deletion of an edge to ensure a safe deletion within the final result, or progressively, e.g., only 50% of filters need to propose the deletion of an edge to result in a deletion within the final result. Cluster Flow is designed so that the two strategies can be easily combined.

3.4 Concept Overview and Discussion

Figure 2 abstracts a possible manifestation of the Cluster Flow architecture. A gray box represents an atomic building block (either the initial graph generation step, one of the proposed filters, or a consensus component). Important is the graph construction at the beginning, which is the basis for the further procedure. After this mandatory first step, the user can apply any filter presented in the previous paragraphs. Interaction possibilities to tweak parameters and visualizations of intermediate results are integrated at each building block and enable maintaining the overview at all times. The red, solid framed box indicates a sequential filter chain, while the blue dotted framed box shows a parallel filter strategy. A filter always works on the result of the previous building block. Each filter in the parallel component works on the input from the previous filter independently. Finding a consensus is also an independent building block that determines how many parallel filters need to agree upon deleting an edge to delete an edge ultimately. The output of the consensus-building block results in a subsequent filter's input, if one is applied, or in this case, as the final result. We also want to emphasize that the modular design allows storing intermediate results that do not require recalculation whenever a subsequent filter is updated. With this, it is possible to try out different parameters without starting from the bottom. The user experience greatly benefits from this architecture. The experiments were not runtime optimized, but a single filter's execution is in the millisecond to second range for the tested datasets. A significant advantage of our concept is its great flexibility, but in some sense, it might also be challenging to find a good way to start. We recommend relying on a filter-refinement-like strategy to start with a fast, non-exact deletion of edges and go over to more costly fine-tuning. The *EDF* is the most simple filter in our repertoire and is also runtime-efficient. Though it does not delete edges between, e.g., clusters of different densities like *IDCF*, it is a perfect start to delete many unnecessary edges without complex computations, allowing additional filters to work on a fraction of the original edges. The *EBF* is the most complicated filter, which is well suited to be applied at the end as a refinement step. The combination of filters is

a good strategy for either the start or the end, depending on the goal of the clustering. If a conservative approach is generally preferred, the combination of filters can be applied initially, which, depending on the consensus function, deletes only obviously unwanted edges to prevent premature deletion of edges. However, the combination is also suitable to serve as a refinement step.

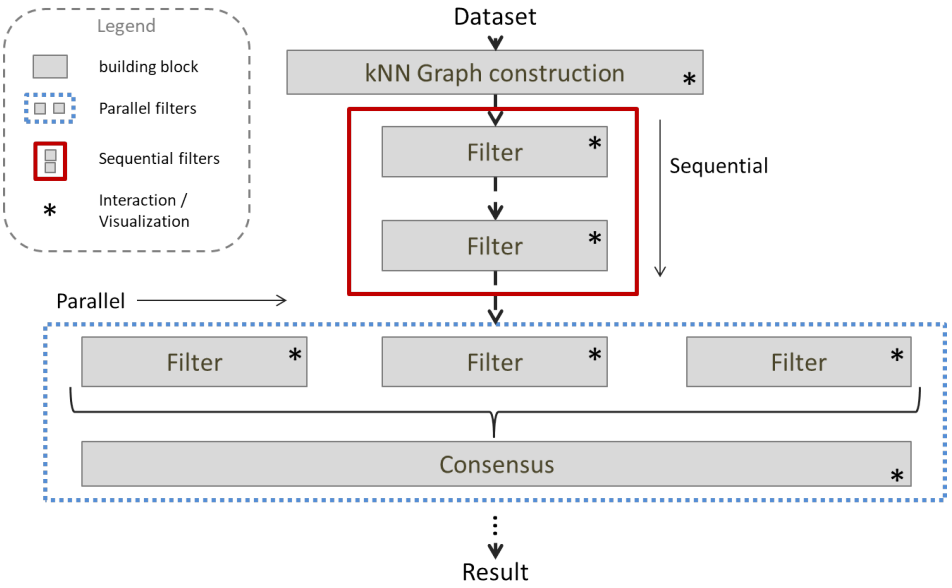


Fig. 2: Design concept of Cluster Flow.

4 Interactive Clustering with Cluster Flow

Due to the modular character and the step-by-step application of the filters, Cluster Flow is well suited for interactive clustering. As a proof of concept, we implemented a lightweight tool that provides a simple interface for loading datasets in CSV format, creating and saving individual projects. Within each project, it is possible to add filters, either at each step for the sequential case or several filters included in one step for the parallel case. One step is represented by a tile containing one or more inner cards, which offer a visualization of the current result on the left and configuration options on the right. For simplicity, each card offers the option to visualize the current result graph within a 2D view, a 3D view, or to apply PCA decomposition [Pe01] for dimensionality reduction. The first step within a project is always the kNN graph generation. Here users can choose the graph type, i.e., symmetric or mutual, and the value for k. Users can attach filters to form a chain in which each filter is executed one after another. Each filter step can be executed and adjusted

Tab. 1: Evaluation datasets with number of clusters c and number of dimensions d.

Set 1 (Gaussian)				Set 2 (Non-Convex)				Set 3 (Mixture)			
Name	c	d	Size	Name	c	d	Size	Name	c	d	Size
Cassini[3]	3	2	1000	Two Moons[3]	2	2	300	Aggregation[GMT07][4]	7	2	788
Cuboids[3]	4	3	1002	Donutcurves[3]	4	2	1000	Compound[Za71][4]	6	2	399
Hypercube[3]	8	3	800	Long2[3]	2	2	1000	Pathbased[CY08][4]	3	2	300
Cure-t0-2000n-2D[3]	3	2	2000	Dartboard1[3]	4	2	1000	Lsun[3]	3	2	400
Pmf[3]	5	3	649	Donut3[3]	3	2	999	Spiralsquare[3]	6	2	1500
Twenty[3]	20	2	1000	Smile2[3]	4	2	1000	Longsquare[3]	6	2	900
Twodiamonds[3]	2	2	800	Zelnik1[3]	3	2	299	Dpc[3]	6	2	1000
Spherical_4_3[3]	4	3	400	Zelnik5[3]	4	2	512	Target[3]	6	2	770
Zelnik4[3]	5	2	622	Jain[JL05][4]	2	2	373	R1_complete[5]	4	2	600
R15[VRB02][4]	15	2	600	Spiral[CY08][4]	3	2	312	Mouse[6]	4	2	500

separately, allowing a high degree of transparency and intervention. However, if a previous filter parameter has been changed so that the resulting kNN graph changes, all subsequent filters are recalculated because their input has changed. A traffic-light system indicates the status of computations. The tool is a local web application written in Python, so it is platform-independent. We used Flask[1] as web framework and SQLite[2] for the database. Figure 3 shows an example screenshot of the interactive tool. Our prototype can reuse already computed values when chaining filters as far as possible to prevent the calculation costs from increasing proportionally per added filter and ensure a pleasant, smooth usage. It fulfills all criteria of [VPRS11] explained in Section 2: (1) robustness, (2) consistency, (3) novelty, and (4) stability. (1) Cluster Flow is robust, i.e., the ensemble is on average better than its single components. Especially when looking at complex datasets with diverse types of clusters, the superiority of combining several filters becomes obvious. (2) Since Cluster Flow's consensus strategies are simple operations on sets or majority votes, results are comprehensibly similar to their components' results. (3) The combination of filters produces novel results, which cannot be achieved by a single filter since there are datasets for which one filter cannot delete all necessary edges for correct clustering, even though another one could. E.g., datasets containing clusters of different densities and clusters connected by a chain. Combining both can lead to a perfect result. (4) Using an adequate consensus strategy reduces the sensitivity regarding noise and outliers.

[1] https://palletsprojects.com/p/flask/

[2] https://www.sqlite.org/index.html

[3] `https://github.com/deric/clustering-benchmark/tree/master/src/main/resources/datasets/artificial`

[4] http://cs.joensuu.fi/sipu/datasets/

[5] https://github.com/wahlflo/Datasets

[6] https://elki-project.github.io/datasets/

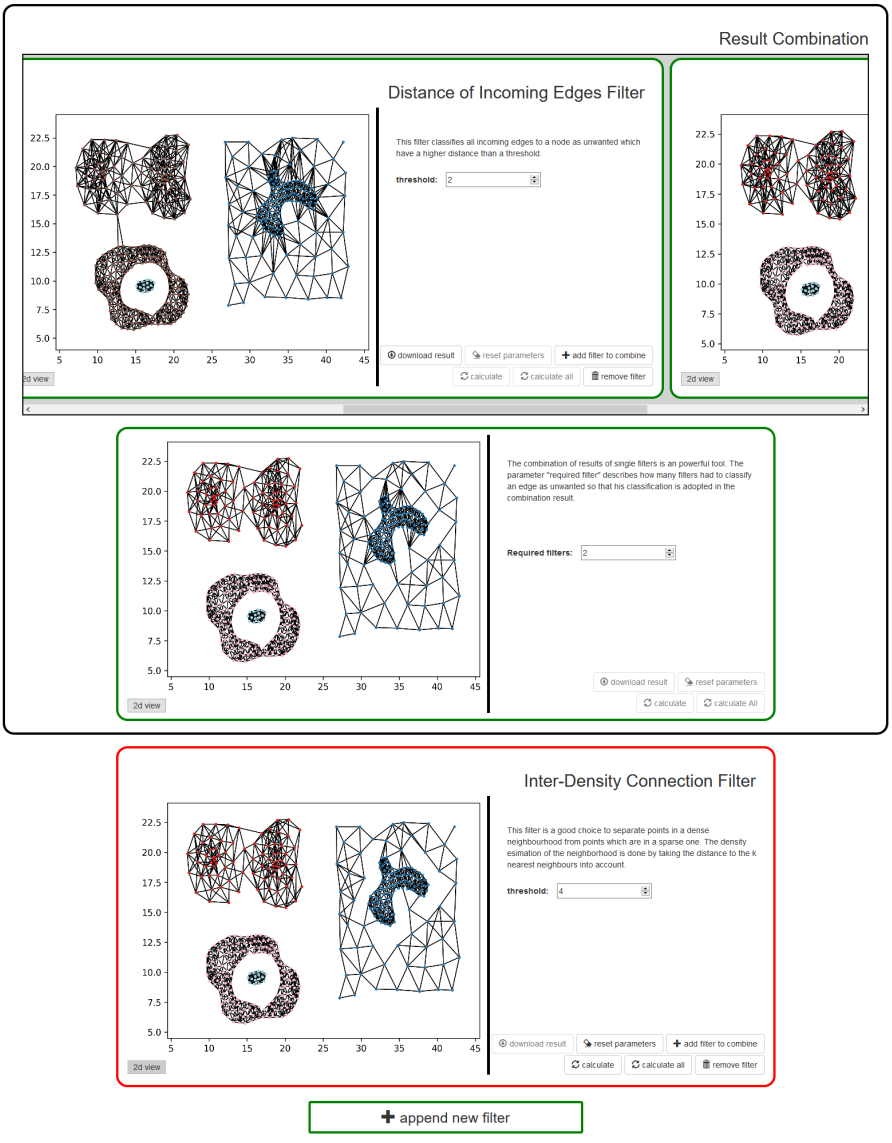


Fig. 3: Screenshot of the prototypical interactive clustering tool. The top row contains parallel filters where the green frame indicates that the calculation has been finished. The bottom row shows a single sequential filter, which is still working.

Tab. 2: Parameter settings used for Cluster Flow.

Parameter	Description	Range	PFC1	PFC2
$graph\_type$	Type of the generated graph.	[symmetric, mutual]	symmetric	symmetric
k	Number of nearest neighbors.	[10, 12, 14]	14	14
EDF_p	Parameter p of the *EDF*.	[1.5, 2, 2.5, 3]	3	2.5
$DoIEF_p$	Parameter p of the *DoIEF*.	[1.5, 2, 2.5, 3]	-	1.5
$IDCF_p$	Parameter p of the *IDCF*.	[2, 3, 4, 5]	-	5
EBF_p	Parameter p of the *EBF*.	[0.0025, 0.005, 0.0075]	0.0075	0.0075
EBF_i	Number of iterations of the *EBF*.	[0, 1, 2, 3, 5, 7, 9]	7	7
$Consensus_n$	Number of filters to agree upon deletion.	[2, 3]	-	2

5 Experiments

5.1 Datasets

We evaluate Cluster Flow on 30 publicly available clustering benchmark datasets as described in Table 1. We grouped them into three groups based on the type of clusters they contain: In the first set, data sets contain Gaussian-like clusters, in the second set, they contain non-convex clusters, and in the third set, they contain a mixture of different types. Additionally, we evaluate on several high dimensional datasets taken from [FS18] and first introduced by [FVH06]. These all have 16 clusters and 1024 points, their dimensionality is $d \in [32, 64, 128, 256, 512]$, and they are called dim032, dim064, dim128, etc.

5.2 Baseline

We want to evaluate our approach on an objective basis. For this, we compare our concept with relevant graph-based methods and other established clustering methods. More precisely, we evaluated using the following methods and performed a grid search on the corresponding parameter settings:

- k-means [Ll82]: $k_{k-means}$: $\{c-2, \ldots, c+2\}$[7]
- CHAMELEON: k: [5, 10, 15], $MinSize$: [2%, 3%], α: [1.5, 2.0, 2.5]
- MkNN clustering: k: $\{3, 4, \ldots, 20\}$
- Rock [BKS19]: $tmax$: [10, 15, 20]
- DBSCAN [Es96]: $MinPoints$: $\{2, 3, \ldots, 15\}$, ϵ: $\{0.01, 0.02, \ldots, 0.4\}$
- Spectral clustering [VL07]: k_{knn}: $\{10, 15\}$, $k_{k-means}$: $\{c-2, \ldots, c+2\}$[7]

[7] c stands for the number of ground truth clusters

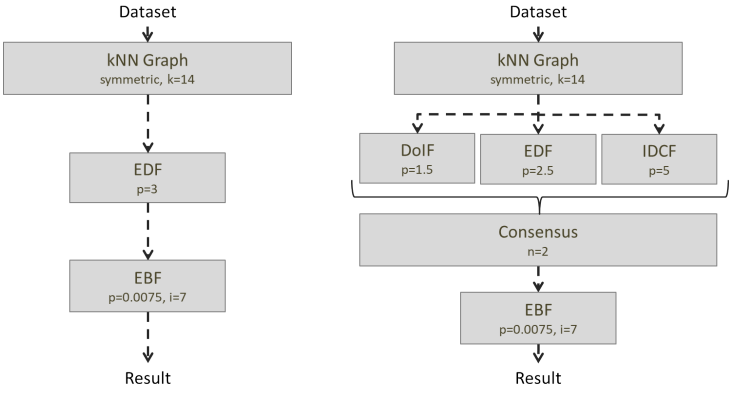
5.3 (Predefined) Filter Cascades

An essential characteristic of Cluster Flow is its high level of flexibility. To ensure better reproducibility and comparability, we here describe filter cascades, which consist of a defined filter composition. The first filter cascade (*FC1*) solely relies on the *sequential* strategy and incorporates the *EDF* and the *EBF*. The *EDF* is applied repeatedly until no edges are newly classified as unwanted. After that, the *EBF* is applied multiple times.

The second filter cascade (*FC2*) relies on a combination of the *sequential* and the *parallel* strategy. The *EDF*, *DoIEF* and the *IDCF* are applied in parallel on the same input dataset. An edge is classified as unwanted if, at minimum, two of the three filters classified it as such. Afterward, the EBF filter is applied multiple times. To make our concept as simple as possible and to obviate time-consuming parameter searches, we also tested both filter cascades with constant, predefined parameters over differently structured data sets, i.e., in a fully automatic setting without user interaction. Table 2 summarizes the tested parameters and their ranges in general as well as the fixed hyper-parameters for the predefined filter cascades (*PFC1*, *PFC2*) that were used in the subsequent analysis. Figure 4 shows the construction of PFC1 and PFC2 for a better understanding. These two filter cascades follow different goals. PFC1 is a more progressive approach that tries to remove many edges directly from the beginning. The *EDF* is a good choice for this, as the focus is solely on the distance between two edges without considering the neighborhood. It is also one of the simplest and fastest filters we propose and thus serves as a good first filter to delete the most obvious edges. The *EBF* is more complex but also more powerful since it can detect bridges between communities. This filter takes more runtime than the other filters, and we recommend applying it towards the end where a refinement is needed. FC1 could also be seen as a filter-refinement procedure, where the *EDF* deletes the most obvious edges fast, and the *EBF* fine-tunes the result. PFC2, in contrast, is more conservatively constructed; that is, in case of doubt, an edge is rather not deleted so as not to cause clusters to decay prematurely. The parallel building block in the beginning only deletes an edge if the majority of the three filters agrees upon it to give a more reliable result. The powerful *EBF* is then used again for fine-tuning the result. In terms of objective evaluation, PFC1 often performs better, but for sensitive applications where the dataset must not be split up in too many clusters too fast, PFC2 is a good option.

5.4 Performance with varying parameters

The left part of Table 3 shows the average F1-scores of all evaluated clustering algorithms for each of the combined sets and for all 30 data sets, whereby we allowed different hyperparameter values for each experiment, to achieve the best possible results at the dataset level. The algorithms are sorted in descending order by their total average performance. To show the importance of the filters, we have evaluated the performance of the kNN graph with different values for k without any filters (CF in the table). FC1, DBSCAN, FC2, and MkNN



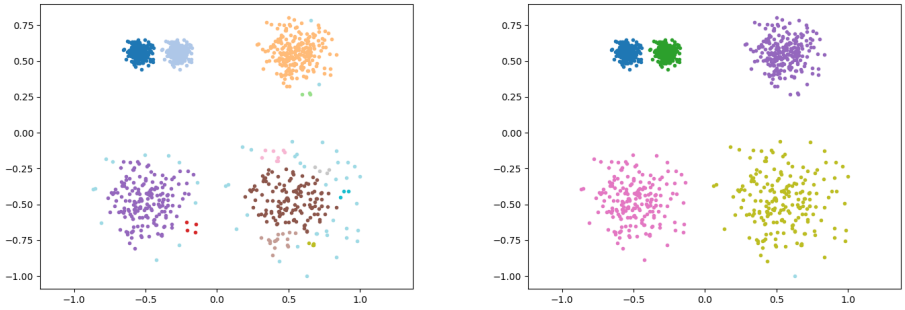
(a) PFC1 (b) PFC2

Fig. 4: Structure of predefined filter cascades 1 (PFC1) and 2 (PFC2).

Tab. 3: AVG F1-Score of Cluster Flow compared to the baseline. Left: Performance of all methods with varying parameters. Right: Performance of the best baseline algorithms, PFC1 and PFC2 with constant parameters.

	Changing Parameters				Constant Parameters	
Algorithm	Set1	Set2	Set3	AVG total	Algorithm	AVG total
FC1	**0.9972**	0.9985	**0.9754**	**0.9902**	PFC1	**0.995**
DBSCAN	0.9967	**0.9999**	0.9716	0.9894	DBSCAN	0.828
FC2	0.9967	0.9938	0.9599	0.9834	PFC2	0.931
MkNN	0.9494	0.9990	0.9029	0.9504	MkNN	0.904
CF (no filters)	0.9224	0.9993	0.7979	0.9065		
CHAMELEON	0.8793	0.8377	0.8401	0.8519		
Spectral	0.9605	0.6857	0.8153	0.8205		
k-means	0.9360	0.6505	0.7407	0.7757		
Rock	0.6935	0.6124	0.7326	0.6795		

based clustering achieved the best results. On *set*2, MkNN performed a little bit better than the kNN clustering approaches. On the other two sets, FC1 and FC2 outperformed the MkNN clustering significantly. In total, FC1 achieved the best results with an average F1-score of 0.990, while DBSCAN came second with 0.989. However, FC1, FC2, and DBSCAN achieved very similar results on all sets apart from small fluctuations. DBSCAN is known to perform well on many of the selected sets. The goal here was to reveal that Cluster Flow consistently delivers better or equally strong results, even on data sets with distributions predestined for DBSCAN or other competitors. However, we also want to explicitly point out situations where our approach significantly outperforms DBSCAN, i.e., identifying clusters with varying densities. Therefore regard Figure 5: (a) shows the best



(a) DBSCAN (b) PFC1

Fig. 5: Qualitative example where PFC1 outperforms DBSCAN as it is able to detect clusters of varying densities.

clustering result of DBSCAN after testing different parameter settings and (b) shows the clustering result of the fully automatically PFC1 on a self created dataset[8] with varying densities.

5.5 Performance while maintaining constant parameters

In the previous analysis, we explicitly determined the optimal parameters for each algorithm and data set individually to achieve the best possible result. However, choosing the right parameters is a laborious and time-consuming task, especially for laymen, since the optimal hyperparameters can vary significantly from dataset to dataset. Thus, we evaluated the performance when using the same parameter settings for all 30 low dimensional data sets. As baseline we used DBSCAN ($\epsilon = 0.08$, $MinPoints = 3$) and MkNN ($k = 10$), as these gave the best results in the upper analysis. The right side of Table 3 summarizes the achieved results of each algorithm constraint to not changing parameters across all 30 benchmark datasets. Here, PFC1 and PFC2 outperformed the other algorithms. These results show the potential of predefined filter cascades in general and that PFC1 and PFC2 are well-suited to obtain a useful clustering without adjusting the parameters, especially without knowing the type or the distribution of the data in advance. Most algorithms only work for specific shapes and distributions of clusters but then fail for other cluster forms. In the real world, however, data distribution is not known in advance, so it is of great importance to offer clustering algorithms that can achieve consistently good results regardless of the distribution and shape of the clusters without having to tweak the hyperparameters. While PFC1 is more progressive in that it deletes all edges considered unwanted, PFC2 offers a more

[8] Dataset *different_density* on `https://github.com/wahlflo/Datasets`

conservative approach, where a certain percentage of filters must share a consensus to force the deletion of edges.

We also performed experiments on the previously described high-dimensional datasets dim032, dim064, etc., which consist of well-separated, randomly sampled Gaussian clusters. In total, the F1-scores of PFC1 (**0.96**, **0.93**, **0.95**, **0.97**, **0.97**) were slightly better or equal to the scores of PF2 (0.95, **0.93**, 0.94, 0.94, 0.93). In general, both showed consistently good results.

6 Conclusion

We developed Cluster Flow, a new advanced concept to cluster data based on kNN graphs. Our approach's key components are modularity, which is also the key for offering intermediate interaction stages, explainability, and simultaneously identifying various cluster shapes. Experiments on more than 30 benchmark datasets show that the proposed technique consistently achieves superior results when used interactively, i.e., varying parameters for different datasets. On top of that, even not seen in an interactive context, the predefined filter cascades PFC1 and PFC2 can be used as entirely autonomous clustering algorithms that work fully automatically and achieved remarkable results over various experiments. Non-convex clusters are found as well as clusters of varying density. The easy to understand concept allows researchers from all areas with no previous knowledge in clustering to explore, cluster, and understand the data in depth. Hence, we have shown an efficient clustering concept that can successfully find diverse clusters and is highly understandable. As the focus of this paper is developing a concept of how to compose easy steps so that laymen can understand what their clustering and results mean, we leave a user study for an even more beautiful visualization and potentially better usability for future work. Additionally, in future work, one could integrate other data types than numerical data and investigate further filter and combination methods. Another goal is to generate branches within the interactive clustering workflow, i.e., to work with several independent intermediate states in parallel or use a change history. To further support the user in the decision process metadata of the nodes or other interesting properties could be displayed. Of course, current acceleration methods like accelerating the kNN graph computation [CLR20] could be integrated, too. For high dimensional data, kNN could be computed according to the subspace importance [Ba04].

Acknowledgments

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

Bibliography

[Ba04] Baumgartner, Christian; Plant, Claudia; Railing, K; Kriegel, H-P; Kroger, Peer: Subspace selection for clustering high-dimensional data. In: Fourth IEEE International Conference on Data Mining (ICDM'04). IEEE, pp. 11–18, 2004.

[Ba20] Bae, Juhee; Helldin, Tove; Riveiro, Maria; Nowaczyk, Sławomir; Bouguelia, Mohamed-Rafik; Falkman, Göran: Interactive Clustering: A Comprehensive Review. ACM Computing Surveys (CSUR), 53(1):1–39, 2020.

[BKS19] Beer, Anna; Kazempour, Daniyal; Seidl, Thomas: Rock - Let the points roam to their clusters themselves. In: Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019. pp. 630–633, 2019.

[Br97] Brito, MR; Chavez, EL; Quiroz, AJ; Yukich, JE: Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. Statistics & Probability Letters, 35(1):33–42, 1997.

[CLR20] Chávez, Edgar; Ludueña, Verónica; Reyes, Nora: Heuristics for Computing k-Nearest Neighbors Graphs. In: Computer Science–CACIC 2019: 25th Argentine Congress of Computer Science, CACIC 2019, Río Cuarto, Argentina, October 14–18, 2019, Revised Selected Papers 25. Springer, pp. 234–249, 2020.

[CY08] Chang, Hong; Yeung, Dit-Yan: Robust path-based spectral clustering. Pattern Recognition, 41(1):191–203, 2008.

[Es96] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd. volume 96, pp. 226–231, 1996.

[FJ05] Fred, Ana LN; Jain, Anil K: Combining multiple clusterings using evidence accumulation. IEEE transactions on pattern analysis and machine intelligence, 27(6):835–850, 2005.

[FS18] Fränti, Pasi; Sieranoja, Sami: K-means properties on six clustering benchmark datasets. Applied Intelligence, 48(12):4743–4759, 2018.

[FVH06] Fränti, Pasi; Virmajoki, Olli; Hautamaki, Ville: Fast agglomerative clustering using a k-nearest neighbor graph. IEEE transactions on pattern analysis and machine intelligence, 28(11):1875–1881, 2006.

[GMT07] Gionis, Aristides; Mannila, Heikki; Tsaparas, Panayiotis: Clustering aggregation. ACM Transactions on Knowledge Discovery from Data (TKDD), 1(1):4–es, 2007.

[GN02] Girvan, Michelle; Newman, Mark EJ: Community structure in social and biological networks. Proceedings of the national academy of sciences, 99(12):7821–7826, 2002.

[HHL10] Hahmann, Martin; Habich, Dirk; Lehner, Wolfgang: Visual decision support for ensemble clustering. In: International Conference on Scientific and Statistical Database Management. Springer, pp. 279–287, 2010.

[HKF04] Hautamaki, Ville; Karkkainen, Ismo; Fränti, Pasi: Outlier detection using k-nearest neighbour graph. In: Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. volume 3. IEEE, pp. 430–433, 2004.

[JL05] Jain, Anil K; Law, Martin HC: Data clustering: A user's dilemma. In: International conference on pattern recognition and machine intelligence. Springer, pp. 1–10, 2005.

[KHK99] Karypis, George; Han, Eui-Hong; Kumar, Vipin: Chameleon: Hierarchical clustering using dynamic modeling. Computer, 32(8):68–75, 1999.

[Li15] Liu, Hongfu; Liu, Tongliang; Wu, Junjie; Tao, Dacheng; Fu, Yun: Spectral ensemble clustering. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. pp. 715–724, 2015.

[Ll82] Lloyd, Stuart: Least squares quantization in PCM. IEEE transactions on information theory, 28(2):129–137, 1982.

[MHVL07] Maier, Markus; Hein, Matthias; Von Luxburg, Ulrike: Cluster identification in nearest-neighbor graphs. In: International Conference on Algorithmic Learning Theory. Springer, pp. 196–210, 2007.

[MLH09] Maier, Markus; Luxburg, Ulrike V; Hein, Matthias: Influence of graph construction on graph-based clustering measures. In: Advances in neural information processing systems. pp. 1025–1032, 2009.

[Pe01] Pearson, Karl: LIII. On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2(11):559–572, 1901.

[SB14] Sardana, Divya; Bhatnagar, Raj: Graph clustering using mutual K-nearest neighbors. In: International Conference on Active Media Technology. Springer, pp. 35–48, 2014.

[SG02] Strehl, Alexander; Ghosh, Joydeep: Cluster ensembles—a knowledge reuse framework for combining multiple partitions. Journal of machine learning research, 3(Dec):583–617, 2002.

[SM00] Shi, Jianbo; Malik, Jitendra: Normalized cuts and image segmentation. Departmental Papers (CIS), p. 107, 2000.

[SRS00] Sridhar, Ramaswamy; Rastogi, Rajeev; Shim, Kyuseok: Efficient algorithms for mining outliers from large data sets. In: International Conference on Management of Data: Proceedings of the 2000 ACM SIGMOD international conference on Management of data: Dallas, Texas, United States. volume 15, pp. 427–438, 2000.

[VL07] Von Luxburg, Ulrike: A tutorial on spectral clustering. Statistics and computing, 17(4):395–416, 2007.

[VPRS11] Vega-Pons, Sandro; Ruiz-Shulcloper, José: A survey of clustering ensemble algorithms. International Journal of Pattern Recognition and Artificial Intelligence, 25(03):337–372, 2011.

[VRB02] Veenman, Cor J.; Reinders, Marcel J. T.; Backer, Eric: A maximum variance cluster algorithm. IEEE Transactions on pattern analysis and machine intelligence, 24(9):1273–1280, 2002.

[Wu13] Wu, Junjie; Liu, Hongfu; Xiong, Hui; Cao, Jie: A theoretic framework of k-means-based consensus clustering. In: Twenty-Third International Joint Conference on Artificial Intelligence. 2013.

[Za71] Zahn, Charles T: Graph-theoretical methods for detecting and describing gestalt clusters. IEEE Transactions on computers, 100(1):68–86, 1971.