Executive Summary of the Thesis

Cluster Ensemble via Synchronized Relabeling

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Michele Alziati, Giovanni Amarù

Advisor: Prof. Federica Arrigoni

Co-advisor: Prof. Luca Magri

Academic year: 2022-2023

1. Introduction

One of the goals in unsupervised learning is to recognize hidden structures and patterns within data. For this reason there exist a multitude of different clustering algorithms, whose purpose is the categorization of different structures of data in multiple groups, in an autonomous way. Each algorithm operates under specific assumptions and captures diverse properties of clusters, yielding very different results. To address these limitations and improve clustering outcomes, Cluster Ensemble methods have been proposed [30][34][5]. Their aim is to combine the outputs of multiple clustering algorithms or runs over the same set of data objects, to obtain a single result which enhances the overall quality and stability. Each clustering in the ensemble is called a Partition, and the output is called the Consensus Partition.

In Figure 1 the overall method is shown: we start from from a generated ensemble, where the partitions are represented by the blocks on the left, each of which contains its own number of clusters, represented by differently colored circles that partition the dataset. On this ensemble, the algorithm, which is represented by the box in the middle, is applied, and this returns a new partition with its own number of clusters, represented by the final block, having a different number of differently colored circles.

A Cluster Ensemble can be used in a variety of application fields, for example image-segmentation [35] or data mining [15]. It can be very useful when it is required to make decisions when there is high heterogeneity in how data should be classified, for example malware detection [18] and discovery of cancer types [36]. The main limitations of cluster ensemble methods is that typically they are computationally intensive as they require the extraction and combination of multiple clustering from the data. In addition, they are very sensitive to the quality and diversity of the input clusterings.

The main challenge is combining the partitions generated by various clustering algorithms in an ensemble, as it cannot be done in a simple and straightforward way and different algorithms make different assumptions to simplify the problem. Additionally some methods require the selection of a reference partition [31] or ordering [3], leading to potential bias.

The input partitions are defined up to an arbitrary permutation, as shown in Figure 2. On the left it is shown that each partition has different labels, represented by clusters of different colors, which may represent the same or similar information. The process of finding an optimal

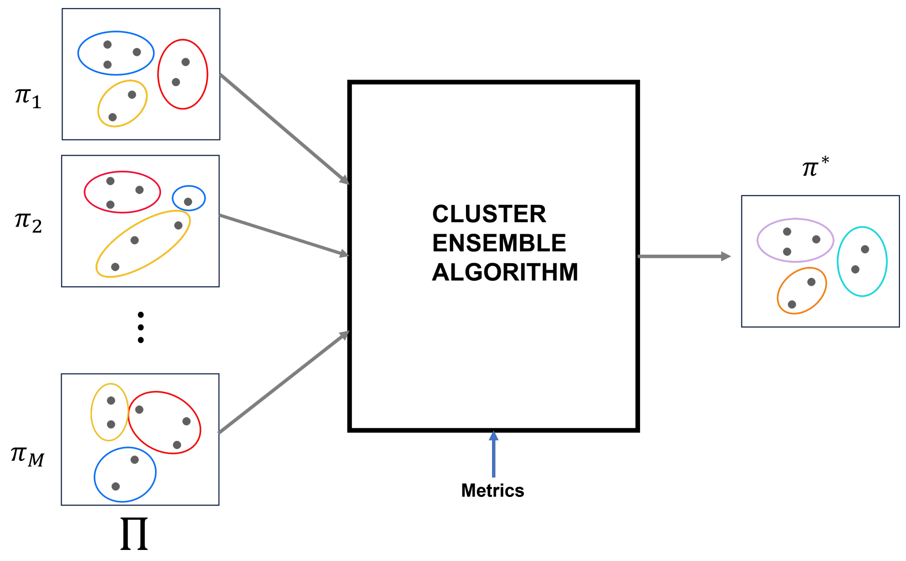


Figure 1: Framework of the Cluster Ensemble Problem

permutation of the labels is known as *relabeling*: this phase brings coherence among the labels in the input partitions and this is shown by the partitions on the right, which have the same cluster labels grouping the same points more or less. This, in combination with a voting technique, produces a solution to the cluster ensemble problem.
We propose a method to address the problem by employing a *Permutation Synchronization* framework [28], to achieve flexibility and robustness. Instead of retrieving the correct matching for the cluster labels by choosing a reference, our method considers all input partitions on equal footing and recovers in one shot, all the relabelings for each partition, providing a global view.

2. Problem Formulation

The problem of Cluster Ensemble can be formulated as follow: $X = \{x_1, \dots, x_N\}$ is a set of N data points, where each $x_i \in X \subset \mathbb{R}^D$ is a D-dimensional vector.
Let $\mathcal{P}_X$ be the set of all possible partitions over the data set X. An ensemble Π is defined as a set of M partitions:

$$\Pi = \{\pi_1, ..., \pi_M\} \subseteq \mathcal{P}_X$$

Each partition π_g is composed of k_g clusters, which are a subset of the dataset X. Each partition may have a different number of clusters:

$$\pi_g = \{C_1, C_2, \dots, C_{k_g}\}$$

Each partition π_g can be viewed as a map that brings a set of data points X to a set of labels $l_g = 1, 2, ..., k_g$, namely:

$$\pi_g : X \to l_g \tag{1}$$

such that $l_g(x_i) = k$ if point i belongs to cluster k according to partition g.
The main assumptions are that the each cluster is disjoint from one another and each partition clusters all the points in the dataset:

$$C_t \cap C_s = \emptyset, s \neq t$$
$$\bigcup_{t=1}^{k_g} C_t = X$$

The aim of a Cluster Ensemble algorithm is to define a novel partition $\pi^* = \{C_1, \dots, C_{k^*}\}$, called the *consensus partition*, with its own number of clusters k^*. The consensus partition combines the information that can be extracted from the ensemble and represents a summary of the partitions in input.

3. Related Work

We present a taxonomy of Cluster Ensemble algorithms, where we identify an intersection between the categories presented in the works of Vega-Pons [34] and De Amorim [8]. On one hand we have the two macro-categories of Co-Occurrence, which count the occurrence of points in clusters, and Median Partition, that

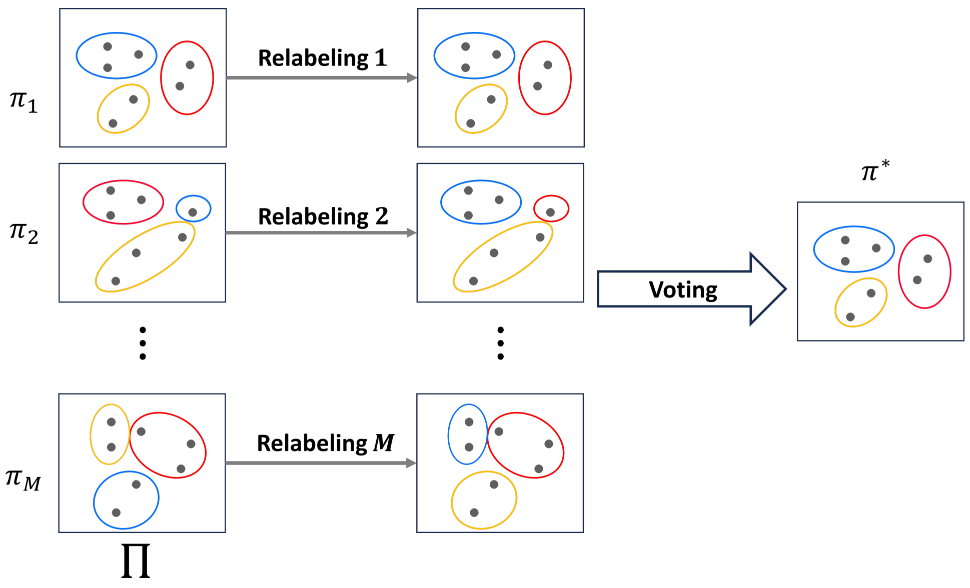


Figure 2: Example of the relabelings retrieved by our method

try to find the median partition using heuristics. The definition of median partition is

$$\pi^* = \arg\max_{\pi \in \mathcal{P}_X} \frac{1}{M} \sum_{g=1}^{M} s(\pi, \pi_g) \tag{2}$$

On the other hand we have the current subcategories:

- Direct methods, which utilize an explicit combination strategy to obtain the consensus
- Feature-based Approaches, which use the features of the points directly, without confronting directly the cluster labels
- Pairwise-similarity based approaches that are based on the similarity between data points
- Graph based methods, which solve the consensus problem by partitioning a graph

Co-occurrence methods are the most adopted ones, since they are simple to understand and quick to implement. From this macro-category our work will focus mainly on some methods from the families of Direct Approach, in particular the ones pertaining to the subfamily of *Relabeling and Voting*, for their intuitive approach to the solution of the Cluster Ensemble problem, and the family of Graph-based methods, for their simplicity and performance.

3.1. Relabeling and Voting methods

These methods are comprised of two main phases, relabeling and voting. Relabeling consists in the process of finding a permutation of cluster labels between different partitions such that the labels in all partitions are aligned and represent the same information. The voting phase consists in selecting the most recurring label for each data point after the cluster labels in all partitions have been aligned. During the voting phase, each relabeled partition represents a vote for a cluster label for all points in the dataset. A simple approach to relabeling is to formulate it as a maximum weight bipartite matching problem[31], which computes the common data points between pairs of cluster labels in two different partitions and tries to maximize the number of commonly assigned points. This problem is then solved by the Hungarian Algorithm [24]. Another approach is to formulate relabeling as a multivariate regression problem where the relabeling is computed by fitting a permutation to a given partition [3], using the mean squared error as loss function. Relabeling is done in two ways: either it is computed between a chosen reference partition and all the others [31], separating clearly the two phases of relabeling and voting, or by choosing an order of partitions and updating the relabelings iteratively, for example as done by ada-bVote and c-Vote [3], so that the two phases are done to-

gether at each iterative step. Voting is done in two ways, depending on the type of relabeling formulation employed: if the relabelings are all calculated before voting, the consensus partition is obtained by summing all votes and then selecting the most recurring label for each point [31]. If relabeling is done iteratively instead, voting is performed at each iteration either as the sum of the vote in the step with all the votes of the previous steps, or as the cumulative weighted average of the votes in each step [3].
It is possible to add precision to the voting phase by adding extra information derived from external indices or a priori information on the data and partitions, which takes the form of weights, using for example the NMI between partitions [39][37].

3.2. Graph-based methods

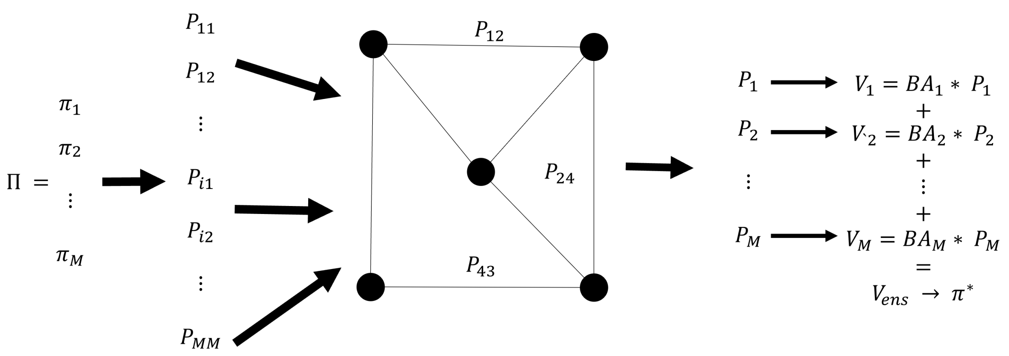
Graph-based algorithms usually require two steps: Construction of a weighted undirected graph from the input data, and Partitioning of the graph in k parts using a graph partitioning technique, the most known being METIS[23] and spectral clustering [27]. These types of algorithms have in common only the fact they use a graph representation, and can be divided between those that use the graph to model the similarity between clusters, those that use it to model the similarity between points, and those that model the relationships between clusters and points. In the first group, MCLA [30] and L-MCLA [29] create a weighted graph based on the similarity between clusters using the Jaccard similarity. In the second group, CSPA [30], SNNC [2], WSPA [9], CTS and SRS[21] create a weighted graph by measuring how often points occur together in the same cluster. In the third group, WBPA [9] creates a graph by measuring the distances between points and clusters, HBGF[12] and LCE[22] create a graph from the known association between points and clusters in each partition. There exist also different methods that construct a hyper-graph from the membership of each point to each cluster, like HGPA [30] and CESHL [38]. Each of these graphs is then partitioned in k^* clusters, which is assumed known a priori. The families of similarities between clusters and relationships between clusters and points are those that can be seen as closer to relabeling and voting, since they group together clusters that hold similar information and decide then which points belong to them the most.

4. Proposed method

We propose a solution to the Cluster Ensemble problem, by formulating the relabeling phase of Relabeling and Voting techniques as a problem of permutation synchronization. We called this novel approach Synchronization and Voting.
To address how our method works, we employ a matrix representation of the ensemble. Each partition π_g can be represented by a $N \times k_g$ matrix BA_g called the *binary association matrix*:

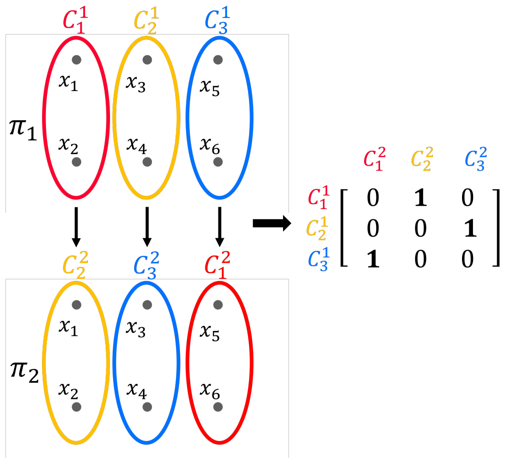
$$[BA_g]_{i,k} = \begin{cases} 1 & \text{if } l_g(x_i) = k \\ 0 & \text{otherwise} \end{cases}$$

Each row corresponds to a data point and each column corresponds to a cluster, where ones reveal the membership of points to a specific cluster. The number of rows in BA_g is equal to the number of points, the number of columns is equal to the number of clusters in partition g.
Relabeling can be seen as the problem of finding an optimal permutation of the columns of BA_g for each g. We look for a permutation matrix P_g of size $k_g \times k^*$, such that BA_gP_g represents the relabeled binary association matrix of π_g. In general, P_g may be a partial permutation.
Our objective is to find M permutation matrices $P_1, \ldots, P_M$ called *absolute permutations* following a consistency criterion. To accomplish this task, we introduce the concept of *relative permutations*. Given two partitions π_i and π_j, we denote by P_{ij} the relative permutation of the pair (i, j), that is the $k_i \times k_j$ permutation matrix that best satisfies the equation $BA_iP_{ij} = BA_j$. P_{ij} represents an optimal relabeling that makes π_i coherent with π_j, in the sense that the number of data points with the same labels across the two partitions is maximized. P_{ij} represents a local relabeling as it involves only two partitions at a time. An example is given in Figure 4.
For given i and j, the relative permutation P_{ij} can be computed by solving a linear assignment problem, usually solved with the Hungarian algorithm [24] or equivalent variations that can solve the problem for partial permutations [].
From the theory of synchronization, the relationship between absolute and relative permutation



$\Pi = \pi_1, \pi_2, \vdots, \pi_M$
$P_{11}, P_{12}, \vdots, P_{i1}, P_{i2}, \vdots, P_{MM}$
P_{12}
P_{24}
P_{43}
$P_1 \rightarrow V_1 = BA_1 * P_1$
+
$P_2 \rightarrow V_2 = BA_2 * P_2$
+
$P_M \rightarrow V_M = BA_M * P_M$
=
$V_{ens} \rightarrow \pi^*$

Figure 3: The pipeline of our proposed method



C_1^1 C_2^1 C_3^1
π_1
x_1 x_2 x_3 x_4 x_5 x_6
C_2^2 C_3^2 C_1^2
π_2
x_1 x_2 x_3 x_4 x_5 x_6

$$\begin{array}{c|ccc} & C_1^2 & C_2^2 & C_3^2 \\ \hline C_1^1 & 0 & \mathbf{1} & 0 \\ C_2^1 & 0 & 0 & \mathbf{1} \\ C_3^1 & \mathbf{1} & 0 & 0 \end{array}$$

Figure 4: Local relabeling yielding a relative permutation

is given by:

$$P_{ij} = P_i * P_j^T \tag{3}$$

which is known as *consistency constraint*, which means that the local relabeling between i and j should be equal to the composition between the absolute relabeling of i and the inverse of the absolute relabeling of j.

The problem can be formulated as a graph $G = (V, E)$ where vertices correspond to unknown absolute permutations and edges correspond to known relative permutations. The number of vertices is equal to the number of input partitions, i.e. M. The graph can be constructed by calculating relative permutations between all pairs of partitions, resulting in a complete graph, but it also can, for efficiency reasons, be constructed with a subset of all possible pairs resulting in a graph with missing edges.

The problem of finding M unknown permutation matrices $P_1, \ldots, P_M$ starting from a redundant set of relative permutations P_{ij}, with $(i, j) \in E$ such that the consistency constraint of Eq. 3 is satisfied is known in literature as *permutation synchronization* [28]. The relative permutations are collected in the block matrix $\mathbf{P}$:

$$\mathbf{P} = \begin{bmatrix} P_{11} & \dots & P_{1M} \\ \vdots & \ddots & \vdots \\ P_{M1} & \dots & P_{MM} \end{bmatrix}$$

where missing edges (if any) are zero blocks. We focus on three different permutation synchronization approaches:

- In [28] the absolute permutations are recovered by finding the k^* leading eigenvectors of $\mathbf{P}$. This method works under the assumptions that k^* is known a priori and all permutations are total.
- In [4] the absolute permutations are computed from the non-negative matrix factorization (NMF) of $\mathbf{P}$. This method works with partial permutations but requires to know k^* in advance.
- In [32] the matrix $\mathbf{P}$ is viewed as the adjacency matrix of an M-partite graph and permutation synchronization is cast to a graph-clustering problem; this method is named QuickMatch, it works with partial permutations and it automatically estimates the value of k^*.

Our method is composed of two steps:

1. the computation of the relative permutations $P_{12}, \ldots, P_{ij}, \ldots$ by solving multiple linear assignment problems;
2. the recovery of the absolute permutations $P_1, \ldots, P_M$ by solving a single permutation synchronization problem.

Once relabeling is solved, any voting technique can be used to derive the consensus partition.

Our method is named SV, shorthand for synchronization and voting. When used in combination with [28] it is named SV-EIG, when combined with [4] it is named SV-NMF and when combined with [32] it is named SV-QM.
The main advantage of our method is that it is modular and many version of permutation synchronization can be plugged in depending on the use case. Another advantage it has over known relabeling and voting methods, such as [31], is the fact that it doesn't depend on a reference partition for relabeling and considers a global view, taking into account the redundancy represented by the whole graph, promoting error compensation.

5. Experinents

To assess and verify the assumptions we made for Synchronization and Voting, we ran different experiments using different metrics. We used multiple datasets, both synthetic and from real-world measurements, to create heterogeneous testing scenarios with different characteristics in ensemble generation, dataset features and number of data points.

5.1. Metrics

As previously explained, there isn't a unanimous consensus on the definition of the optimal solution of the Cluster Ensemble. For this specific experimental assessment we decided to use metrics that evaluate the similarity with respect the partitions in input and the ground truth. This metrics are widely used in literature, and measure different characteristic of the solution.
We have chosen NMI, ARI, classification accuracy and execution time.

NMI and ARI The Normalized Mutual Information (NMI) measure how close a partition is to another, by looking at how much information they have in common considering the statistical information they share, that in Clustering is represented by the common labelled points between clusters. The NMI is used to measure the consistency of the found consensus partition with right to the ensemble by computing the average NMI:

$$ANMI(\pi^*, \Pi) = \frac{1}{M} \sum_{\pi_g \in \Pi} NMI(\pi^*, \pi_g)$$

NMI doesn't need a priori information about the cluster labels or need to match the labels between two different partitions, making it easy and fast to use.
The Adjusted Rand Index (ARI), measures the similarity between partitions instead by considering the overlapping between all the possible clusters, by looking at all the combinations of shared points between clusters.
Just like the NMI, ARI is used to measure the consistency of the found consensus partition with right to the ensemble by computing the average ARI:

$$AARI(\pi^*, \Pi) = \frac{1}{M} \sum_{\pi_g \in \Pi} ARI(\pi^*, \pi_g)$$

Classification accuracy From a quantitative standpoint, it can be defined as

$$ACC(\pi^*, \pi_{true}) = 1 - \frac{\sum_{x_i=1}^{N} l^*(x_i) \neq l^{true}(x_i)}{N}$$

assuming that the clusters between the consensus and ground truth partitions are already matched. This measure is used to quantify how accurate the consensus partition is w.r.t to the ground truth, when available.

Execution Time Every method we tested had its execution time measured from the start of the method (from the moment it receives the ensemble data in input), to the end (the moment in which the final voting matrix is returned), in seconds.

5.2. Compared Methods

- Weighted Bipartite Matching [31]
- Multivariate Regression [3]
- MCLA [30]
- CSPA [30]
- CESHL [38]
- Random selection of a partition

CSPA is a state of the art algorithm developed by Strehl [30] that uses a different approach not comparable directly to Relabeling And Voting. Nonetheless We decided to insert it in our experiments because is widely used for comparisons and it's a valid benchmark. For the voting part, we have chosen simple voting and weighted voting, choosing as weights the NMI with respect to a reference and the inverse of the average NMI over all partitions.

5.3. Datasets

For testing we used a synthetic dataset generated with a Gaussian distribution of the data over clusters and 9 real-world datasets, provided with the ground truth and presented in table 1.

Datasets			
Name	*N*	*D*	*k*
Iris [14] [13]	150	4	3
Multiple Feature [10] [6]	2000	649	10
Mnist [25]	5000	784	10
Usps [20] [19]	11000	256	10
Isolet [7] [11]	1560	617	26
Lung Cancer [17] [16]	203	3312	5
Wine [1] [33]	178	13	3
Silhouette [26]	150	18	3

Table 1: List of tested datasets

5.4. Results

We obtained some significant findings by testing our method in the various scenarios we tested.

Better performance on complex datasets We found that SV-EIG, SV-NMF and SV-QM obtained slightly better results compared to the other ones on dataset particularly difficult to cluster, due to the high number of features, the high number of points and the number of clusters in the ground truth (Usps, Isolet, Mnist and Multiple Features). In all the other cases the performances were either comparable or slightly better to the other methods.

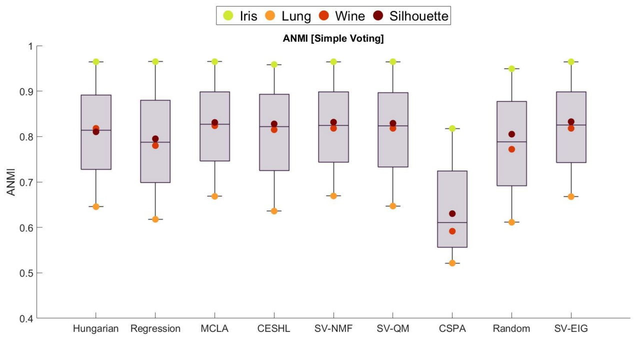


Figure 5: NMI on low complexity datasets

Faster in real-case scenarios From the experiments on the synthetic dataset we found that the three methods we presented are faster than the competitors based on graph partitioning in case the number of partitions in less than 50, which is a reasonable upper bound for real cases scenarios. With a bigger number of partitions instead the time tends to increase more than the graph-based methods. The same considerations can be done by keeping fixed the number of partitions and increasing the number of cluster labels of each partition.

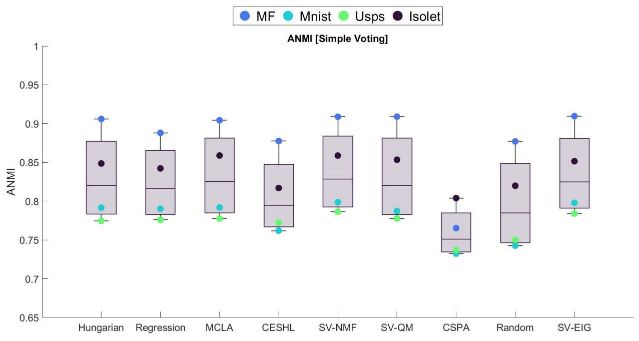


Figure 6: NMI on High complexity datasets

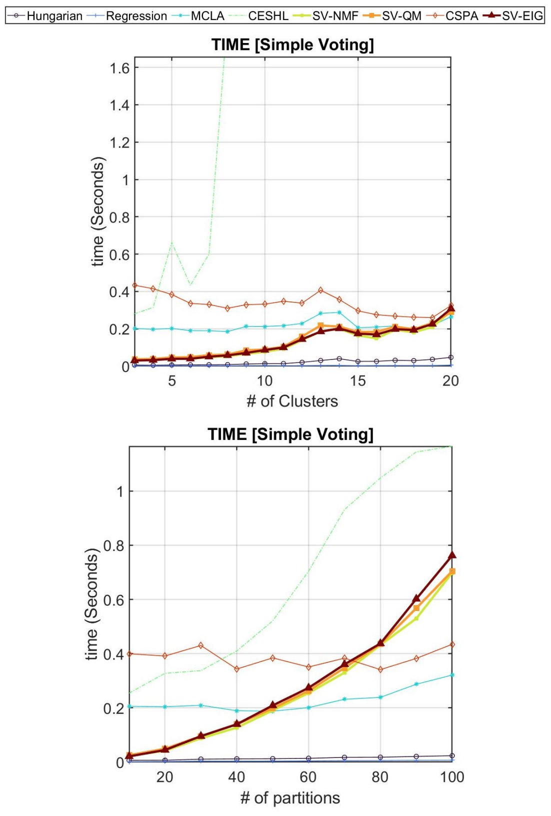


Figure 7: Execution Time with varying number of clusters and partitions

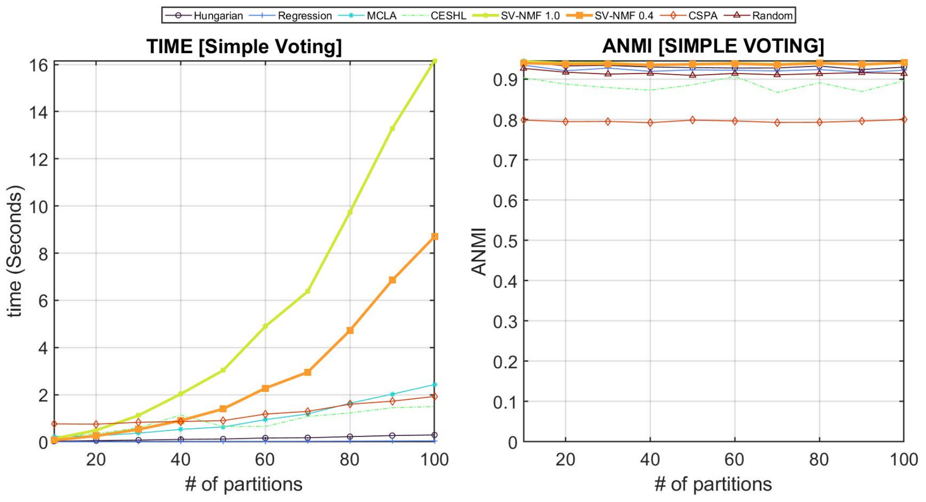


Figure 8: Comparison between usage of all the edges and only the 40 %

Further Considerations The framework of Synchronization and Voting compared to the competitors we shown, offers the possibility to customize the performance and execution time by selecting an appropriate parameter which regulates the completeness of the graph generated for synchronization, so that it can be suitable to different use cases, especially when execution time is a sensitive issue. In our experiment setup, with our specific ensemble generation, we could exploit a reduced number of computations for the edges without having a significant loss in performance.

6. Conclusions

The Cluster Ensemble Problem can be hard to solve and there exist a variety of methods that try to solve it using a multitude of approaches, depending on the usage-scenario and the knowledge required on the data. We presented a novel approach, called Synchronization and Voting, that takes advantage of the Graph Synchronization technique to assess the problem of relabeling between partitions: this is the first time in literature that this type of formulation is applied to the framework of Relabeling And Voting, and presents clear advantages in different use cases. There are specific assumptions on the number of partitions in the ensemble, which is the range of the number of partitions used in the vast majority of real-world scenarios. The execution time is lower with respect to comparable methods, and can be furthermore optimized by using a subset of edges for the creation of the graph. This choice in specific scenarios doesn't alter the performance in a significant way. The overall performance is higher, particularly on datasets that present a sizable number of data points, clusters and features.

References

[1] Stefan Aeberhard and M. Forina. Wine. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C5PC7J.

[2] Hanan Ayad and Mohamed Kamel. Finding natural clusters using multi-clusterer combiner based on shared nearest neighbors. *Multiple Classifier Systems*, pages 166–175, 2003.

[3] Hanan G. Ayad and Mohamed S. Kamel. On voting-based consensus of cluster ensembles. *Pattern Recognition*, 43:1943–1953, 2010.

[4] Florian Bernard, Johan Thunberg, Jorge Goncalves, and Christian Theobalt. Synchronisation of partial multi-matchings via non-negative factorisations. *Pattern Recognition*, 92:146–155, 2019.

[5] Tossapon Boongoen and Natthakan Iam-On. Cluster ensembles: A survey of approaches with recent extensions and applications. *Computer Science Review*, 28:1–25, 2018.

[6] M Van Breukelen, R P W Duin, D M J Tax, and J E Den Hartog. Handwritten digit

recognition by combined classifiers. *Kybernetika*, 34:381–386, 1998.

[7] Ron Cole and Mark Fanty. ISOLET. UCI Machine Learning Repository, 1994. DOI: https://doi.org/10.24432/C51G69.

[8] Renato Cordeiro de Amorim, Andrei Shestakov, Boris Mirkin, and Vladimir Makarenkov. The minkowski central partition as a pointer to a suitable distance exponent and consensus partitioning. *Pattern Recognition*, 67:62–72, 2017.

[9] Carlotta Domeniconi and Muna Al-Razgan. Weighted cluster ensembles: Methods and analysis. *ACM Trans. Knowl. Discov. Data*, 2, 2009.

[10] Robert Duin. Multiple Features. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5HC70.

[11] Mark Fanty and Ronald Cole. Spoken letter recognition. In *Advances in Neural Information Processing Systems*, volume 3, 1990.

[12] Xiaoli Zhang Fern and Carla E Brodley. Solving cluster ensemble problems by bipartite graph partitioning. page 36. Association for Computing Machinery, 2004.

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

[14] R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

[15] Derek Greene and Pádraig Cunningham. Efficient ensemble methods for document clustering. *University of Dublin, Trinity College, Dublin 2, Ireland*, 2013.

[16] Zi Quan Hong and Jing Yu Yang. Optimal discriminant plane for a small number of samples and design method of classifier on the plane. *Pattern Recognition*, 24:317–324, 1991.

[17] Z.Q. Hong and J.Y Yang. Lung Cancer. UCI Machine Learning Repository, 1992. DOI: https://doi.org/10.24432/C57596.

[18] Shifu Hou, Lifei Chen, Egemen Tas, Igor Demihovskiy, and Yanfang Ye. Cluster-oriented ensemble classifiers for intelligent malware detection. *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing, IEEE ICSC 2015*, pages 189–196, 2015.

[19] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:550–554, 1994.

[20] Jonathan J. Hull. Usps, 1994. https://jundongl.github.io/scikit-feature/datasets.html.

[21] Natthakan Iam-on, Tossapon Boongoen, and Simon Garrett. Refining pairwise similarity matrix for cluster ensemble problem with cluster relations. In Jean-François Jean-Fran, Michael R Berthold, and Tamás Horváth, editors, *Discovery Science*, pages 222–233. Springer Berlin Heidelberg, 2008.

[22] Natthakan Iam-on, Tossapon Boongoen, and Simon Garrett. Lce: a link-based cluster ensemble method for improved gene expression data analysis. *Bioinformatics*, 26:1513–1519, 2010.

[23] George Karypis and Vipin Kumar. Kumar, v.: A fast and high quality multilevel scheme for partitioning irregular graphs. siam journal on scientific computing 20(1), 359-392. *Siam Journal on Scientific Computing*, 20, 11 1999.

[24] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[25] Yann LeCun and Corinna Cortes. Mnist handwritten digit database. *AT & T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 7, 2010.

[26] Pete Mowforth and Barry Shepherd. Statlog (Vehicle Silhouettes). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5HG6N.

[27] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In T Dietterich,

S Becker, and Z Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[28] Deepti Pachauri, Risi Kondor, and Vikas Singh. Solving the multi-way matching problem by permutation synchronization. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[29] Changlong Shao and Shifei Ding. Link-based cluster ensemble method for improved meta-clustering algorithm. *IFIP Advances in Information and Communication Technology*, 581 AICT:14–25, 2020.

[30] Alexander Strehl and Joydeep Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617, 2003.

[31] A P Topchy, M H C Law, A K Jain, and A L Fred. Analysis of consensus partition in cluster ensemble. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 225–232, 2004.

[32] Roberto Tron, Xiaowei Zhou, Carlos Esteves, and Kostas Daniilidis. Fast multi-image matching via density-based clustering. *IEEE International Conference on Computer Vision*, 2017.

[33] B. Vandeginste. Parvus: An extendable package of programs for data exploration, classification and correlation. *Journal of Chemometrics*, pages 191–193, 1990.

[34] Sandro Vega-Pons and José Ruiz-Shulcloper. A survey of clustering ensemble algorithms. *IJPRAI*, 25:337–372, 2011.

[35] P. Wattuya, K. Rothaus, J. S. Praßni, and X. Jiang. A random walker based approach to combining multiple segmentations. *Proceedings - International Conference on Pattern Recognition*, 2008.

[36] Zhiwen Yu, Le Li, Hau San Wong, Jane You, Guoqiang Han, Yunjun Gao, and Guoxian Yu. Probabilistic cluster structure ensemble. *Information Sciences*, 267:16–34, 2014.

[37] Li Zhang, Weida Zhou, Caili Wu, Jieting Huo, Haishuang Zou, and Licheng Jiao. Center matching scheme for k-means cluster ensembles. In Mingyue Ding, Bir Bhanu, Friedrich M. Wahl, and Jonathan Roberts, editors, *MIPPR 2009: Pattern Recognition and Computer Vision*, volume 7496, page 749614. SPIE, 2009.

[38] Peng Zhou, Xia Wang, Liang Du, and Xuejun Li. Clustering ensemble via structured hypergraph learning. *Information Fusion*, 78:171–179, 2022.

[39] Zhi Hua Zhou and Wei Tang. Clusterer ensemble. *Knowledge-Based Systems*, 19:77–83, 2006.

Cluster Ensemble via Synchronized Relabeling

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Michele Alziati, Giovanni Amarù**

Student ID: 976209, 962839
Advisor: Prof. Federica Arrigoni
Co-advisors: Prof. Luca Magri
Academic Year: 2022-23

Abstract

The cluster ensemble problem is an important problem in unsupervised learning that aims at aggregating multiple noisy partitions into a unique/better clustering solution. It can be formulated in terms of relabelling and voting, where relabelling refers to the task of finding optimal permutations that bring coherence among labels in input partitions. In this thesis we propose a novel solution to the relabelling problem based on permutation synchronization. By effectively circumventing the need for a reference clustering, our method achieves superior performance than previous work under varying assumptions and scenarios, demonstrating its capability to handle diverse and complex datasets.

Keywords: unsupervised learning, clustering, cluster ensemble, consensus function, graph synchronization, permutation synchronization, relabeling and voting, synchronization and voting

Abstract in lingua italiana

Il problema del cluster ensemble è un importante problema nel contesto dell'unsupervised learning che punta ad aggregare più partizioni rumorose in una unica/migliore soluzione di clustering. Può essere formulato in termini di relabeling e voting, dove il relabeling si riferisce al problema di trovare permutazioni ottimali che portino coerenza tra le label nelle partizioni in input. In questa tesi proponiamo una soluzione innovativa al problema del relabeling, basata sulla sincronizzazione di permutazioni. Aggirando effettivamente la necessità di un clustering di riferimento, il nostro metodo raggiunge prestazioni superiori rispetto a lavori preesistenti tenendo conto di diverse ipotesi e scenari, dimostrando la sua capacità di gestire dataset eterogenei e complessi.

Parole chiave: apprendimento non supervisionato, clustering, cluster ensemble, funzione consenso, sincronizzazione dei grafi, sincronizzazione di permutazioni, relabeling e voting, sincronizzazione e voting

Contents

1 | Introduction

One of the goals in unsupervised learning is to recognize hidden structures and patterns within data. For this reason there exist a multitude of different clustering algorithms, whose purpose is the categorization of different structures of data in multiple groups, in an autonomous way. Each algorithm operates under specific assumptions and captures diverse properties of clusters, yielding very different results. To address these limitations and improve clustering outcomes, Cluster Ensemble methods have been proposed [43][48][7]. Their aim is to combine the results of multiple clustering algorithms or runs over the same set of data objects, to obtain a single new result which enhances the overall quality and stability. Each clustering in the ensemble is called a Partition, and the output is called the Consensus Partition. The partitions in the ensemble can be obtained in a variety of ways: for example, by using different algorithms, different subsets of data objects or different runs of the same algorithm tuned with different parameters or initialization. With Cluster Ensemble methods it's possible to obtain a summary of all these different results by aggregating their information. This aggregation should have better quality than a single result, and in turn provide a new view on how the data should be organized. A Cluster Ensemble can be used in a variety of application fields, for example image-segmentation [49] or data mining [22]. It can be very useful when it is required to make decisions when there is high heterogeneity in how data should be classified, for example malware detection [25] and discovery of cancer types [51].

The main limitations of cluster ensemble methods is that typically they are computationally intensive as they require the extraction and combination of multiple clustering from the data, depending on the number of input partition provided to the algorithm. In addition, they are very sensitive to the quality and diversity of the input clusterings, since low quality partitions affect the overall quality of the consensus partition.

The main challenge is combining the partitions generated by various clustering algorithms in an ensemble, as it cannot be done in a simple and straightforward way and different algorithms make different assumptions to simplify the problem. Additionally some methods require the selection of a reference partition [45] or ordering [4], leading to potential bias.

There exist a variety of cluster ensemble algorithms, who approach the problem in many different ways. Some of them may solve directly the problem by approaching it as an optimization problem [8], while other algorithms may solve first a variety of sub-problems before yielding the solution, such as bringing coherence to the input cluster labels [45] or partitioning a graph generated from the input data [43].

In general, the input partitions are defined up to an arbitrary permutation. An approach to solve the Cluster Ensemble problem, which is also one of the most famous sub-problems mentioned above, is to make the cluster labels in the different partitions coherent with each other so that the extraction of common information or the detection of anomalies is easier. As shown in Figure 1.1, on the left each partition has different cluster labels, represented by clusters of different colors, which may represent the same or similar information. The process of finding an optimal permutation of the labels is known as *relabeling*: this phase brings coherence among the labels in the input partitions and this is shown by the partitions on the right, which have the same cluster labels grouping the same points more or less. This, in combination with a voting technique, produces a solution to the cluster ensemble problem. There exist different algorithms that find the consensus parition in this way, and they are called *Relabeling & Voting algorithms*.

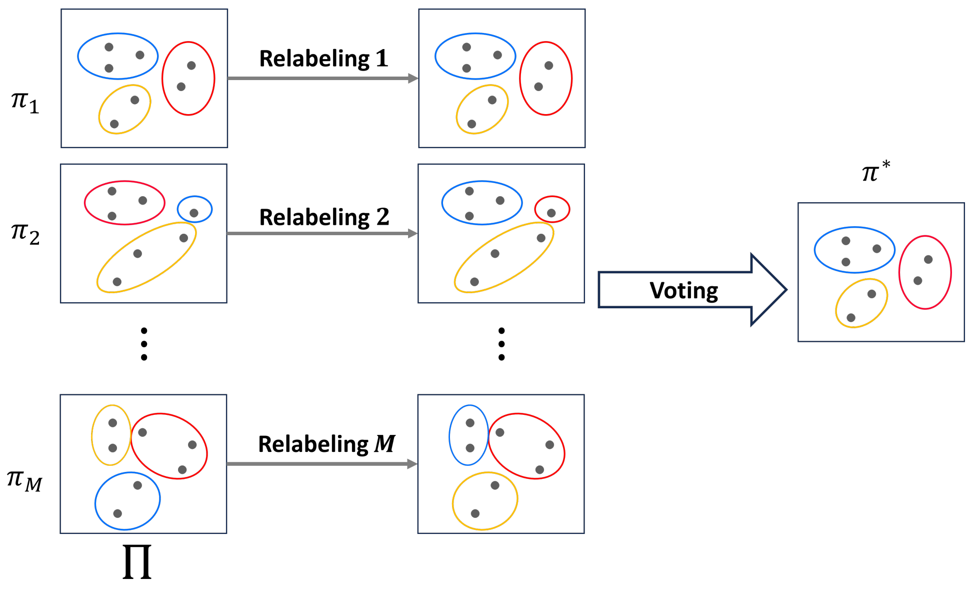


Figure 1.1: Phases of a relabeling and voting algorithm

We propose a method to address the problem by employing a *Permutation Synchronization* framework [39], to achieve flexibility and robustness. Instead of retrieving the correct matching for the cluster labels by choosing a reference, our method considers all input partitions on equal footing and recovers in one shot, all the relabelings for each partition, providing a global view. We also made available the code we developed to assess this problem, it is a working implementation of a cluster ensemble algorithm using permutation synchronization, called *Synchronization and Voting*[1].

This thesis is structured as follows: in Chapter 2 we provide a formal description of the specifics of the cluster ensemble problem, its output and the assumptions we have made addressing it. In Chapter 3 we provide an overview of the literature regarding the state of the art of cluster ensemble, focusing on two particular categories of algorithms we deemed relevant for our proposed method: Relabeling And Voting and Graph-based techniques. In Chapter 4 we describe the theoretical background necessary to understand the algorithms employed in our method, while in Chapter 5 we explain our solution to the cluster ensemble problem. In Chapter 6 we show the results obtained by assessing the performance of our method using different metrics on different datasets, and in Chapter 7 we draw our conclusions based on the experimental results and discuss some possible developments.

[1]`https://github.com/AlziatiM/SynchronizationAndVoting`

2 | Problem Formulation

In this chapter we provide all the useful notation, equations and concepts to provide a conceptual framework for the algorithms we describe in the following chapters.

2.1. The cluster ensemble problem

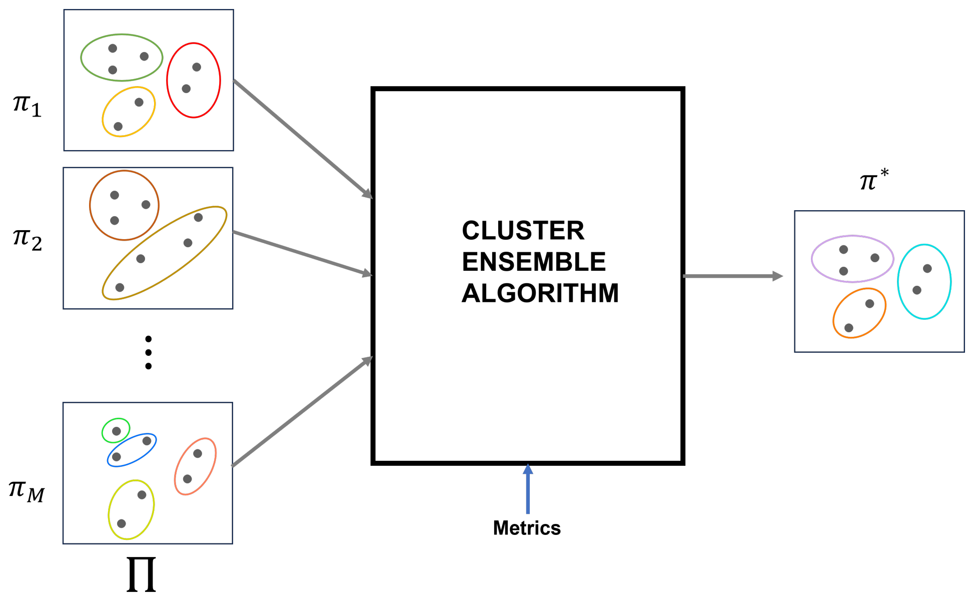


Figure 2.1: General schematic of the cluster ensemble Problem

The Cluster Ensemble problem, also known as Clustering Ensemble or Ensemble Clustering, first discussed by Strehl [43], consists in the task of combining the knowledge from different *partitions*, which consists in a set of clusters over the same set of data objects, to obtain a consensus on what the final clustering of the dataset should be, called *Consensus Partition*. In Figure 2.1 we illustrate the general framework of the problem: each partition has its own set of clusters, represented in the figure by a different set of colored groups, which contain a different set of points of the original dataset. To these partitions a cluster ensemble algorithm is applied, which returns the consensus partition, which may have its own different set of clusters, represented in the figure by the difference in the colors of the

clusters. A partition is generated by a specific run of any given clustering algorithm with specific parameters, and the ensemble can be generated by creating partitions by using different algorithms or different parameters. Each partition may provide a different view on how the data should be grouped in clusters.
The purpose of combining the information from each partition is to create a final partition that may have a better quality than each single one, and in turn providing a new view on how the data should be organized, which may be closer to the ground truth.

2.1.1. Input And Output

The input of the problem is as follows:

- $X = \{x_1, \dots, x_N\}$ is a set of N data points, where each $x_i \in X \subset \mathbb{R}^D$ is represented by a vector of D attribute values, i.e., $x_i = (x_{i,1}, \dots, x_{i,D})$. There are cases where X is not required for the computation of the output, and an Ensemble is provided directly to the algorithm.

- Let $\mathscr{P}_X$ be the set of all possible partitions over the data set X. An ensemble Π is defined as a set of M base clusterings, each of which is referred to as an 'ensemble member' or 'partition':

  $$\Pi = \{\pi_1, ..., \pi_M\} \subseteq \mathscr{P}_X$$

  A partition $\pi \in \mathscr{P}_X$ consists in a set of k_g clusters, which are a subset of the dataset X. Each partition may have a different number of clusters.

  $$\pi = \{C_1, C_2, \dots, C_{k_g}\}, \quad C_t \subset X$$

  Each partition π_g can be viewed as a map that brings a set of data points X to a set of labels $l_g = 1, 2, ..., k_g$, namely:

  $$\pi_g : X \to l_g$$

  such that $l_g(x_i) = k$ if point i belongs to cluster k according to partition g. l_g is called the *label vector* of partition π_g

  The main assumptions are that the each cluster is disjoint from one another and each partition clusters all the points in the dataset. This property is called *hard clustering*.

$$C_t \cap C_s = \emptyset, \quad s \neq t$$

$$\bigcup_{t=1}^{k_g} C_t = X$$

The aim of a Cluster Ensemble algorithm is to find a partition $\pi^* = \{C_1, \dots, C_{k^*}\}$, with its own number of clusters k^*, called the *consensus partition.* The consensus partition combines the information that can be extracted from the ensemble and represents a summary of the partitions in input.

2.1.2. Goals

The goodness of the consensus partition is evaluated using some similarity function $s(\pi_i, \pi_j)$, which ideally should be higher with respect to all partitions in the ensemble:

$$\pi^* = arg \max_{\pi \in \mathscr{P}_X} \frac{1}{M} \sum_{g=1}^{M} s(\pi, \pi_g) \tag{2.1}$$

. For example, s can be the *Normalized Mutual Information*(NMI), a measure from information theory which is defined as such:

$$NMI(\pi_a, \pi_b) = \frac{\sum_{i=1}^{k_a} \sum_{j=1}^{k_b} n_{ij} \log(\frac{n_{ij} \cdot N}{|C_i^a| \cdot |C_j^b|})}{\sqrt{(\sum_{i=1}^{k_a} |C_i^a| \log(\frac{|C_i^a|}{N}))(\sum_{j=1}^{k_b} |C_j^b| \log(\frac{|C_j^b|}{N}))}}$$

where $n_{ij} = |C_i^a \cap C_j^b|$ is the number of commonly labeled points between two clusters.

Other similarity functions, such as the *Adjusted Rand Index* [26] or the *Classification Accuracy* [31] can also be used to measure the quality of the consensus partition. They will be explained more in detail in Section 6.

2.2. Notation

To better understand the techniques and algorithms used to approach the problem, we introduce some notation and a new representation of the data to express precisely how relevant values are extracted from the inputs.

Total number of clusters Some algorithms requires to know the overall number of distinct cluster labels in the Ensemble. We call this measure k_{tot} and it is defined as follows:

$$k_{tot} = \sum_{g=1}^{M} k_g$$

In case the ensemble is generated with a fixed $k_g = k \quad \forall g \in \{1, \dots, , M\}$, k_{tot} is simply equal to $M * k$.

Binary Association Matrix

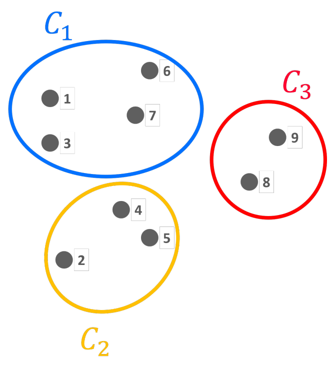
$$u_g \colon X \times \{1, \dots, k_g\} \to [0, 1]$$
$$u_g(x_i, t) = P(x_i \in C_t) = P(l(x_i) = t)$$

u represents the probability of a point being assigned to a given cluster in a partition. It is used to construct another useful representation of the input, a $N \times k_g$ matrix called the *stochastic matrix* $[U_g]_{i,t}$. Each element of U is subject to:

$$\sum_{t=1}^{k_g} u(x_i, t) = 1, \quad \forall i$$

In general, U may represent a hard partition with $u(x_i, t) \in \{0, 1\}$ or a soft partition with $u(x_i, t) \in [0, 1]$. In the former case the stochastic matrix takes the name of *Binary Cluster-Association matrix* (BA).

$$[BA_g]_{i,k} = \begin{cases} 1 & \text{if } l_g(x_i) = k \\ 0 & \text{otherwise} \end{cases}$$



$$U(X,T) = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{array} \begin{array}{c} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} \\ \begin{bmatrix} 0.8 & 0.15 & 0.05 \\ 0.1 & 0.7 & 0.2 \\ 0.8 & 0.1 & 0.1 \\ 0.05 & 0.9 & 0.05 \\ 0.1 & 0.6 & 0.3 \\ 0.7 & 0.15 & 0.15 \\ 0.7 & 0.1 & 0.2 \\ 0.1 & 0.1 & 0.8 \\ 0 & 0.05 & 0.95 \end{bmatrix} \end{array} \quad BA(X,T) = \begin{array}{c} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \quad l(X) = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 2 \\ 2 \\ 1 \\ 1 \\ 3 \\ 3 \end{bmatrix}$$

Figure 2.2: Example of label vector, stochastic and binary association matrices of a given partition.

In Figure 2.2 we propose a simple example of how these different functions work: in the example partition there are three clusters, denoted by the blue, yellow and red colors. In each representation we color-coded the columns or the entries of each matrix or vector with the color associated to that specific cluster.

3 | Related Work

In this chapter we propose a look at the general framework of Cluster Ensemble methods, some history and well known algorithms. An important premise is to be made, which is the main drive behind the existence of Cluster Ensemble as a problem: there is no single clustering algorithm that is able to yield the best performance for all datasets and no algorithm is able to discover all types of clusters. Different clustering algorithms can produce very different results, even with small parameter changes.

For this reason Cluster Ensemble algorithms exist, their main goal is to combine the information from a collection of different sets of clusters in a way that achieves "better results" with respect to those of an individual algorithm, and provide either a new vision of how the data should be classified or return a final classification that should ideally be closer to the truth of how those data should be grouped in clusters.

Given a set of data points X, the framework of a cluster ensemble method consists in two main steps:

- Generation of the ensemble: this phase consists in creating a set of partitions from X.
- Combination: the consensus partition is computed, by using the information from the partitions obtained in the generation step, using a specific procedure.

These are some key properties that a clustering ensemble algorithm should fulfill, as stated by *Topchy et al.* [44]:

- Robustness: The performance of a Cluster ensemble algorithm should have a good average performance between different domains and datasets.
- Consistency: The result of the combination should be somehow, very similar to all combined partitions. Consistency can be measured, for example, using metrics like the average NMI over all the partitions in the ensemble.
- Novelty : Cluster ensemble algorithms can find new solutions unattainable by single clustering algorithms.

- Stability: Results have low sensitivity to noise and outliers.

The consensus partition obtained from a Cluster Ensemble algorithm cannot be expected to be exactly equal to the ground truth on the clustering of dataset X: the output of a Cluster Ensemble algorithm strictly depends on the quality of the input partitions. If many partitions provide a view of the input data that is far from the ground truth, the output may be influenced by those views. For this reason, to evaluate the quality of a Clustering Ensemble algorithm, the most popular metrics in literature don't compare the quality of the result with the ground truth, but they are usually a measure of closeness between the output and all of the input partitions. Assuming that a consensus of all partitions contains more information than a single one and that a consensus process is resistant to errors, we expect that the consensus partition performs better than any partition in the ensemble.
Now that we highlighted the main characteristics of a Cluster Ensemble algorithm, we can enter into the details of how a Cluster Ensemble algorithm works.

3.1. Ensemble generation methods

For the ensemble generation phase, we present a taxonomy of some of the methods and heuristics found in literature [7], based on the different generation strategies. It can be noted that any of these heuristics can be mixed and combined with the others to enrich the ensemble generation phase.

Homogeneous ensemble Partitions are created by running the same clustering algorithm with different sets of parameters (for example k-means, with random initialization of cluster centers).

Different-k The output of many clustering algorithms is dependent on the initial choice of the number of clusters k. To acquire ensemble diversity, partitions are created using the same or different algorithms with a specific value of k or randomly selecting k from a pre-specified interval, which is the same for every algorithm. The specific value of k for this generation technique is usually greater than the expected number of clusters.

Random subspacing/sampling An ensemble can also be achieved by using subsets of initial data to each partition. It is assumed that each clustering algorithm can provide different levels of performance for different subsets of a dataset. Additionally, data partitions can be simply obtained by projecting data onto different subspaces, choosing different subsets of features, or using data sampling techniques.

Heterogeneous ensembles heterogeneous ensembles may be exploited, where diversity is introduced by allowing each partition to be generated using different clustering algorithms. Multiple algorithms can provide different decisions on data partitions and complement each other to assure a better end-result. Each partition is generated independently from the others and may represent a completely different view on the data. An example of this generation technique may be, applying both k-means with $k = 5$ and Agglomerative Hierarchical Clustering on the same dataset to generate two completely different partitions.

Stochastic partition generation Each observed partition π_g in the ensemble Π is generated by two transformations of the true partition π_{true}, which is synthetically generated and known a priori:

1. Noise, $\pi' = F(\pi_{true})$: a random noise with probability $(1-p)$ is applied to a cluster label $l(x_i)$ of each object x_i. The value $l(x_i)$ is replaced by a new random label $t \in \{1, \dots, k\}, \forall t \neq l(x_i)$ with probability q.
   A data point keeps a correct label $l(x_i)$ with probability p, and changes to an incorrect label with probability $(1-p)$. All incorrect labels have the same probability:
   $$q = \frac{1-p}{k-1}$$
   This step generates a noisy version $l'(X)$ of the true partition π_{true}, called $\pi' = \{l'(x_1), l'(x_2), \dots, l'(x_N)\}$
2. Label permutation, $\pi'' = T(\pi')$: the label permutation $T = \{\sigma(1), \sigma(2), \dots, \sigma(k)\}$ is drawn from the set of all possible permutations of k labels with uniform probability. The partition $\pi'' = l''(X) = T(l'(X))$ becomes now a member of the ensemble and $\pi^{''} = \{l''(x_1), l''(x_2), \dots, l''(x_N)\}$.
   π_1, the first partition in the cluster ensemble, takes the value of π''. The above process is repeated with different realizations of $F(.)$ and $T(.)$ to generate other partitions π_i in the ensemble

The observed ensemble Π becomes then the collection of M random partitions: $\Pi = \{\pi_1, \dots, \pi_M\}$. This method of partition generation works best for solving the label correspondence problem when using the weighted bipartite graph formulation [45].

3.2. Ensemble combination methods

A variety of algorithms to generate the consensus partition have been developed in literature. Each algorithm is heavily reliant on the context of application: in general, a Cluster Ensemble algorithm has the goal of generating the consensus partition, but how it is generated depends on multiple factors, like the category of ensemble generation. Two main taxonomies of algorithms have been proposed:

- Vegapons et al. [48] propose two main categories of algorithms, those based on object co-occurrence and those who follow a Median partition approach. In the first category, the idea is to determine which cluster label must be associated to each object in the consensus partition by analyzing how many times an object belongs to one cluster or how many times two objects belong together to the same cluster. In the second category, the consensus partition is obtained by solving the problem of finding the median partition as formulated in Equation 2.1. The median partition is defined as the partition that maximizes the similarity with all partitions in the cluster ensemble.

- De Amorim [12] and Boongoen [7] divide the types of algorithms in four main categories: Direct approach, feature-based, pairwise similarity-based, graph-based.
  The first family is characterized by the use of a combination strategy (i.e. voting), and it may require the knowledge of the number of clusters a priori. Since the cluster labels in each partition are arbitrary, steps that find 'label correspondence' and relabel each partition w.r.t to a reference partition are necessary to implement a voting mechanism. Most methods in this category require the number of clusters in each partition to be k, i.e., $k_g = k, \quad g = 1, \ldots, M$.
  The second family of methods is based on the categorical data available in the dataset. Feature-based methods cluster data points using the nominal information that is originally obtained from an ensemble, without searching for correspondence amongst labels or relabeling.
  The third family is based principally on the pairwise similarity amongst data points: they compute a specific similarity matrix that expresses the information of how often two points are grouped together in the same cluster and then compute the consensus partition from this matrix.
  The fourth family of algorithms makes use of a graph representation to solve the cluster ensemble problem. These graphs may be generated from the co-occurrence between points, from the association between points and clusters, from the similarity between clusters or from arbitrary similarity measures.

	CO-OCCURRENCE	MEDIAN PARTITION
DIRECT APPROACH	• Relabeling And Voting • LCS	-
FEATURE-BASED	• Cluster Aggregation	• Mixture Model • Iterative Voting Consensus
PAIRWISE SIMILARITY	• Agglomerative Hierarchical Clustering • Fuzzy Ensemble Clsutering	-
GRAPH-BASED	• GCC, CSPA • HBGF • SNNC • CESHL • HGPA • WSPA,WBPA • MCLA • CTS, SRS • L-MCLA • LCE	-

Table 3.1: Map of the overlapping between the two proposed taxonomies in literature.

In Table 3.1 we show a possible interpretation of how the two taxonomies can be compared and where they overlap, to better identify and localize the context of the methods we present. Many methods from the categories presented in the taxonomy by De Amorim and Bongooen can be considered quite similar to the Co-occurrence family defined by Vegapons et al.: for example, all methods under Direct Approach, Pairwise similarity based and Graph based all make use of the information of either how a point is associated to a cluster, how clusters relate to one another, or how points relate to one another. For this reason they overlap quite well with the family of co-occurrence methods. Just a few methods from the feature-based family can be placed under the median partition approach since they search for the consensus partition either by clustering the information available in the ensemble or by working with probability, without looking how points relate to clusters or one another.

In our work we will focus mainly on some methods from the families of Direct Approach methods, mainly the ones pertaining to the subfamily of *Relabeling and Voting* and the family of Graph-based methods, since the solution we developed can be directly compared to a good number of them. In the following, we offer a description of the general rationale behind both families, and we then enter in the detail of the main algorithms known in literature from each one.

In Figure 3.1 we show a visual representation of how the methods we focused on are placed in each family: in the figure there are two sets that represent the two families

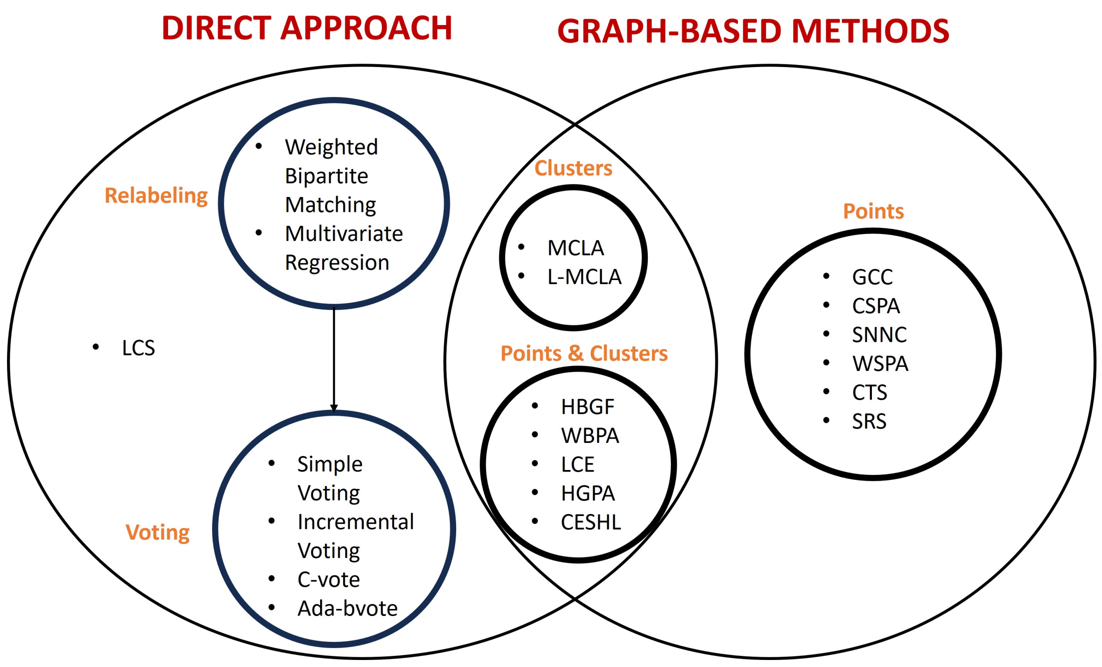


Figure 3.1: Comparison between graph-based and direct approach methods

of Direct Approach methods and Graph-based methods. Inside each family there are subsets that represent specific categories of methods. In the Direct Approach set there are the subcategories of Relabeling algorithms and Voting algorithms that together form the category of Relabeling and Voting methods: the relabeling algorithms we present are weighted bipartite matching problem solved with Hungarian [35] and Multivariate Regression [4], while for voting there are Simple Voting [45], Incremental Voting [4], C-Vote [4] and Ada-Bvote [4]. Label Correspondence Search [8] is placed outside of those two subcategories since it solves both problems at the same time without a clear separation between the two.

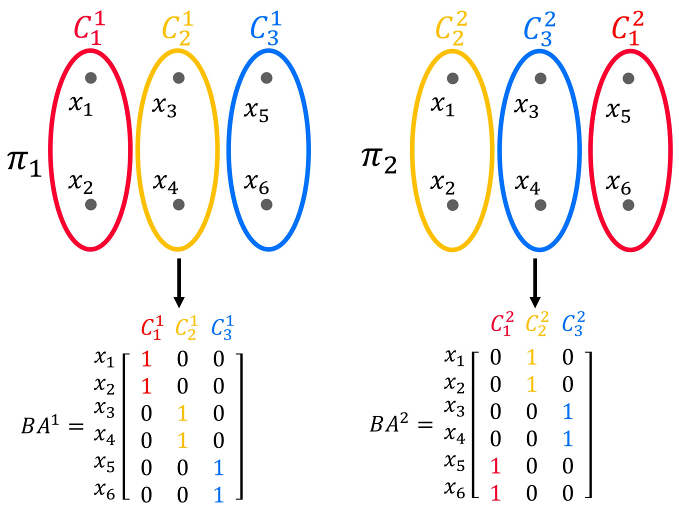
For graph-based methods there are the subcategories of methods that construct a graph from the similarity between points, those that construct it from the similarity between clusters, and those that construct it from the relationships between points and clusters. For the first subcategory we have GCC [50], CSPA [43], SNNC [3], WSPA [13], CTS [29], SRS [30]. For the second category we have MCLA [43] and L-MCLA [40]. For the third category we have HBGF [17], WBPA [13] and LCE [30], HGPA [43] and CESHL [54]. The subcategories of methods that construct the graph from the relationships between clusters and those that construct the graph from the relationships between points and clusters are placed at the intersection of the two families of methods since either they have a clear distinction between a phase in which clusters are grouped together and a phase in which points are then assigned to the closest group of clusters or because they yield a result in which points are assigned to groups of clusters, rendering them quite similar to the methods belonging to the category of relabeling and voting.

3.2.1. Relabeling and Voting methods

All the following methods are based on solving first the *labeling correspondence problem*, also known as *relabeling*, to create a matching between clusters in different partitions. After creating a correspondence between clusters in each partition, a voting function is applied to obtain the consensus partition.

Relabeling

In some cases, to compute the consensus partition it is important that all the partitions in the ensemble use the same set of labels for the clusters, and that clusters in different partitions are consistently labeled, by minimizing the number of mislabeled clusters. Relabeling algorithms strive to solve this problem by using different techniques. Relabeling consists in the process of finding a permutation of cluster labels between different partitions such that the labels in all partitions are aligned and represent the same information.



$$BA^1 = \begin{array}{c|ccc} & C_1^1 & C_2^1 & C_3^1 \\ x_1 & 1 & 0 & 0 \\ x_2 & 1 & 0 & 0 \\ x_3 & 0 & 1 & 0 \\ x_4 & 0 & 1 & 0 \\ x_5 & 0 & 0 & 1 \\ x_6 & 0 & 0 & 1 \end{array} \qquad BA^2 = \begin{array}{c|ccc} & C_1^2 & C_2^2 & C_3^2 \\ x_1 & 0 & 1 & 0 \\ x_2 & 0 & 1 & 0 \\ x_3 & 0 & 0 & 1 \\ x_4 & 0 & 0 & 1 \\ x_5 & 1 & 0 & 0 \\ x_6 & 1 & 0 & 0 \end{array}$$

Figure 3.2: Example of partitions with different cluster labels representing the same information

In Figure 3.2 we show an example of a scenario in which a relabeling algorithm may be applied. For simplicity, there are six points, grouped in three clusters, in two partitions. The clusters are color coded in such a way that clusters with the same label have the same color, even if they group different points. From the BA matrices it is evident that the same information, is placed in different columns with different colors. A relabeling algorithm should be able to identify which column in BA_1 corresponds to the same cluster in BA_2.

In the following we show some methods to solve the label correspondence problem.

Bipartite Graph To obtain a consistent labeling between partitions, the problem is solved as a maximum weight bipartite matching problem [45]. For two partitions π_g and π_h, with the same number of cluster labels $k_g = k_h = k$, a contingency matrix W is created, containing the co-occurrence of the cluster labels in the two different partitions on the same data set:

$$W_{pq} = \sum_{\forall x_i \in X} w(x_i), \quad C_p \in \pi_g \text{ and } C_q \in \pi_h$$

where

$$w(x_i) = \begin{cases} 1 & \text{if } l_g(x_i) = C_p \wedge l_h(x_i) = C_q \\ 0 & \text{otherwise} \end{cases}$$

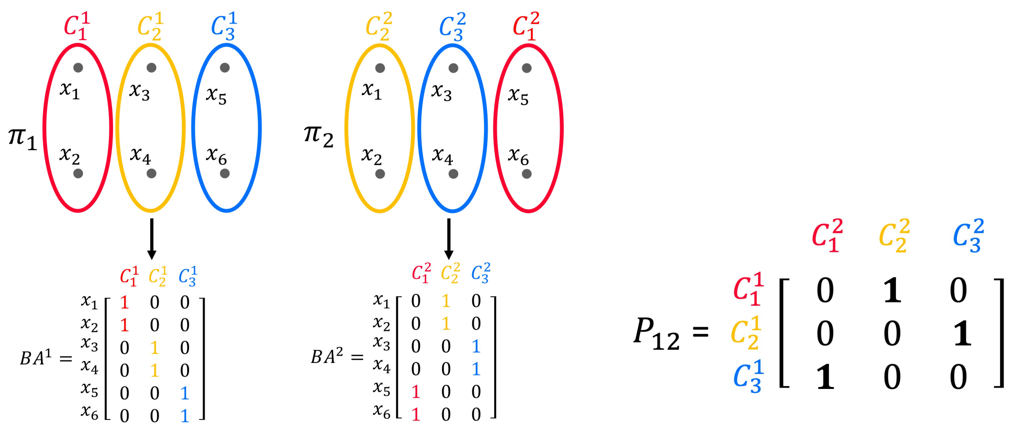
From that, the correspondences between clusters that minimize the misassigned objects are selected, solving the following maximization problem, which finds the entries of a a $k \times k$ permutation matrix P_{gh} between the clusters of partitions π_g and π_h,:

$$\max_{p_{ij}} \sum_{i=1}^{k} \sum_{j=1}^{k} W_{ij} p_{ij}$$

$$\text{s.t.} \sum_{i=1}^{k} p_{ij} = \sum_{j=1}^{k} p_{ij} = 1, \quad p_{ij} \in 0, 1$$

where W_{ij} are the contingency matrix values, and p_{ij} are the entries of the permutation matrix P_{gh} used to determine the correspondence between the two set of clusters.

One of the first and most famous algorithms that solve this problem is the Hungarian Algorithm [35]. This algorithm given an assignment cost matrix between sets of labels, returns the assignment matrix that for each row (representing an element in the first set of cluster labels) assigns a column (representing an element in the second set of cluster labels) that minimizes or maximizes the cost. This formulation was developed for the case in which all partitions in the ensemble have the same number of clusters, and it is one of its main limitations. This formulation can be easily extended to the different number of clusters scenario, without modifying how the weighted bipartite graph is generated. In this case, the Hungarian Algorithm will return a partial permutation that may associate to clusters in π_g a subset of clusters in π_h or viceversa.



C^1_1 C^1_2 C^1_3
π_1
x_1 x_2 x_3 x_4 x_5 x_6
C^2_2 C^2_3 C^2_1
π_2
$BA^1 = \begin{array}{c|ccc} & C^1_1 & C^1_2 & C^1_3 \\ x_1 & 1 & 0 & 0 \\ x_2 & 1 & 0 & 0 \\ x_3 & 0 & 1 & 0 \\ x_4 & 0 & 1 & 0 \\ x_5 & 0 & 0 & 1 \\ x_6 & 0 & 0 & 1 \end{array}$
$BA^2 = \begin{array}{c|ccc} & C^2_1 & C^2_2 & C^2_3 \\ x_1 & 0 & 1 & 0 \\ x_2 & 0 & 1 & 0 \\ x_3 & 0 & 0 & 1 \\ x_4 & 0 & 0 & 1 \\ x_5 & 1 & 0 & 0 \\ x_6 & 1 & 0 & 0 \end{array}$
$P_{12} = \begin{array}{c|ccc} & C^2_1 & C^2_2 & C^2_3 \\ C^1_1 & 0 & 1 & 0 \\ C^1_2 & 0 & 0 & 1 \\ C^1_3 & 1 & 0 & 0 \end{array}$

Figure 3.3: Example of permutation matrix resulting from relabeling

In Figure 3.3 we show the solution of the relabeling problem introduced in Figure 3.2. The matrix P_{12} is the permutation matrix found by the solution to the bipartite matching formulation. The permutation matrix simply associates each cluster of π_1 to a cluster in π_2, and $BA_2 = BA_1 P_{12}$.

When relabeling is solved using this formulation, to obtain a complete alignment between all partitions, a reference partition π_{ref} is selected and relabeling is executed for each pair (π_g, π_{ref}). In this way each partition will have the same cluster labels of the reference.

Regression Let U_{ref} be the stochastic matrix of a reference partition selected a priori. The goal is to find the optimal relabelling of stochastic matrix U_g, called $\theta(U_g)$, that is the closest to U_{ref} with respect to a distance function d. The problem is formulated as a regression problem with multiple-input and multiple response variables [4]. In this way the problem of finding $\theta(U_g)$ becomes

$$\min_{\theta(U_g)} L_i(U_{ref}, \theta(U_g)$$

Where L_i is the pairwise relabeling loss. For each couple (U_g, U_{ref}), a training dataset $\mathscr{D}_g$ is generated, containing the vectors $\{([U_g]_j, [U_{ref}]_j)\}_{j=1}^N$, where $[U_g]_j$ is the jth row (with size k_i) of U_g, and $[U_{ref}]_j$ is the jth row (with size k_{ref}) of U_{ref} and represents the target vector for the regression. Assuming that it exists a linear model for each output variable, $\theta(U_g)$ can be written as $\theta(U_g) = V_g = U_g P_{gref}$, where P_{gref} is a $k_g \times k_{ref}$ matrix of weights, representing the utility matching between the clusters of partition π_g and π_{ref}. For the estimation the mean squared error is used as loss function:

$$L_g(U_{ref}, \theta(U_g) = MSE(U_{ref}, \theta(U_g) = \frac{1}{N}\mathrm{tr}[(U_{ref} - U_g P_{gref})^T (U_{ref} - U_g P_{gref})]$$

The estimated solution is $\hat{V}_g = U_g \hat{P}_{gref}$, where:

$$\hat{P}_{gref} = ({U_g}^T U_g)^{-1} {U_g}^T U_{ref}$$

Label Correspondence Search In this algorithm the correspondence between clusters is found creating an Agreement Matrix G, from which k^* sets of clusters $C^*_m, m \in 1, \ldots, k^*$ are selected, called metaclusters, where each cluster in the set corresponds to the same label [8].

The agreement $G(C_p, C_q)$ between two cluster C_p, C_q in the partitions π_g, π_h is defined as

$$G(C_p, C_q) = (U_p)^T \cdot U_q$$

where U_t is the stochastic vector of the t-th cluster.

Then the problem of finding the optimal metaclusters is formulated as a maximization of a function F^Λ, defined as following for each partition:

$$F^\Lambda = \sum_{m=1}^{k^*} \sum_{t=1}^{k_g} \Lambda(C_t, C^*_m) \cdot S(C_t, C^*_m)$$

subjected to

$$\sum_{m=1}^{k^*} \Lambda(C_t, C^*_m) = 1, \forall t \in \{1, \ldots, k_g\}$$

where

$$\Lambda(C_t, C^*_m) = \begin{cases} 1 & \text{if } C_t \in C^*_m \\ 0 & \text{otherwise} \end{cases}$$

$$S(C_t, C^*_m) = \begin{cases} \frac{1}{|C^*_m|} \sum\limits_{\forall cl \in C^*_m, cl \neq C_t} G(C_t, cl) & \text{if } \Lambda(C_t, C^*_m) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Λ is a binary association matrix between the total number of clusters in the ensemble and the k^* metaclusters, and S defines a score assigned to each cluster inside a metacluster. Optionally, the objective function F^Λ can be maximized also with respect to the additional constraint of:

$$\sum_{t=1}^{k_g} \Lambda(C_t, C_m^*) = 1, \quad \forall m \in \{1, \dots, k^*\}$$

The result of the optimization problem is the matrix $F \in \mathbb{R}^{N \times k^*}$, where each element F_{im} defines the association between point x_i and metacluster C_m^*.

Voting

After relabeling, a voting step is executed. Each relabeled partition represents a vote for a cluster label for all points in the dataset, and the consensus partition is then created by collecting all the proposed labels and selecting the most suitable one for each point according to a variety of selection criteria, depending on the method.

Simple Voting When using the weighted bipartite matching formulation, after retrieving all relabelings, plurality voting is used to choose the best label assigned to each point. For the consensus it is just required to select the most assigned label, and it is not required that the majority of the partitions agree on the result. This is done by selecting for each point the most frequent label selected among all partitions:

$$l^*(x_i) = \arg\max_t V_t(x_i)$$

Where $V_t(x_i)$ is the number of times the tth cluster is associated to the point x_i

Label Correspondence Search The same approach as Simple Voting is used when applied to label correspondence search, where plurality voting is used to select the most appropriate label [8]. The final label of each is point is:

$$l^*(x_i) = \arg\max F_{im}, \quad m = 1, \dots, k^*$$

Iterative Voting Obtaining a complete correspondence between cluster labels is not always achievable, so some methods employ a cumulative voting process, where the consensus partition is updated in iterations. In these algorithms consistency is guaranteed with respect to the consensus partition in the previous iteration. We present some of them in the following:

- **Basic method [4]** After selecting an arbitrary ordering for the partitions, each iteration step a matrix V^g is computed by applying relabeling to the couple (V^{g-1}, BA_g) using linear regression, where V^{g-1} represents the merging of the previous g partitions, combined using their Binary Association matrix:

$$\begin{aligned} &V^g = V^{g-1} + BA_g * P_{V^{g-1}g} \\ &V^1 = BA_1 \\ &g \in \{1, \ldots, M\} \end{aligned}$$

The consensus labels are then selected by plurality voting:

$$l^*(x_i) = \arg\max_t V_t^M(x_i)$$

where V_t^M is the tth column of the final V^M matrix.

- **Bvote [4]** After selecting an arbitrary ordering for the partitions and a reference (i.$\pi_{ref} = \pi_1$), at each step $i \in 2, \ldots, M$, the optimal relabelling $V^i = U_g \cdot P_{iref}$ with respect to the stochastic matrix U_{ref} is found using linear regression. Then U_{ref} is recomputed as the weighted average of U_{ref} in the previous step and V^i. To enhance performance, bVote can be run several times with different partitions orderings and keeping the solution with lowest MSE.

- **Ada-cvote [4]** The iterative part is the same as Bvote, but the two main differences are in the partition ordering and the relabeling problem formulation. The order is created by taking the partitions in decreasing order of the mutual information shared between π_i and X $I(\pi_i, X)$.

  In this way the first partition to be selected will be

  $$\pi_1 = \arg\max_{\pi_i \in \Pi} I(\pi_i, X)$$

  At each step V_i is computed with the linear regression approach.

  After finding the consensus partition π^* in the previous two algorithms, an agglomerative algorithm based on the information bottleneck method developed by the same authors, is applied to obtain a refined solution $\hat{\pi}$ with $\hat{k}$ clusters.

Weighted Voting It is possible to increase precision to the voting phase by adding extra information derived from external indices or a priori information on the data and partitions, which takes the form of weights. The critical step in the weighting process is the selection of a metric or a property which represents the quality of the single vote with

respect to the others. Weighting can be applied on different levels (weights on the data, weights on clusters, weights on partitions), and for cluster ensemble methods based on a relabeling approach, two methods based on weighting partitions are available.
The method developed by Zhou and Tang [55] utilizes as weight for a partition the inverse of the average NMI:

$$w_g = \frac{(M-1)}{\sum_{\pi_g \neq \pi_i} NMI(\pi_i, \pi_g)}$$

such that

$$\sum_{g=1}^{M} \frac{w_g}{Z} = 1$$

where Z is a constant used to normalize the weights.

This is done because the less information partition π_g has in common with other partitions in the ensemble, the more original and relevant information it provides to the voting process, so it must get a higher weight.
The method developed by Zhang et al. [52] makes use instead of the NMI between each partition of the ensemble and a reference partition:

$$w_g = NMI(\pi_g, \pi_{ref})$$

This method prioritizes the partitions which are more similar to the reference we choose, while it discards dissimilar partitions on the assumption that a higher difference means lower quality.

Advantages and disadvantages

The relabeling and voting methods we presented have the advantage of being quite easy to understand and implement, as it is really intuitive to see that these methods look for where each point should belong by looking at every (matched) cluster in the ensemble. They also have the advantage of having a computational cost not heavily dependent on the number of points, which in real applications is usually a high number. Their main disadvantages are the fact that the label correspondence problem can be computationally expensive, as algorithms such as Hungarian have computational cost of $O(k^3)$, but as long as the number of clusters is relatively low, this cost can be contained. Another disadvantage is that the quality of the solution is heavily influenced either by the chosen reference partition or chosen ordering in case of iterative voting, and we could say then that the solution is biased towards those partitions or orderings.

3.2.2. Graph-based methods

All these algorithms are based on a graph representation to solve the Cluster Ensemble problem. The typical graph-based algorithm will usually require at least two mandatory steps:

- Construction of a weighted undirected graph $G = (V, W)$ from the input data
- Partitioning of the graph in k^* parts using a well-known graph partitioning technique. The final number of clusters is assumed known a priori

These types of algorithms have in common only the fact they use a graph representation. The output of the graph partitioning, for this reason, will have a different meaning depending on what the graph represents. For example a graph can represent either the association between data points and clusters, or it can represent an agglomeration of only clusters.

Graph Generation

How the graph is generated from the input data greatly differs between algorithms. Not only different matrices or criteria can be chosen to construct a weighted graph, but also the type of graph that is partitioned can be of a different type depending on the algorithm. There are three main categories of graphs that are generated by graph-based cluster ensemble algorithm: those that use the graph to model the similarity between clusters, those that use it to model the similarity between points, and those that model the relationships between clusters and points.

Similarity between clusters All these methods generate a graph from a matrix or a measure that holds information between pairs of clusters. These methods, by looking at the similarities between clusters, solve a problem that is akin to relabeling, since they group together the clusters in meta-clusters that represent those clusters that hold the same information. All of them then use a voting technique to decide which point belongs to which meta-cluster. For this reason, these methods are very similar in their procedure to Relabeling and Voting methods.
The main representatives of this family of methods are MCLA [43] and L-MCLA [40], which generate the graph from the Jaccard similarity between pairs of clusters.

Similarity between points All these methods generate a graph from a matrix or a measure that holds information between pairs of data points. The resulting graph will be partitioned and return directly how the data points should be grouped together in the

consensus partition.
The main representatives of this family of methods are:

- GCC [50], CSPA [43] and SNNC [3], which generate the graph from the Co-association matrix between data points, which measures how often a point x_i figures in the same cluster as a point x_j
- WSPA [13], which generates the graph by measuring the distance between points and clusters, and calculating the similarity between points as the cosine similarity between these distances
- CTS [29] and SRS [29] which start from the graph created from the Jaccard similarity between pairs of clusters and expanding it using the notion of connected weighted triple, to create a matrix of similarities between data points

Relationship between clusters and points These algorithms generate a graph from the BA matrix, which holds the information regarding which points belong to which clusters. This graph usually contains a representation of both points and clusters, and when partitioned points will be assigned to group of clusters, rendering this family of algorithms similar to Relabeling and Voting algorithms.
The main representatives of this family of methods are:

- HBGF [17], which generates a bipartite graph from the Binary Cluster-Association matrix
- WBPA [13] and LCE [30], which generate a bipartite graph by either substituting or enriching the information given by the BA matrix with the one obtained from a similarity measure between points and clusters
- HGPA [43], which uses the BA matrix to generate a hypergraph from the ensemble
- CESHL [54], which from the BA matrix dynamically learns a hypergraph

Graph Partitioning

Beside of the type of graph used, the graph-based cluster ensemble algorithms also differ in the kind of partitioning algorithm used to generate the solution in the second step.

METIS and HMETIS METIS [33] is the most popular partitioning algorithm in literature, it is a multi-level k-way graph partitioning algorithm, which is used to partition the graph G into k clusters of same size. METIS consists of three phases:

1. coarsening phase, in which the size of the graph is decreased gradually
2. initial partitioning phase, in which a k-way partition of the smaller graph is computed
3. uncoarsening phase, in which the partitioning is refined in steps by projecting it in larger graphs

METIS is used by the vast majority of graph-based cluster ensemble algorithms: CSPA [43], SNNC [3], MCLA [43], HBGA [17], WSPA [13], WBPA [13], CTS [29] and SRS [29] all make use of it.
A variation of METIS for hypergraphs has also been developed, called HMETIS [34] and is mainly used by HGPA [43].

Spectral graph partitioning Spectral graph partitioning [38] is the second most popular partitioning algorithm used by these methods, in which the second eigenvector of a graph's Laplacian is used to define a semi-optimal cut.
This algorithm is used by HBGF [17], WBPA [13] and LCE [30].

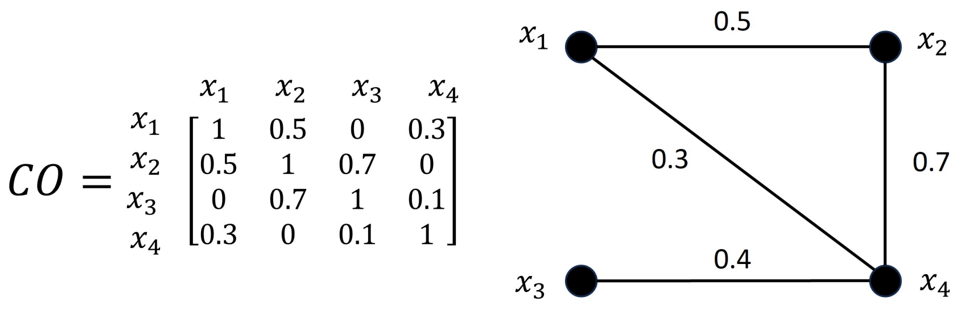
Normalized cut The normalized cut [41] is an algorithm employed by GCC [50] and L-MCLA [40]. The normalized cut criterion measures both the total dissimilarity between the different clusters as well as the total similarity within a cluster.
This algorithm works by bipartitioning recursively the graph until convergence.

In the next part we enter more in detail with the explanation of different graph-based algorithms, showing for each one the differences in the type of graph generated and type of partitioning used.

Similarity between points

All these algorithms have in common the fact that they generate a graph from a measure of similarity between couples of points, that measures if two points should be grouped together in a cluster.

Graph-based Consensus Clustering (GCC) and Cluster-based Similarity Partitioning Algorithm (CSPA) GCC [50] and CSPA [43] share the same basic principle to construct a weighted graph $G = (V, W)$ from the ensemble.
They both take as input the co-association matrix CO that can be generated from the ensemble: this matrix holds the information on how often two points x_i, x_j figure together in the same cluster on average in the whole ensemble. The entry $CO(x_i, x_j)$ of the



$CO = \begin{array}{c|cccc} & x_1 & x_2 & x_3 & x_4 \\ \hline x_1 & 1 & 0.5 & 0 & 0.3 \\ x_2 & 0.5 & 1 & 0.7 & 0 \\ x_3 & 0 & 0.7 & 1 & 0.1 \\ x_4 & 0.3 & 0 & 0.1 & 1 \end{array}$
x_1
x_2
x_3
x_4
0.5
0.3
0.7
0.4

Figure 3.4: Example of the graph created by GCC and CSPA.

co-association matrix is defined as

$$CO(x_i, x_j) = \frac{1}{M} \sum_{g=1}^{M} s^g(x_i, x_j)$$

where $s : X \times X \longrightarrow 0, 1$ is a similarity function defined as such for each partition:

$$s(x_i, x_j) = \begin{cases} 1 & \text{if } l(x_i) = l(x_j) \\ 0 & otherwise \end{cases}$$

In the graph generated by these two methods, an entry of each node $v_i \in V$ corresponds to a data point $x_i \in X$. Each weight $w_{ij} \in W$, connecting nodes $v_i, v_j \in V$ has value taken from the corresponding entry in the CO matrix $CO(x_i, x_j)$.
In Figure 3.4 we show an example of how this works: given an example dataset of 4 points, each one of them is represented as a node in the graph, and the value from the co-association matrix between two points is used as a label for the edge between the nodes representing those two points. Self-loops are ignored the graph construction.

After constructing the graph, both methods find the final partition π^* by finding a k^*-way cut. GCC does this by employing the normalized cut algorithm, while CSPA does this by using METIS software package.

Shared Nearest Neighbours-Based Combiner (SNNC) Like GCC and CSPA, SNNC [3] makes use of the CO matrix to construct the graph.
The difference introduced by SNNC is a threshold μ: for any $x_i, x_j \in X$, if $CO(x_i, x_j) > \mu$, then the entry remains unchanged, otherwise it is changed to 0.
After this change, each data point x_i is associated to a nearest neighbour set $N_{x_i} \subset X$, composed by each data point x_j such that $CO(x_i, x_j) > 0$.
A weighted graph $G = (V, W)$ is then constructed, where V is a set of weighted vertices

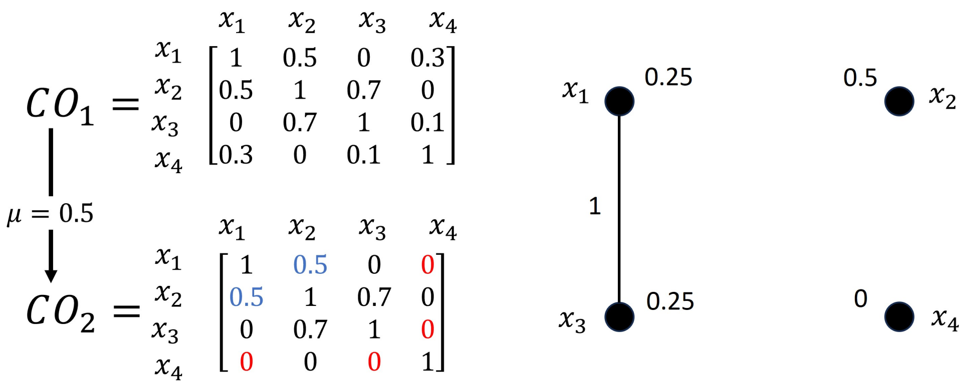


Figure 3.5: Example of the graph created by SNNC.

and W a set of weighted edges: each vertex has weight $v_i = \frac{|N_{x_i}|}{N}$, and each edge w_{ij} connecting vertices $v_i, v_j \in V$ has weight $w_{ij} = 2 \times \frac{|N_{x_i} \cap N_{x_j}|}{|N_{x_i}| + |N_{x_j}|}$.
In Figure 3.5 we show an example of this process on the same example dataset as Figure 3.4: by setting as threshold $\mu = 0.5$, a couple of entries are set to 0, for example $CO(x_1, x_4)$ and $CO(x_3, x_4)$. The neighbour sets of the example are the following: $N_{x_1} = \{x_2\}, N_{x_2} = \{x_1, x_3\}, N_{x_3} = \{x_2\}, N_{x_4} = \emptyset$. The graph is then generated by looking at the cardinalities of neighbour sets.
The final partition π^* is obtained by partitioning in k^* parts the graph using METIS.

Weighted Similarity Partitioning Algorithm (WSPA) This method [13] creates a matrix very similar to BA, called WDM, from which the final clustering is obtained. Each entry WDM_{ij} can be interpreted as a closeness between data point x_i and the cluster $C_j \in \pi_g$. The value of each WDM_{ij} is given by:

$$WDM_{ij} = \frac{D(x_i) - d(x_i, \overline{C_j}) + 1}{k_g D(x_i) + k_g - \sum_{t=1}^{kg} d(x_i, \overline{C_t})}$$

where $d(x_i, \overline{C_j})$ is the distance between data point x_i and $\overline{C_j}$, the centroid of cluster C_j, and $D(x_i)$ is defined as the maximum distance between point x_i and a cluster in π_g,

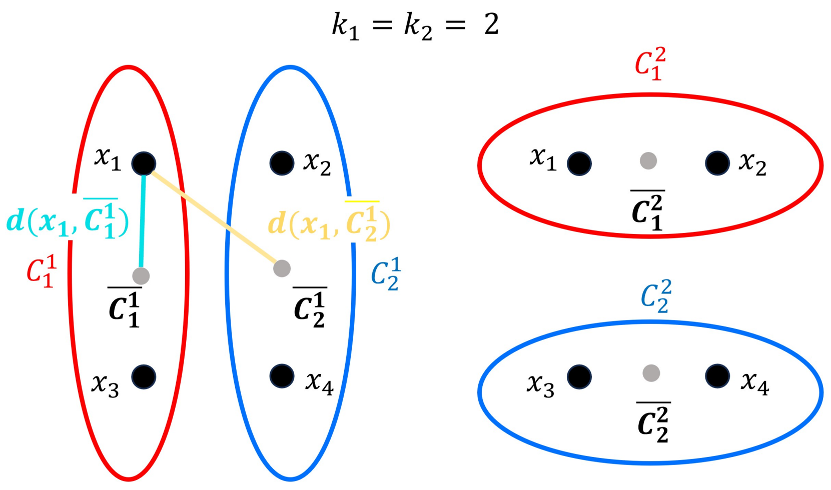
$$D(x_i) = \max_{\forall C_j \in \pi_g} d(x_i, \overline{C_j})$$

.

The distance $d(x_i, \overline{C_j})$, with $w_{C_{j,s}} \in [0,1]$ being the weight of the sth attribute that is specific to the cluster $C_j \in \pi_g$, $x_{i,s}$ being the value of the sth attribute of data point x_i , and $\overline{C_{j,s}}$ being the sth attribute value of the cluster centre $\overline{C_j}$, can be defined as:

$$d(x_i, \overline{C_j}) = \sqrt{\sum_{s=1}^{D} w_{C_{j,s}} (x_{i,s} - \overline{C_{j,s}})^2}$$

$C_j \in \pi_g$, subject to $\sum_{s=1}^{D} w_{C_{j,s}} = 1$.



$$\boldsymbol{WDM}(x_1, C_1^1) = \frac{d(x_1, \overline{C_2^1}) - d(x_1, \overline{C_1^1}) + 1}{2 \cdot d(x_1, \overline{C_2^1}) + 2 - \left(d(x_1, \overline{C_2^1}) + d(x_1, \overline{C_1^1}) \right)}$$

Figure 3.6: Example of the WDM measure created by WSPA.

In Figure 3.6 is shown a simple example of how the WDM measure is computed. For the sake of simplicity, we have two clusters with two respective cluster centers, and we want to compute the WDM measure between point x_1 and cluster C_1^1 in the first partition. Making the assumption that the distance between point x_1 and cluster C_2^1 is the biggest in the whole partition, $D(x_1)$ will return $d(x_1, \overline{C_2^1})$, and WDM_{11} will be the one shown in the figure.

The set of cluster-specific weights is specifically obtained using a soft-subspace clustering technique, the most known one being Locally Adaptive Clustering (LAC).

Once the WDM matrix is obtained, a vector P_i is defined, which is a vector of all the entries in the WDM matrix which correspond to data point x_i and all the clusters in π_g.

$$P_i = \left(WDM_{i1}, \ldots, WDM_{ik_g} \right).$$

These vectors are then used to define a $N \times N$ similarity matrix S. Each entry S_{ij}, which represents the similarity between points x_i, x_j in partition π_g, is defined as the cosine similarity between the vectors P_i, P_j associated to the two points:

$$S_{ij} = \frac{P_i P_j}{||P_i||||P_j||}.$$

The overall similarity for all points in the ensemble is condensed in the matrix S, defined simply as the normalized sum of all the M similarity matrices:

$$S = \frac{1}{M} \sum_{g=1}^{M} S^g$$

This similarity matrix is then converted into a weighted graph, and then partitioned in k^* clusters using METIS.

Connected-Triple Similarity (CTS) and SimRank-based Similarity (SRS) Both CTS and SRS algorithms have been proposed to improve on the performance of CSPA, which has problems in handling a large number of unknown (0 entries in the CO matrix) relations between data points. In this case the CO matrix can provide only a small number of pairwise similarities between data points, which brings the need to add additional information about relations between clusters in an ensemble.

- **CTS** In CTS [29], similarity information between clusters is extracted from a cluster-level graph $G = (V, W)$ not unlike the one constructed in MCLA, where each $v_i \in V$ represents a cluster $C_i \in \Pi$, and each weight $w_{ij} \in W$ for the edge between $v_i, v_j \in V$ is computed using the binary Jaccard similarity between the two clusters associated to the two vertices, such that $w_{ij} = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$.
  Given this representation as a base, additional information about similarity between clusters is extracted using the Weighted Connected-Triples (WCT) measure: for each pair of clusters $C_p, C_q \in V$, it is counted how many Connected-Triples they are part of.
  A triple, $Triple = (V, W)$, is a subgraph of G containing three vertices
  $V_{Triple} = \{C_p, C_q, C_o\} \subset V$ and two non-zero edges $W_{Triple} = \{w_{po}, w_{qo}\} \subset W$, with $w_{pq} = 0$.
  For each pair of clusters $C_p, C_q \in V$, the WCT measure with respect to each triple with $C_o \in V$ is measured as $WCT^o_{pq} = min(w_{po}, w_{qo})$, where $w_{po}, w_{qo} \in W$ are weights of the edges connecting clusters C_p and C_o, and clusters C_q and C_o.

All triples $(1...\alpha)$ between clusters C_p and C_q are then counted as follows:

$$WCT_{pq} = \sum_{o=1}^{\alpha} WCT_{pq}^{o}.$$

The similarity between two clusters C_p and C_q is then estimated as:

$$Sim_{WCT}(C_p, C_q) = \frac{WCT_{pq}}{WCT_{max}} \times DC,$$

where WCT_{max} is the maximum WCT_{pq} value of any two clusters C_p, C_q in the ensemble, and $DC \in [0, 1]$ is a constant decay factor.

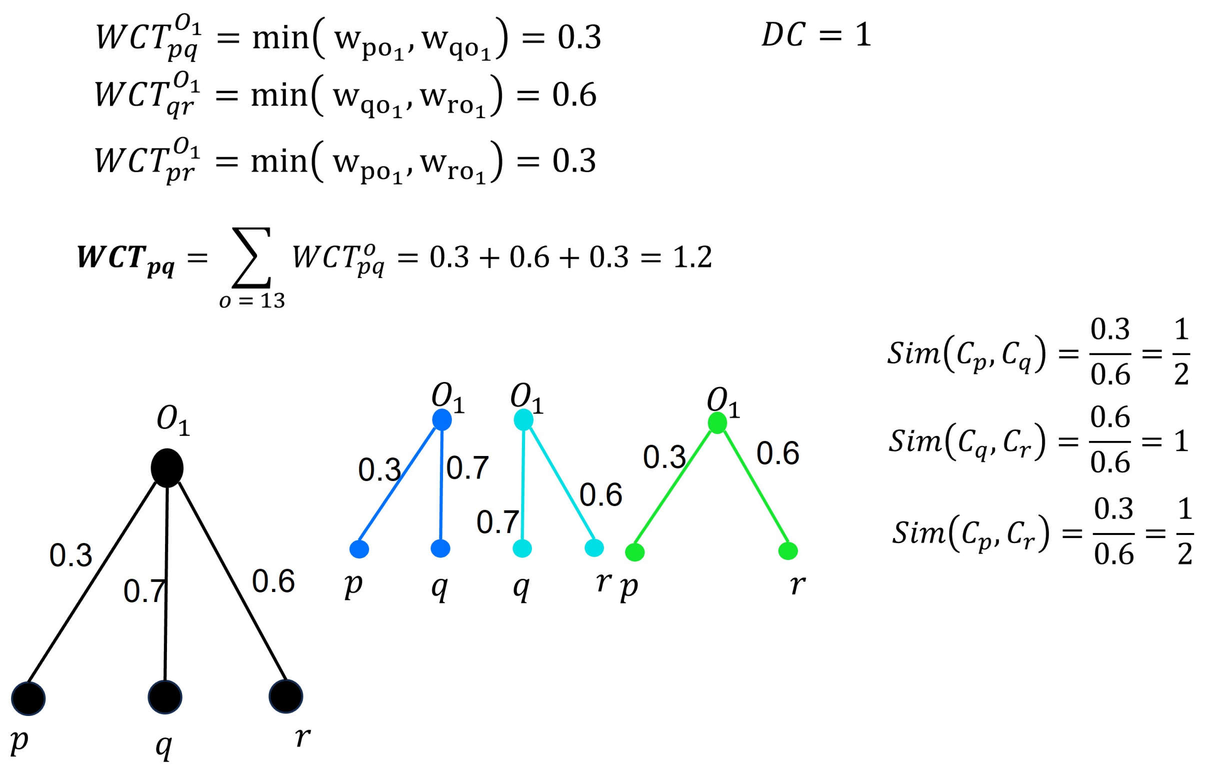

$WCT_{pq}^{O_1} = \min(w_{po_1}, w_{qo_1}) = 0.3$
$DC = 1$
$WCT_{qr}^{O_1} = \min(w_{qo_1}, w_{ro_1}) = 0.6$
$WCT_{pr}^{O_1} = \min(w_{po_1}, w_{ro_1}) = 0.3$
$\boldsymbol{WCT_{pq}} = \sum_{o=13} WCT_{pq}^{o} = 0.3 + 0.6 + 0.3 = 1.2$
$Sim(C_p, C_q) = \frac{0.3}{0.6} = \frac{1}{2}$
$Sim(C_q, C_r) = \frac{0.6}{0.6} = 1$
$Sim(C_p, C_r) = \frac{0.3}{0.6} = \frac{1}{2}$
O_1
0.3
0.7
0.6
p
q
r

Figure 3.7: Example of a Weighted Connected Triple.

In Figure 3.7 we provide an example of triples and how the WCT and Sim measures work: given three nodes C_p, C_q, C_o, the triple of C_o is just the part of the graph that connects node C_o to nodes C_p and C_q. In the example we show a simple graph composed by four nodes, $C_p, C_q, C_r C_{o1},$, connected in such a way that there are three triples in it, $\{C_p, C_q, C_{o1}\}$, $\{C_q, C_r, C_{o1}\}$ and $\{C_p, C_r, C_{o1}\}$, highligted in the image by the separated sub-graphs of different colors. Given these three triples all the respective WCT measures between all pairs of clusters that are connected to O_1

are computed, and then from these we are able to calculate the similarities between all pairs of clusters.

After having generated the similarities for all clusters in the ensemble, a procedure really similar to the one presented in WSPA is proposed: for each partition π_g, the similarity between data points x_i, x_j is estimated as:

$$S_{ij} = \begin{cases} 1 & \text{if } l(x_i) = l(x_j) \\ Sim_{WCT}(C_{l(x_i)}, C_{l(x_j)}) & otherwise \end{cases}$$

An overall $N \times N$ similarity matrix S_{CTS} is then calculated as the average of all the S^g matrices:

$$S_{CTS} = \frac{1}{M} \sum_{g=1}^{M} S^g.$$

Just like CSPA, the similarity matrix S_{CTS} is transformed into a weighted graph, from which the final partition π^* is obtained using METIS by partitioning the graph in k^* parts.

- **SRS**

SRS [29] makes use too of a network of clusters to reveal information about unknown relations, but it does so by employing a bipartite graph formulation identical to the one presented in HBGF, and on top of that it adds a similarity measure.
The bipartite graph is defined as $G = (V, W)$, where $V = V^X \cup V^C$ is the union of the set of vertices associated to data points V^X and clusters V^C, with $V^X = \{x_i | x_i \in X\}$ and $V^C = \{C_j | C_j \in \Pi\}$.
The weight $w_{ij} \in W$ between vertices $v_i, v_j \in V$ takes different values depending on which subset of V the vertices belong to:

$$w_{ij} = \begin{cases} 0 & \text{if } v_i \in V^X \wedge v_j \in V^X \vee v_i \in V^C \wedge v_j \in V^C \\ [BA_g]_{ij} & \text{if } v_i \in V^X \wedge v_j \in V^C \end{cases}$$

Two matrices are defined, namely $S^{SRS} \in \mathbb{R}^{N \times N}$ and $S'^{SRS} \in \mathbb{R}^{k_{tot} \times k_{tot}}$, which represent respectively pairwise similarities between points and pairwise similarities between clusters, with their values taken as such:

$$S_{ij}^{SRS} = \begin{cases} 1 & \text{if } x_i = x_j \\ \frac{DC}{|N_{x_i}||N_{x_j}|} \sum\limits_{\forall C_p \in N_{x_i}} \sum\limits_{\forall C_q \in N_{x_j}} S'^{SRS}_{pq} & otherwise \end{cases}$$

where $DC \in [0,1]$ is a constant decay factor and $N_{x_i} \subset V^C$ is the set of cluster vertices connected to the data point vertex $x_i \in V^X$, i.e $w_{ij} = 1, \forall C_j \in N_{x_i}$,

$$S'^{SRS}_{pq} = \begin{cases} 1 & \text{if } C_p = C_q \\ \frac{DC}{|N_{C_p}||N_{C_q}|} \sum\limits_{\forall x_i \in N_{C_p}} \sum\limits_{\forall x_j \in N_{C_q}} S^{SRS}_{ij} & otherwise \end{cases}$$

where $N_{C_p} \subset V^X$ is the set of data point vertices connected to the cluster vertex $C_p \in V^C$, i.e $w_{ip} = 1, \forall x_i \in N_{C_p}$.
Given this formulation, both matrices are obtained through an iterative refinement process:

The estimate at iteration $r+1$ is given by:

$$\begin{aligned} S^{SRS}_{ij_{r+1}} &= \frac{DC}{|N_{x_i}||N_{x_j}|} \sum_{C_p \in N_{x_i}} \sum_{C_q \in N_{x_j}} S'^{SRS}_{pq_r} \\ S'^{SRS}_{pq_{r+1}} &= \frac{DC}{|N_{C_p}||N_{C_q}|} \sum_{x_i \in N_{C_p}} \sum_{x_j \in N_{C_q}} S^{SRS}_{ij_r} \end{aligned} .$$

where $S^{SRS}_{ij_r}$ and $S'^{SRS}_{pq_r}$ are similarity degrees between points x_i, x_j or clusters C_p, C_q at the rth iteration. At the initialization phase, the two matrices are initialize as such:

$$S^{SRS}_{ij_0} = \begin{cases} 1 & \text{if } x_i = x_j \\ 0 & otherwise \end{cases}$$

and

$$S'^{SRS}_{pq_0} = \begin{cases} 1 & \text{if } C_p = C_q \\ 0 & otherwise \end{cases}$$

In Figure 3.8 we show how SRS works, in a simple manner: in red we show the bipartite graph from which the S^{SRS} matrices are computed, similar to the one shown in Figure 3.10, in yellow are shown the edges that are generated from the S^{SRS} matrix, which are similarities between points, while in green are shown the edges that are generated from the S'^{SRS} matrix, which are similarities between clusters.

Once the final similarity matrix between data points S^{SRS} is obtained, it is transformed into a weighted graph just like the other methods that make use of a similarity matrix between points, and the final partition π^* is obtained by employing METIS to partition the graph in $k^* parts$.

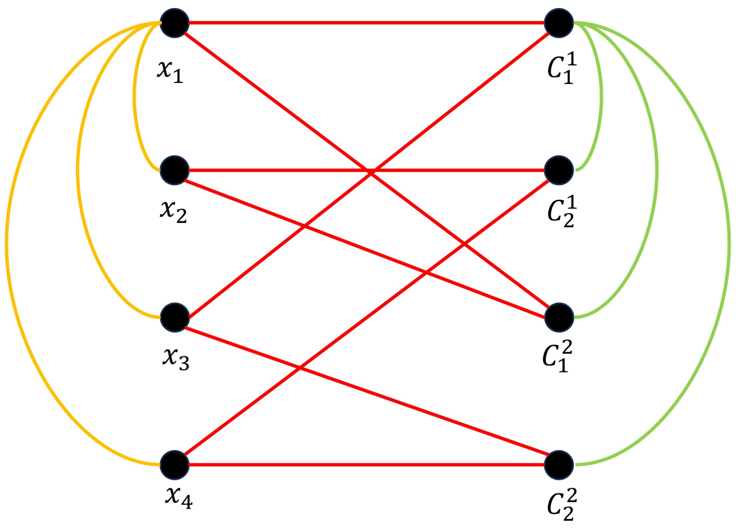


Figure 3.8: Example of the bipartite graph created by SRS.

Similarity between clusters

These algorithms generate a graph from a measure of similarity between pairs of clusters, and solve the problem of grouping together similar clusters, something akin to relabeling and voting algorithms.

Meta-Clustering Algorithm (MCLA) In MCLA [43], a graph $G = (V, W)$ is constructed at cluster-level: each $v_i \in V$ represents a cluster $C_i \in \Pi$, and each weight $w_{ij} \in W$ for the edge between $v_i, v_j \in V$ is computed using the binary Jaccard similarity between the two clusters associated to the two vertices, such that

$$w_{ij} = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$$

After the construction of the graph, METIS is employed to partition the graph in k meta-clusters.

In Figure 3.9 it can be seen how this process works: given an example ensemble with two partitions π_1, π_2, with $k_1 = k_2 = 2$, we show in red and blue the clusters of each partition. The graph on which MCLA operates on is created by taking as node each individual cluster in the ensemble and putting as label for the edge between their associated nodes the Jaccard similarity measure, as shown in the figure by the numbered weight on each edge.

Each data point has a certain association degree to each meta-cluster, estimated from the number of original clusters in the meta-cluster to which the point belonged in the

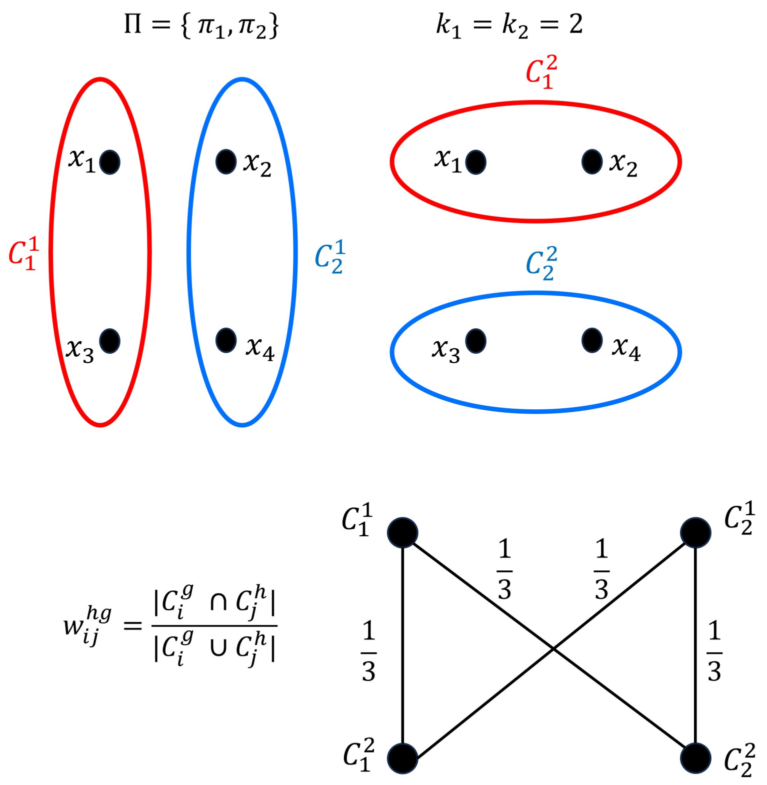


Figure 3.9: Example of the graph created by MCLA

ensemble.
The final partition π^* is obtained then by associating each data point to the meta-cluster which has the highest association degree. This procedure can be seen as a way to do relabeling between clusters without employing the relabeling techniques presented in the next section.

L-MCLA With the notion of triple presented for the previous methods, Shao and Ding [40] extended MCLA to include it in the construction of the graph using the Jaccard Similarity, to address the fact that the indirect relationships between clusters weren't addressed. The final partition π^* is obtained by employing the normalized cut algorithm on the graph generated by L-MCLA, which is partitioned in k^* parts.

Relationship between points and clusters

These algorithms generate either a bipartite graph or a hypergraph from the BA matrix, and they return directly how points are associated to groups of clusters, making also these methods akin to relabeling and voting algorithms.

Hyper-Graph Partitioning Algorithm (HGPA) HGPA [43] makes use of the binary cluster-association matrix BA to construct a hypergraph from the ensemble.
Each data point is associated to a vertex, while each cluster is associated to a same-weighted hyper-edge.
The resulting hypergraph is then partitioned in k^* parts using HMETIS to obtain the final partition π^*.

Clustering Ensemble via Structured Hypergraph Learning (CESHL) Starting from the same data used by HGPA, this method, developed by Zou et al. [54], dynamically learn the topology and the weights of the hypergraph by solving iteratively different optimization problems. The assumption is that a pre-defined static hypergraph is not sufficiently reliable and usually requires a posteriori computation to remove ambiguity. With a dynamic hypergraph it's possible to enforce desired requirements on the clustering structure. This is the only graph-based algorithm we present that does not employ a graph partitioning phase due to its dynamic nature.

Hybrid Bipartite Graph Formulation (HBGF) This method creates a bipartite graph $G = (V, W)$ to solve the cluster ensemble problem.
In HBGF [17], $V = V^X \cup V^C$ is the union of the set of vertices associated to data points V^X and clusters V^C, with $V^X = \{x_i | x_i \in X\}$ and $V^C = \{C_j | C_j \in \Pi\}$.
The weight $w_{ij} \in W$ between vertices $v_i, v_j \in V$ takes different values depending on which subset of V the vertices belong to:

$$w_{ij} = \begin{cases} 0 & \text{if } v_i \in V^X \wedge v_j \in V^X \vee v_i \in V^C \wedge v_j \in V^C \\ [BA_g]_{ij} & \text{if } v_i \in V^X \wedge v_j \in V^C, \end{cases} .$$

In Figure 3.10 we show how this graph is constructed, over the same example dataset as Figure 3.9: each node representing a point is connected only to a node representing a cluster and viceversa.
The bipartite graph is then partitioned in k^* parts either using the spectral graph partitioning algorithm or METIS to obtain the final partition π^*.
This method can be seen as an extension of CSPA and MCLA, in the sense that it combines both the information at cluster-level and data-level to obtain the final partitioning.

Weighted Bipartite Partitioning Algorithm (WBPA) This method [13] follows the same basic principle of HBGF, but instead of having weights equal to 1, which were the values of the BA matrix, it makes use of the WDM entries defined in the WSPA

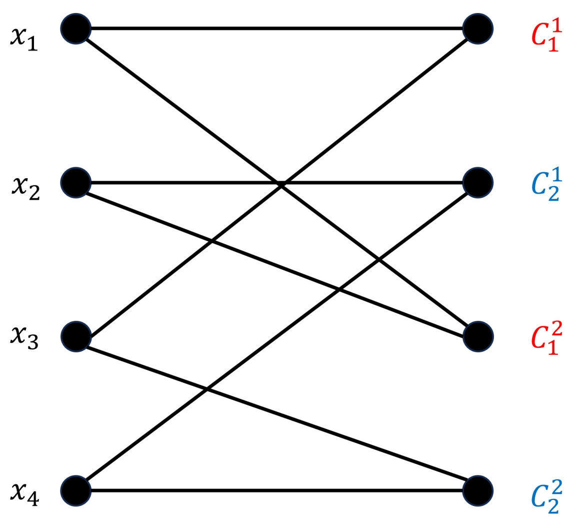


Figure 3.10: Example of the bipartite graph created by HBGF.

method.
Just like HBGF, $V = V^X \cup V^C$ is the union of the set of vertices associated to data points V^X and clusters V^C, with $V^X = \{x_i | x_i \in X\}$ and $V^C = \{C_j | C_j \in \Pi\}$.
The weight $w_{ij} \in W$ between vertices $v_i, v_j \in V$ takes different values depending on which subset of V the vertices belong to:

$$w_{ij} = \begin{cases} 0 & \text{if } v_i \in V^X \wedge v_j \in V^X \vee v_i \in V^C \wedge v_j \in V^C \\ WDM_{ij} & \text{if } v_i \in V^X \wedge v_j \in V^C \end{cases}.$$

The bipartite graph is then partitioned in k^* parts either using the spectral graph partitioning algorithm or METIS to obtain the final partition π^*.
This method can be seen as an enrichment of HBGF, which not only considers the information at data-level and cluster-level, but it also takes into consideration how much the data points should belong to clusters in the given ensemble.

Link-based Cluster Ensembles (LCE) Alternatively to CTS and SRS, LCE [30] is proposed as an extension of HBGF, which makes use of the BA matrix. Just like the CO matrix, the BA matrix can have a problem of too many unknown (0 entries) relations, which can be dealt with by estimating additional similarities between clusters, using a graph representation of links between clusters just like in CTS. LCE introduces an enhanced version of the BA matrix called the *refined cluster-association* (RA) *matrix*, which aims to make use of known relations between clusters (the 1 entries in the BA matrix) to estimate the value of unknown ones. To initialize the RA matrix, the known entries in BA are used:

$$[BA_g]_{it} = 1 \implies [RA_g]_{it} = 1.$$

After the initialization of known relations, for each partition π_g and its clusters $C_1, ..., C_{k_g}$, the association degree $RA_{it} \in [0,1]$ that sample x_i has with each cluster $C_t \in \{C_1, ..., C_{k_g}\}$ is estimated as:

$$RA_{it} = \begin{cases} 1 & \text{if } C_t = l(x_i) \\ Sim_{WCT}(C_t, C_{l(x_i)}) & otherwise \end{cases}$$

Once the RA matrix is obtained, it is transformed into a bipartite graph $G = (V, W)$, in a similar way to HBGF: $V = V^X \cup V^C$ is the union of the set of vertices associated to data points V^X and clusters V^C.
The weight $w_{ij} \in W$ between vertices $v_i, v_j \in V$ takes different values depending on which subset of V the vertices belong to:

$$w_{ij} = \begin{cases} 0 & \text{if } v_i \in V^X \land v_j \in V^X \lor v_i \in V^C \land v_j \in V^C \\ RA_{ij} & \text{if } v_i \in V^X \land v_j \in V^C \end{cases}$$

The final partition π^* is obtained by employing the spectral graph partitioning algorithm on G, which is partitioned in k^* parts.
The main advantage that LCE has over CTS is that instead of computing a pairwise similarity using the WCT algorithm for every pair of data points x_i, x_j, a significant chunk of the computation is taken away by exploiting already known information using the much easier to compute BA matrix.

Advantages and disadvantages

The graph-based methods we shown have the advantage of being intuitive to understand and really easy to implement. In general they all look for some form of similarity or association measure, which usually is really easy to compute (for example the co-association matrix or binary cluster-association matrix) between points or clusters to construct the graph, which is then partitioned to return how these should be grouped together in the consensus partition. Depending on the method, they can have low computational complexity, since methods such as HGPA, MCLA and HBGF have computational complexity of respectively $O(kNM), O(k^2NM^2), O(kNM)$. Their main downsides are the fact that several of them are computationally heavy, since the computational cost depends on how the graph is constructed and how many nodes there are in it. For example, CSPA has a computational cost of $O(kN^2M)$, and similar methods such as GCC and SNNC have similar performances, since they all construct the graph from the co-association matrix, which is really expensive to compute. WSPA, CTS and SRS share the same downside, with the addition that they have to compute additional measures to create the matrix necessary for the graph. Another downside is that the output of these methods is heavily reliant on the partitioning algorithm that is used, so it is not guaranteed that, for example, using METIS, which is known for partitioning a graph in parts of equal size, it may yield the best result for many of them.

4 | Theoretical Background

In this chapter we illustrate the key concepts that are needed to understand the solution we propose for the Cluster Ensemble problem.

4.1. Synchronization

The method we propose makes use of a technique known as *Synchronization*, also known by the names of *averaging* [21] or *graph optimization* [10]. The synchronization problem is stated as follows: given a collection of nodes belonging to a network, each of which is characterized by a state that is unknown, and a collection of pairwise measures between nodes that measure the difference between states, the goal is to retrieve the information about the unknown states from the pairwise measures. Known applications of synchronization are time synchronization (from which the name *Synchronization* comes from) in distributed systems and networks [32] [20], and different types of computer vision problems such as rotation synchronization, rigid-motion synchronization and permutation synchronization [2].
The problem of synchronization can be modeled as a graph $G = (V, W)$, where the nodes V correspond to the states and the edges W correspond to the pairwise measures. The problem is well posed only if such graph is connected. The states of each node are modeled mathematically as elements of a group Σ and a different choice of group represents a different kind of synchronization problem. Some examples of group choices for synchronization are:

- $\Sigma = \mathbb{R}$, which corresponds to time synchronization.
- $\Sigma = SO(d), SO(d) = \{M \in \mathbb{R}^{d\times d} \text{ s.t. } M^T M = MM^T = I_d, \det(M) = 1\}$ which is called the *Special Orthogonal Group*, it is the set of all rotations and it is used for rotation synchronization.
- $\Sigma = SE(d), SE(d) = \left\{ \begin{bmatrix} M & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}, \text{ s.t. } M \in SO(d), \mathbf{t} \in \mathbb{R}^d \right\}$, which is called the *Special Euclidean Group*, it is the set of direct isometries which is used for rigid-

motion synchronization.

- $\Sigma = \mathscr{S}_d, \mathscr{S}_d = \{M \in \{0,1\}^{d \times d} \text{ s.t. } M\mathbf{1} = \mathbf{1}, \mathbf{1}M = \mathbf{1}\}$, which is called the *Symmetric Group*, it is the set of total bijections (or permutations) between sets of objects and it used for permutation synchronization.

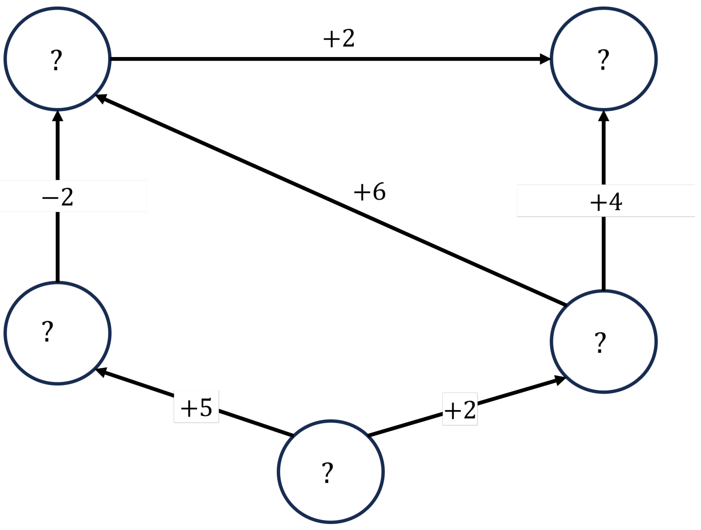


Figure 4.1: Example of the synchronization problem on $(\mathbb{R}, +)$

In Figure 4.1 we show an example of how a synchronization problem is modeled, given that the states and pairwise measures are elements of the group of real numbers with algebraic sum $(\mathbb{R}, +)$. Each edge is labeled with a number that represents the difference (in the mathematical sense) of the unknown states of each vertex, which here we represent with a quotation mark to underline the fact it is not known a priori. The goal of a synchronization algorithm is to retrieve such states only by knowing the measures on the edges.

In our work, we focus on synchronization, specifically synchronization over permutations, since a good part of the framework used for relabeling and voting makes use of permutations to align labels of the same clusters in different partitions and it constitutes a critical part in the efficiency of a given relabeling and voting algorithm. Before entering into the details of permutation synchronization, we introduce some useful concepts and notation first to define the general framework of the synchronization problem.

4.1.1. Framework and notation

First of all, a core concept is the one of *group-labeled graph*: let $(\Sigma, *)$ be a group with its unit element 1_Σ and $G = (V, W)$ a simple directed graph, with vertex set $V = \{1, 2, \ldots, n\}$ and edge set W, with $|W| = m$. A Σ-*labeled* graph is a directed graph where each element of the edge set is labeled with an element of Σ. It is represented by a tuple $\Gamma = (V, W, z)$, where z is a labeling function from elements of W to elements of Σ, that is

$$z : W \longrightarrow \Sigma$$

such that if $(i, j) \in W$ then $(j, i) \in W$ and

$$z(j, i) = z(i, j)^{-1}$$

Definition 1 Given a Σ-labeled graph $\Gamma = (V, W, z)$, a cycle $\{(i_1, i_2), (i_2, i_3), \ldots (i_l, i_1)\}$ is *cycle-consistent* if the composition of the edge labels along the cycle returns the unit element:

$$z(i_1, i_2) * z(i_2, i_3) * \cdots * z(i, i_1) = 1_\Sigma$$

Definition 2 Given a Σ-labeled graph $\Gamma = (V, W, z)$, and $v : V \longrightarrow \Sigma$ a vertex labeling. v is a *consistent labeling* if and only if

$$z(i, j) = v(i) * v(j)^{-1} \quad \forall (i, j) \in W$$

This propriety means that each edge label is the difference between the corresponding vertex labels. This condition is referred to as *consistency constraint* and it is also expressed as

$$z(i, j) * v(j) = v(i) \quad \forall (i, j) \in W$$

A consistent labeling is such that any composition of pairwise matchings over a cycle is cycle-consistent. This condition is called *cycle-consistency.*
A consistent labeling is defined up to a global product with any group element, which means that if $v : V \longrightarrow \Sigma$ is consistent, then $y : V \longrightarrow \Sigma, y(i) = v(i) * s$ is consistent too, for any $s \in \Sigma$.
An important result related to these two definitions needs to be introduced:

Result 1 Given a Σ-labeled graph $\Gamma = (V, W, z)$, *there exists a polynomial algorithm which either finds a non-consistent cycle in* Γ *or finds a consistent labelling of* Γ, achieving *cycle-consistency.* A corollary of this result is that *the* Σ*-labelled graph* $\Gamma = (V, W, z)$ *has a consistent labelling if and only if it does not contain a non-consistent cycle.*

Given these definitions and results, we can define the problem of synchronization in mathematical terms. Assuming there is for Σ a metric function $\delta : \Sigma \times \Sigma \longrightarrow \mathbb{R}^+$, representing a distance between elements of the group, and a non-negative, non-decreasing function $\rho : \mathbb{R}^+ \longrightarrow \mathbb{R}^+$, with unique minimum in 0 and $\rho(0) = 0$, an example of which can be the squared loss function.

Definition 3 Let $\Gamma = (V, W, z)$ be a Σ-labeled graph, and $\tilde{v} : V \longrightarrow \Sigma$ a vertex labeling. The *consistency error* of $\tilde{v}$ is defined as:

$$\epsilon(\tilde{v}) = \sum_{(i,j)\in W} \rho\Big(\delta\Big(\tilde{z}(i,j), z(i,j)\Big)\Big)$$

where $\tilde{z}$ is the edge labeling induced by $\tilde{v}$: $\tilde{z}(i,j) = \tilde{v}(i) * \tilde{v}(j)^{-1}$. cycle-consistency is achieved if and only if there is zero consistency error. In practical applications, where the pairwise measures are affected by noise, this condition is hardly reachable, so synchronization needs to solve a different problem.

Definition 4 Given a Σ-labeled graph $\Gamma = (V, W, z)$, the *group synchronization problem* consists in finding a vertex labeling that minimizes the consistency error. The general scope of the synchronization problem is then to recover the unknown vertex labels from a redundant set of noisy pairwise measurements modeled as edge labels.

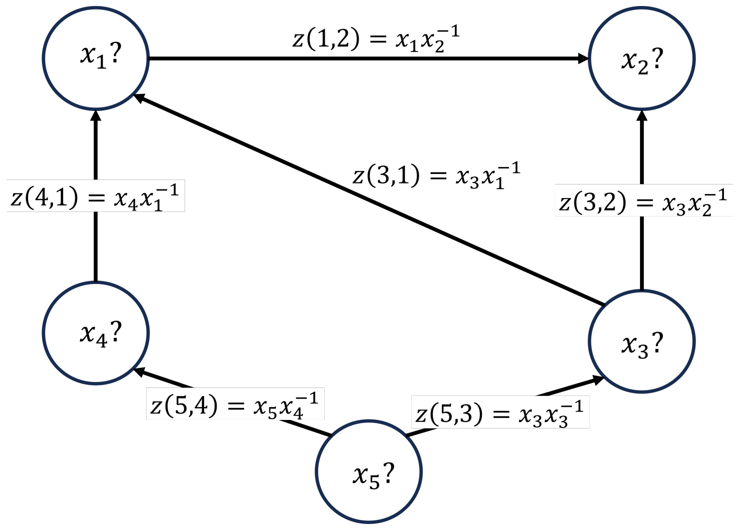


Figure 4.2: General framework of the synchronization problem

In Figure 4.2 we show what is the general theoretical appearance and framework of a generic synchronization problem on a Σ-labeled graph $\Gamma = (V, W, z)$ over any group $(\Sigma, *)$

with any operation. Each edge is labeled with its proper edge labeling $z(i,j)$, and each state is unknown. A synchronization algorithm should retrieve a vertex labeling v that minimizes the consistency error, and in turn should be as close as possible in satisfying the consistency constraint over each edge $z(i,j) \approx v(i) * v(j)^{-1}$, as shown over each edge in the figure.

Synchronization over different groups is solved in different ways using different algorithms, and in general a closed form solution for the problem is not always available. In the following we will focus on the symmetric group and how synchronization is solved for permutations.

4.1.2. Permutation Synchronization

The problem

In permutation synchronization, we consider as group for our Σ-labeled graph the group of permutations between sets of $k_1, k_2, \dots, k_M$ objects. Let us consider a collection of M sets $O_1, O_2, \dots, O_M$ of k_i objects each, $O_i = \{o_1, o_2, \dots, o_{k_i}\}$ and for each pair O_i, O_j, each object in O_i may have a counterpart in O_j. There is a correspondence between O_i and O_j represented by a permutation $P_{ji} : \{1, 2, \dots, k_j\} \longrightarrow \{1, 2, \dots, k_i\}$, and those permutations are what we use as edge labels for our Σ-labeled graph:

$$z(i,j) = P_{ij}, \quad \forall (i,j) \in W$$

To define cycle-consistency of permutations we make use of the definition of universe. Cycle-consistency is achieved if:

$$P_{ij} = P_i P_j^T, \quad \forall (i,j) \in W$$

Each vertex label P_i is interpreted as, given a reference universe set where there is present a reference ordering of its objects, as the realization of an absolute permutation between each object in O_i and an object in the universe. These absolute permutations are unknown and their retrieval is the goal of our synchronization problem, where we have at our disposal noisy pairwise permutations $\tilde{P}_{ij}$ which we have to synchronize.

Given the block matrix of the permutations $\mathbf{P}$, which represents the Σ-labeled graph in matrix form. Each block contains the permutation matrix for each edge (i, j):

$$\mathbf{P} = \begin{bmatrix} P_{11} & \dots & P_{1M} \\ \vdots & \ddots & \vdots \\ P_{M1} & \dots & P_{MM} \end{bmatrix}$$

we say that $\mathbf{P}$ is cycle-consistent if and only if there exists a matrix U such that $\mathbf{P} = U \cdot U^T$. U is a block matrix where each block is a $k_i \times k^*$ permutation matrix of the form:

$$U = \begin{bmatrix} P_1 \\ \vdots \\ P_M \end{bmatrix}$$

This factorization of $\mathbf{P}$ can be directly retrieved for the noise-free case. Since synchronization algorithms operate in the presence of noise, a variety of methods can be used to find the solution that better approximate the ideal one.

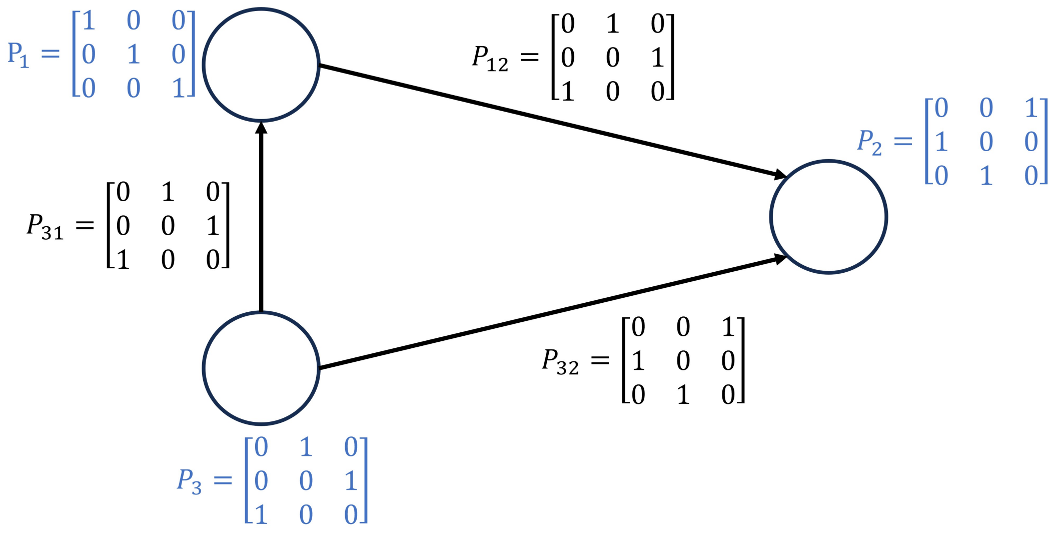

$P_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$P_{12} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$
$P_2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$
$P_{31} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$
$P_{32} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$
$P_3 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$

Figure 4.3: Example of the permutation synchronization problem

In Figure 4.3 we can see an example of how the permutation synchronization problem is modeled. The edge labels in the graph are pairwise permutations between the reference orderings that are instead modeled by the vertex labels, which are unknown. The goal of the permutation synchronization algorithm is to retrieve such reference permutations, which we denoted in the image as permutation matrices associated to each node, in light blue.

Solution

There exist a number of Permutation Synchronization algorithms, that work under different assumptions to approximate the factorization of $\mathbf{P}$.

Eigenvectors-based solution In the work of Pachauri [39], the algorithm operates under the assumption that all permutations have the same size, such that U is a block matrix where each block has the same dimension $k^* \times k^*$ and it is found by finding the k^* leading eigenvectors of $\mathbf{P}$ and then projecting each block back into the Symmetric Group, since they are not guaranteed to belong to it after the best U is found. This is obtained by multiplying each block by a orthogonal matrix Q that in the case of full permutations is $Q = P_1^T$.

Non-negative matrix factorization-based solution From the work of Bernard et al. [5], the permutation synchronization is extended to partial permutations, where non-matchings may be present, and works under the assumption that the desired k^* for the solution is known. Generally the permutation synchronization problem can be formulated as a constrained non-linear least squares problem:

$$\arg\min_{U} ||\mathbf{P} - UU^T||_F^2,$$

Bernard et al. [5] formulate the problem instead as a *Non-negative Matrix Factorisation*:

$$\arg\min_{V\geq 0, H\geq 0} ||\mathbf{P} - VH||_F^2$$

where $V \in \mathbb{R}^{k_{tot}\times k^*}$ and $H \in \mathbb{R}^{k^*\times k_{tot}}$. The problem is not convex, so an iterative solution is required to find V and H. The two critical problems are the initialization of V and H and the final projection of the desired U from V. To solve this problems an algorithm called Successive Block Rotation Algorithm (SBRA) is used to find a rotation matrix, used for aggregating information of each block in a generic block matrix.

QuickMatch in the work of Tron et al. [46], the matrix $\mathbf{P}$ is viewed as the adjacency matrix of an M-partite graph, for this reason this solution is quite dissimilar from the other ones we showed so far, instead of being solved as an optimization problem, here permutation synchronization is cast to a graph-clustering problem. This method is named QuickMatch and it works with partial permutations and it automatically estimates the value of k^* for the solution, so it doesn't have any of the limitations of the other algorithms, at the cost of slightly greater execution time.

5 | Proposed Method

In this chapter we propose a new consensus function for the Cluster Ensemble problem. We explain how the theoretical framework presented in Chapter 4 is integrated in the Cluster Ensemble phases and we analyze its advantages over other consensus functions that make use of similar procedures.

5.1. Permutation Synchronization to solve the Cluster Ensemble problem

In Chapter 3 we discussed some methods pertaining to the state of the art of the solutions of the Cluster Ensemble problem. In particular, we analyzed the techniques that divide the problem in the two phases of Relabeling and Voting: the relabeling can be seen as the problem of finding an optimal permutation of the columns of BA_g for each g. We look for a permutation matrix P_g of size $k_g \times k^*$, such that BA_gP_g represents the relabeled binary association matrix of π_g. In general, P_g may be a partial permutation. Given this formulation, it became natural to formulate the relabeling phase as a problem of permutation synchronization, making use of the theoretical framework presented in Chapter 4.
Given a collection of vertices belonging to a graph, with the state of each unknown, and a collection of edge labels between vertices that measure the difference between states, the goal of graph synchronization is to retrieve the information about the unknown states from the pairwise measures. In the case of permutation synchronization the edge labels are labeled with permutation matrices.

The use of a graph-based solution to solve the Cluster Ensemble problem is already known in literature and widely appreciated, but as far as the current state of the art goes, synchronization is not contemplated to formulate the problem and it's more recurring that the graph is constructed directly from the relationship between points and clusters. In that family of methods the consensus partition is obtained using graph partitioning.

Graph Synchronization can be used in different application scenarios, as presented in

Chapter 4, for example the computer vision tasks of estimation of rotations and rigid body isometries [2], and it can be seen as a general framework for recovering an unknown truth when it isn't available a priori. In the family of Relabeling and Voting methods, we need to obtain a label correspondence between all partitions of the ensemble, and for this reason we deemed very appropriate to extend the known relabeling algorithms with the use of synchronization, given that the majority of them have to choose a reference permutation of cluster labels to operate, leading the result to be biased towards that reference. By extending the Relabeling and Voting framework with Permutation Synchronization we are able to obtain a global relabeling that doesn't depend on any local selection of a reference or a reference ordering of the partitions, since Synchronization takes care of selecting a global reference autonomously.

In the following we will illustrate the steps to achieve a consensus partition, making use of graph synchronization and known voting mechanisms. We call this approach Synchronization and Voting.

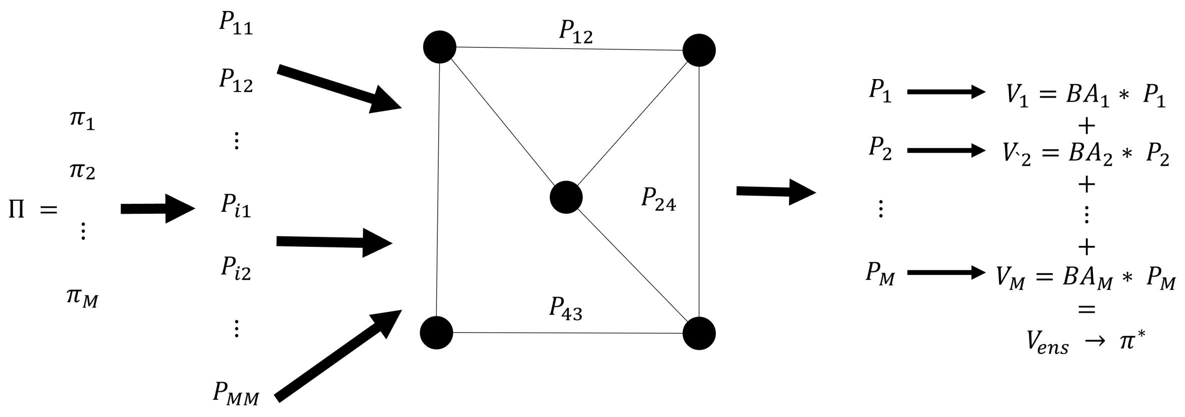

$\Pi = \pi_1, \pi_2, \vdots, \pi_M$
P_{11}
P_{12}
P_{i1}
P_{i2}
P_{MM}
P_{12}
P_{24}
P_{43}
$P_1 \rightarrow V_1 = BA_1 * P_1$
$P_2 \rightarrow V_2 = BA_2 * P_2$
$P_M \rightarrow V_M = BA_M * P_M$
$V_{ens} \rightarrow \pi^*$

Figure 5.1: Cluster Ensemble with permutation synchronization

Figure 5.1 shows the entire process of Synchronization and Voting, starting from a provided ensemble until the consensus partition is obtained. From a given ensemble, the pairwise permutations between partitions P_{ij} are computed, which are then placed as edge labels for the graph. After that, the individual vertex labels P_i are retrieved. Each one of these absolute permutations is multiplied with the BA matrix of the respective partition to obtain a vote (that may be multiplied with a weight), and then each vote is summed to obtain the final voting matrix V_{ens}. From that, the consensus partition π^* is obtained.

Synchronization and Voting can be summarized as being composed of two phases:

1. The construction of a graph by the computation of the relative permutations $P_{12}, \ldots, P_{ij}, \ldots$

by solving multiple linear assignment problems;

2. the recovery of the absolute permutations $P_1, \ldots, P_M$ by solving a single permutation synchronization problem.

In the following sections each phase is contextualized in the Cluster Ensemble scenario and is described in details.

5.1.1. Graph generation

let $G = (V, E)$ being a simple directed graph. V is a set of vertices with unknown state $v(i)$ which correspond to a partition π_i and E is a set of edges labeled with a permutation between cluster labels. We compute the edge labels $z(i, j)$ between node i and node j, obtained as the relabeling P_{ij} between the two partitions π_i and π_j. The graph can be computed using all the relative permutations, resulting in a complete graph, but for efficiency reasons it's possible to select a subset of all the possible edges, resulting in a graph with missing edges. This selection is made in such a way that the graph that is ultimately created is always connected.

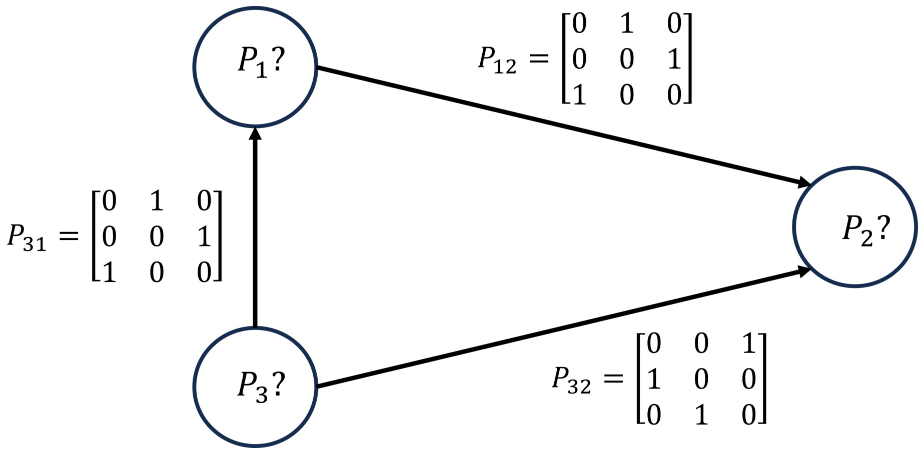
Each matrix P_{ij} is obtained as the resolution of the relabeling problem between the two partitions π_i and π_j through the Hungarian algorithm, and the matrix P_{ji} is simply generated by transposing the matrix P_{ij}, since in the group of permutations the inverse of a matrix is the transpose.

$$\begin{aligned} z(i, j) &= P_{ij}, \quad (\pi_i, \pi_j) \in \Pi \\ z(j, i) &= P_{ji} = P_{ij}^{-1} = P_{ij}^T \end{aligned}$$

All the matrices P_{ii}, which are the self-loops in the graph, are identity matrices of dimension k_i, and in case the measurement of P_{ij} is not available, a matrix of zeros is used. As a consequence, a correspondence between nodes and partitions is created: node i corresponds to the i-th partition. The number of nodes will then be equal to the number of partitions independently from the number of edges used.

This graph is represented mathematically, as presented in equation **??**, with the block matrix $\mathbf{P}$, which contains in each block the permutation matrix P_{ij} between partition π_i and partition π_j.

$$\mathbf{P} = \begin{bmatrix} P_{11} & \ldots & P_{1M} \\ \vdots & \ddots & \vdots \\ P_{M1} & \ldots & P_{MM} \end{bmatrix}$$



P_1?
P_2?
P_3?
$P_{12} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$
$P_{31} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$
$P_{32} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

Figure 5.2: Example of the graph generated by our method

Figure 5.2 shows an example of the graph we generate in our solution, in the case of an ensemble with just three partitions: each edge will be labeled by a permutation matrix generated by solving the labeling assignment problem between each pair of the partitions $\{\pi_1, \pi_2, \pi_3\}$ with the Hungarian algorithm.

5.1.2. Relabeling, synchronization and voting

After computing $\mathbf{P}$, we apply a suitable Permutation Synchronization Algorithm to obtain the vertex labels. The main idea of Permutation Synchronization algorithms is to estimate the global matching vector U, which is a block vector containing in each position a permutation matrix associated to each vertex:

$$U = \begin{bmatrix} P_1 \\ \vdots \\ P_M \end{bmatrix} = PermSync(\mathbf{P})$$

The recovered vertex labels $v(i) = P_i$ associated to each node (and hence the final synchronisation result) can be interpreted as 'absolute' permutations, obtained with respect to a global reference created during the algorithm steps, which should represent the truth of how the clusters in each partition should be reordered. As result we obtain for each node a relabeling created by a permutation with respect to an arbitrary reference, represented in the global matching vector U.

To obtain U we focus on three different permutation synchronization approaches, each one working under different assumptions:

- In [39] the absolute permutations are recovered by finding the k^* leading eigenvectors of $\mathbf{P}$. This method works under the assumptions that k^* is known a priori and all permutations are total.
- In [5] the absolute permutations are computed from the non-negative matrix factorization (NMF) of $\mathbf{P}$. This method works with partial permutations but requires to know k^* in advance.
- In [46] the matrix $\mathbf{P}$ is viewed as the adjacency matrix of an M-partite graph and permutation synchronization is cast to a graph-clustering problem; this method is named QuickMatch, it works with partial permutations and it automatically estimates the value of k^*

After obtaining the result of synchronization, all the partitions are relabeled with the corresponding permutation, to obtain a consistent relabeling for all the partitions in the ensemble, by multiplying the Binary Association Matrix with the absolute permutation matrix associated to the partition:

$$V_i = BA_i \cdot P_i$$

Each matrix V_i represents the vote proposed by the partition π_i. At this point, it's possible to apply any voting technique to obtain the voting matrix V_{ens} and retrieve the consensus label for each point.

5.1.3. Pseudocode

Below we present the pseudocode for the whole algorithm: it is presented in a general way, but for a specific implementation it requires to select a Permutation Synchronization Algorithm and a Voting Mechanism, depending on the application scenario it is utilized. In our example we chose, for simplicity, simple voting as a voting mechanism. The selection of a Permutation Synchronization Algorithm depends on the contextual limitations regarding the knowledge of k both in the input partitions and in the output one: we presented three different algorithms, that cover all the possible use cases, from having the same k_g in each partition [39], to not having any assumption on the input and output number of clusters [46]. We defined the combination of each type of these Permutation Synchronization Algorithms with a specific voting mechanism respectively **SV-EIG**, **SV-NMF** and **SV-QM**.

Algorithm 5.1 Synchronization and Voting

```
    Input: The ensemble $\Pi$
 1: Selection of a set of edges $E$
 2: for $(i, j) \in E$ do
 3:     $\mathbf{P}_{ij} = P_{ij}$ as calculated by applying the Hungarian algorithm between the BA
        matrices $BA_i$ and $BA_j$ of partitions $\pi_i$ and $\pi_j$
 4:     $\mathbf{P}_{ji} = P_{ij}^T$
 5: end for
 6: for $i = 1, \dots, M$ do
 7:     $\mathbf{P}_{ii} = I(k_i)$
 8: end for
 9: $U = PermSync(\mathbf{P})$
10: for $i = 1, \dots, M$ do
11:     $V_i = BA_i \cdot P_i$
12: end for
13: $V_{ens} = \sum_{i=1}^{M} V_i$
14: for $i = 1, \dots, N$ do
15:     $l^*(x_i) = \arg\max V_{ens}(x_i)$
16: end for
    Output: the consensus partition $\pi^*$
```

5.2. Advantages and Limitations

The main advantage of our method is that it is modular and many version of permutation synchronization can be plugged in depending on the use case. Another advantage it has over known relabeling and voting methods, such as [26], is the fact that it doesn't depend on a reference partition for relabeling and considers a global view, taking into account the redundancy represented by the whole graph, promoting error compensation.

One of the main advantages of our method is its modularity: depending on the knowledge regarding the Ensemble or the desired solution different versions of permutation synchronization can be plugged in. The innovation in this type of approach, with respect to existing relabeling and voting algorithms, is that it would make it possible to solve the problem of label correspondence no longer with respect to an arbitrarily chosen reference or sequentially with respect to a reference ordering. The topology of the graph allows it to be more resistant to the noise of the individual permutations, guaranteeing greater

consistency of the result, at the expense of greater computation time. Since the generation and topology of the graph can be customised, it is also possible to generate incomplete graphs with a smaller number of arcs, selected according to a particular criterion. In this way, it is possible to manage the trade-off between execution time and accuracy.
Also, the advantage of synchronization over other methods of relabeling is that it tries to retrieve an information that is "absolute", and for this reason it should be something that is closer to the truth of how the clusters in each partition should be relabeled, leading to a more accurate and unbiased voting process. It is also possible to redefine the traditional relabeling and voting problem as a special case of graph synchronisation: it is in fact possible to solve the problem using a graph where each node-partition is only an arc directed towards a node-reference. This change of paradigm from a biased to a global view increases the computational time of the solution: relabeling with respect to an arbitrarily chosen reference requires to execute relabeling with Hungarian M times, so it would lead to a computational complexity of $O(Mk^3N)$, our method computes $|E|$ relabelings with Hungarian, leading to a computational complexity of $O(|E|k^3N)$, with $M-1 \leq |E| \leq \frac{M(M-1)}{2}$ instead. For this reason the usage of this technique in ensembles containing a high number of partitions could become unfeasible. Additionally, to generate the graph and correctly apply synchronization, our method requires to generate a minimum number of permutations which can increase significantly with the number of partitions, depending on how many edges we want to create for the graph. However in most of the real case scenarios we analyzed, the execution time was comparable and in some cases better than other state-of-the-art algorithms. Even more, we expect our method to give an increase and more guarantees regarding performance: we expect the result to be more accurate than the previously described methods, given the large amount of information used.

Experimental results of the performance of our method will be analyzed in Chapter 6.

6 | Experiments

In this chapter we show some experiments we ran over our method and comparable methods that have been developed in literature, to assess the performance and goodness of our solution based on synchronization compared to what is known and well-used in many application scenarios. To assess and verify the assumptions we made for our proposed method, we ran different experiments using different metrics. We used multiple datasets, both synthetic and from real-world measurements, to create heterogeneous testing scenarios with different characteristics in ensemble generation, dataset features and number of data points.

6.1. Metrics

In this section we show the metrics we adopted to assess the goodness of the results of the experiments. We have chosen for our tests NMI, ARI, classification accuracy and execution time.

NMI and ARI We recall the definition of the Normalized Mutual Information, that measures the common information between two random variables using entropy:

$$NMI(\pi_a, \pi_b) = \frac{\sum_{i=1}^{k_a} \sum_{j=1}^{k_b} n_{ij} \log(\frac{n_{ij} \cdot N}{|C_i^a| \cdot |C_j^b|})}{\sqrt{(\sum_{i=1}^{k_a} |C_i^a| \log(\frac{|C_i^a|}{N}))(\sum_{j=1}^{k_b} |C_j^b| \log(\frac{|C_j^b|}{N}))}}$$

where $n_{ij} = |C_i^a \cap C_j^b|$ is the number of commonly labeled points between two clusters. The idea of the NMI is to measure how close a partition is to another, by looking at how much information they have in common considering the statistical information they share, that in Clustering is represented by the common labelled points between clusters.

The NMI is used to measure the consistency of the found consensus partition with right to the ensemble by computing the average NMI:

$$ANMI(\pi^*, \Pi) = \frac{1}{M} \sum_{\pi_g \in \Pi} NMI(\pi^*, \pi_g)$$

In addition to the NMI we also used another famous metric found in literature, the Adjusted Rand Index (ARI), which measures the similarity between partitions by considering the overlapping between all the possible clusters, by looking at all the combinations of shared points between clusters:

$$ARI(\pi_a, \pi_b) = \frac{\sum_{i=1}^{k_a} \sum_{j=1}^{k_b} \binom{n_{ij}}{2} - \left[\sum_{j=1}^{k_b} \binom{a_j}{2} \sum_{i=1}^{k_a} \binom{b_i}{2} \right] / \binom{n}{2}}{\frac{1}{2} \left[\sum_{j=1}^{k_b} \binom{a_j}{2} + \sum_{i=1}^{k_a} \binom{b_i}{2} \right] - \left[\sum_{j=1}^{k_b} \binom{a_j}{2} \sum_{i=1}^{k_a} \binom{b_i}{2} \right] / \binom{n}{2}}$$

Table 6.1 explains what a_j and b_i are in the formula: given every couple n_{ij} of commonly labeled points between clusters, they represent the sum of the commonly labeled points between a cluster in π_a and every cluster in π_b.

	C_1	C_2	$\cdots$	C_{k_a}	sum
C_1	n_{11}	n_{11}	$\cdots$	n_{1k_a}	a_1
C_2	n_{21}	n_{22}	$\cdots$	n_{2k_a}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
C_{k_b}	n_{k_b1}	n_{k_b2}	$\cdots$	$n_{k_bk_a}$	a_{k_b}
sum	b_1	b_2	$\cdots$	b_{k_a}	

Table 6.1: Contingency table of the sums of common points between clusters

Just like the NMI, ARI is used to measure the consistency of the found consensus partition with right to the ensemble by computing the average ARI:

$$AARI(\pi^*, \Pi) = \frac{1}{M} \sum_{\pi_g \in \Pi} ARI(\pi^*, \pi_g)$$

Both NMI and ARI have the advantage that they don't need a priori information about the cluster labels or need to match the labels between two different partitions, making them easy and fast to use.

Classification accuracy to analyze the similarity with the ground truth, we adopted a metric that evaluates the percentage of correctly classified data points in the consensus partition compared to the pre-defined cluster labels in the ground truth partition, called the Clustering Accuracy (*ACC*) [31]. This measure is defined as:

$$ACC(\pi^*, \pi_{true}) = \sum_{i=1}^{k^*} \frac{\max_{j \in \{1,2,\ldots,k_{true}\}} n_{ij}}{N}$$

This measure for each cluster in the consensus partition, finds the closest cluster in the ground truth partition k_{true} in terms of shared data points. The higher it is, the higher the consensus partition clusters the dataset in the same way and with the same shape as the ground truth.

Execution time Every method we tested had its execution time measured from the start of the method (from the moment it receives the ensemble data in input), to the end (the moment in which the final voting matrix is returned), in seconds. We didn't consider in our benchmark the extraction of the consensus partition since it's a process that should be relatively short and the same for every tested method that makes use of it. For CSPA, that doesn't have an explicit voting phase, the execution time is measured from the receiving of input to the extraction of the consensus partition.

6.2. Compared Methods

To better assess the performance of our proposed method we compared the results obtained with a different number of competitors, taken from the literature discussed in Chapter 3. We decided to use as percentage of edges for the Synchronization and Voting algorithms the 50% of total possible edges to provide an average example of balance between execution time and performance. Additionally the different implementations of Synchronization and Voting are compared with each other to assess advantages and disadvantages in different use cases. Two of them, **SV-NMF** and **SV-QM**, make use of the permutation synchronization algorithms NMFSync[1] and QuickMatch[2] developed and implemented by the respective authors.

Relabeling and Voting algorithms From the relabeling and voting family we have chosen as rival methods those that have a clear separation between the relabeling and voting phases, so that we can compare directly how our relabeling algorithm works against other popular relabeling methods. For this reasons we couldn't use as direct confront methods using iterative voting, which usually requires that the relabeling and voting step are repeated several time, alternately. We have chosen as relabeling algorithms the

[1]`https://github.com/fbernardpi/NmfSync`

[2]`https://sites.bu.edu/tron/2018/07/13/fast-multi-image-matching-via-density-based-clustering-quickmatch/`

bipartite weighted matching solved with Hungarian [45] and the approach using multi-variate linear regression [4]. Both approaches were implemented by us, and the former one was expanded by us to handle partial matchings since the original version didn't support this feature. For the voting part, we have chosen simple voting and weighted voting, choosing as weights the NMI with respect to a reference (which we will call in short Weighted Voting Simple) and the inverse of the average NMI over all partitions (which we will call in short Weighted Voting Average). The voting part of these algorithms was implemented by hand too.

Algorithms from other categories From other families of algorithms we have chosen the methods from the graph-based family developed by Strehl [42], as they are still really popular and perform quite well, so they represent a good benchmark to compare our method against. We have chosen MCLA, as it presents two distinct phases comparable to the relabeling and voting phases. We also have chosen CESHL [53], as it is a method that was developed recently and it comes from a family of graph based algorithms that can be compared to relabeling and voting. CESHL depends on a parameter called λ, which controls the optimization of the hypergraph. We have chosen 10^0 as a value for lambda since the authors specified in their original work that a value of λ contained in $[10^0, 10^3]$ doesn't change the output of the algorithm. The other method we have chosen for this category is CSPA, which was used directly from the implementation of the author [42]. It uses a different approach not comparable directly to Relabeling And Voting, but we decided to insert it in our experiments because in literature it is widely used for comparisons, making it a valid benchmark. All three algorithms were taken from the code provided by the respective authors, and we replaced the voting part of MCLA with plurality voting to make the algorithm aligned with those chosen from the relabeling and voting category. In addition we also chose to evaluate our performance against the selection of a random partition from the ensemble.

6.3. Datasets

To assess our method we chose a different number of datasets, having different characteristics, to test the performance of the Synchronization and Voting and compare it with the other competitors.

Synthetic Dataset We created a synthetic dataset generated with a Gaussian distribution of the data over clusters, in a 2-dimensional space [15]. In this way we had more control over the generation of the input data. The ensemble generation technique used for this dataset is the same as the real ones, and in addition we could better tests every characteristic of the Ensemble.

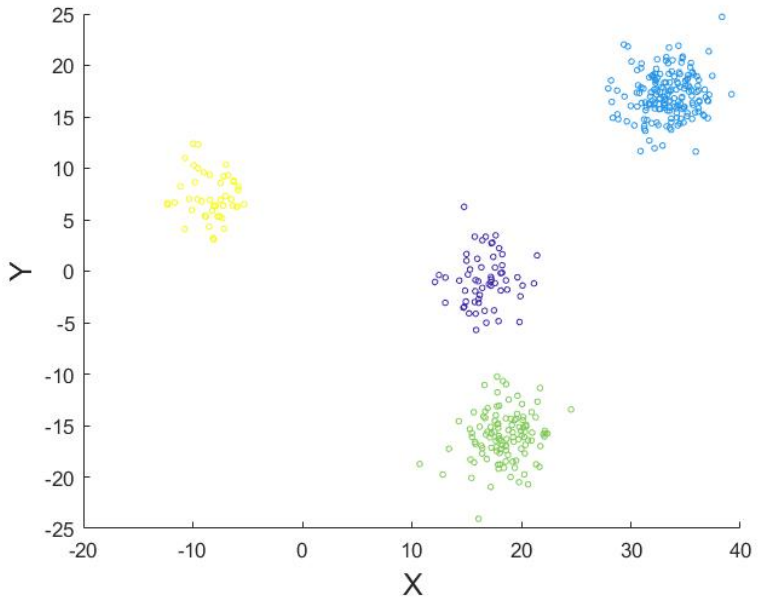


Figure 6.1: Example of a data distribution generated synthetically

In Figure 6.1 an example of data generation is provided, on a 2D space: four different clusters are generated (represented by the differently colored groups of data points in the image). Each group is generated by creating lines orientated randomly and their orthogonal. Points around these lines are then generated using a normal distribution. Each line and the cloud of points around it represent a cluster, that is used as a cluster in ground truth partition from which we generate the Ensemble.

Real World Datasets We considered for our experiments 8 real world datasets with the associated ground truth.

- **Iris** [18][19]: one of the most famous datasets from classification, that includes samples from the three main species of Iris (*setosa*, *virginica* and *versicolor*)
- **USPS, Mnist and Multiple Features (MF)**: each one of these datasets contains handwritten digits, the main differences are the sources and the features. USPS [28][27] was digitally scanned from the U.S. Postal Service in 16 × 16 grayscale pixels images. Mnist [36] images are created by US National Institute of Standards and Technology and are 28 × 28 grayscale images. Multiple Features [9][14] data is extracted from different maps from a Dutch public utility and is composed by different sets of features to describe the same original 30 × 48 images, including

Datasets			
Name	N	D	k
Iris	150	4	3
Multiple Features	2000	6349	10
Mnist	5000	784	10
Usps	11000	256	10
Isolet	1560	617	26
Lung Cancer	203	3312	5
Wine	178	13	3
Silhouette	846	18	3

Table 6.2: Characteristics of the different datasets

Fourier descriptors and morphological characteristics.

- **Isolet** [16][11]: a collection of audio samples of spoken alphabetical letters to train vocal recognition. The features are characteristics of audio waves such as spectral coefficients and contour features.
- **Wine** [47][1]: a collection of results of a chemical analysis of Italian wines cultivated in the same area but derived from three different varieties. The features are the quantities of 13 constituents found in each of the three types of wines.
- **Lung Cancer** [24][23]: The data describes 3 types of pathological lung cancers. The Authors give no information on the individual variables nor on where the data was originally used.
- **Vehicle Silhouette** [37]: In this dataset three types of vehicles are provided, using a set of features extracted from their silhouette. The vehicle may be viewed from one of many different angles.

In Table 6.2 are listed the characteristics of each dataset, in terms of number of points (N), number of features (D) and number of clusters (k).

Input and Ensemble Generation The input for our framework is simply the dataset, where for each point are specified all the features. To generate the ensemble, the k-means algorithm is used for each partition to create its binary association matrix. The number of cluster k is selected in different ways depending on the type of experiment: if the usage of SV-EIG was not required, for each partition a k_i was selected from the range $[k_{true} - 3, k_{true} + 3]$. Otherwise a fixed k is used, equal to k_{true}. K-means is initialized every time with a random seed, to avoid possible correlations between different results.

6.4. Results

In this section we discuss the results obtained with the experiments, addressing each metric and evaluating the performance of our method compared to the others. Each test is repeated a high number of times, to guarantee the consistency of the result. Each test is repeated for the three voting algorithms.

6.4.1. Complex datasets

From the datasets presented in Table 6.2 we identified some that are more complex to cluster than others, due to the high number of feature, points, clusters or all of them. In particular Usps, Mnist, MF and Isolet are the ones that were more resource-intensive, since with a high number of points and features, both k-means and the cluster ensemble algorithms take a significant amount of time to execute. For this reason, in this kind of experiment we found that SV-EIG, SV-NMF and SV-QM obtained slightly better results compared to the other ones on these datasets.

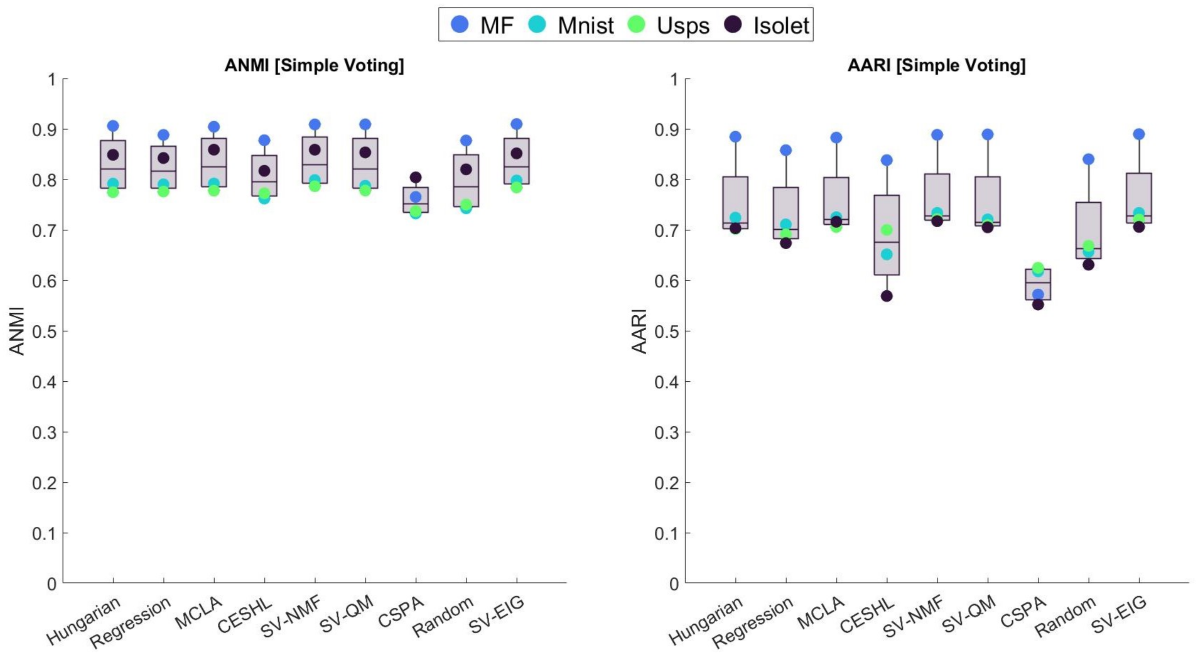


Figure 6.2: Performance on high complexity datasets

In Figure 6.2 a box plot representation of the average NMI and ARI is presented: each one of these complex datasets is a colored dot, and all of them are compared for each method. From the figure it can be seen that the Synchronization and Voting methods yield the average best results and better medians compared to the other cluster ensemble algorithms. To verify that this difference is statistically significant, we ran a a two-sided

Wilcoxon rank sum test between the three Synchronization and Voting algorithms and every other competitor with 5% significance. The result confirmed the significance of our improvement over the other methods.

The other datasets presented a modest number of clusters, features or points, making it easier to generate a consensus partition from the ensemble generated from them. In this case the performances were either comparable or slightly better to the other methods, depending on the specific dataset.

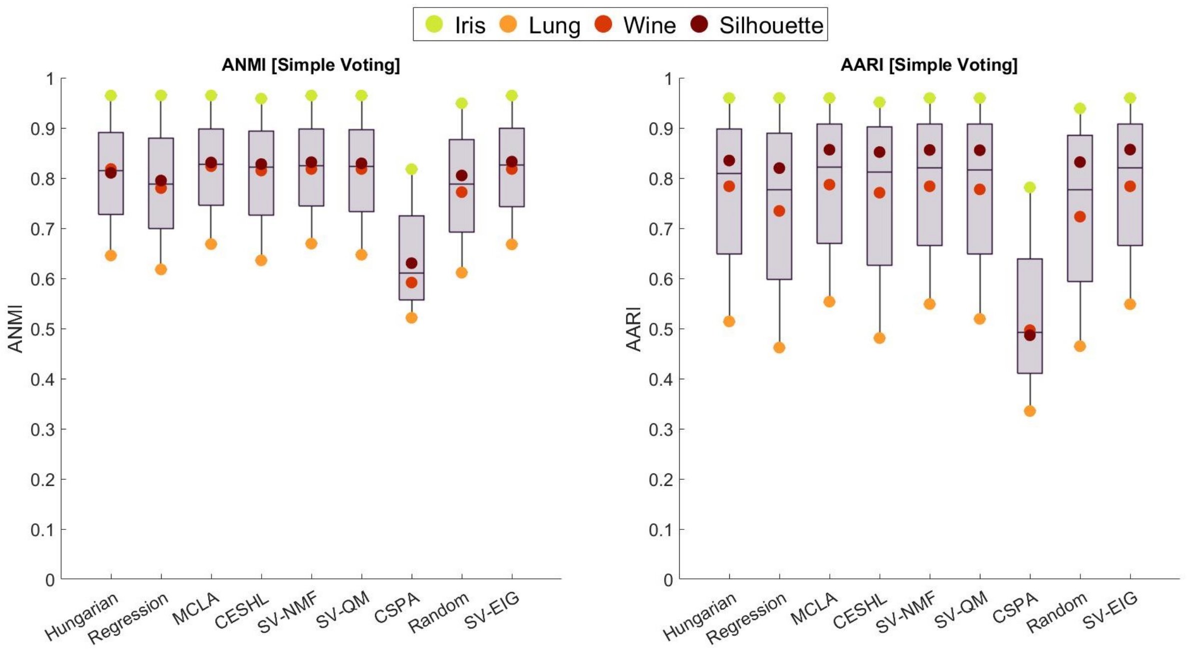


Figure 6.3: Performance on low complexity datasets

The figure 6.3 is structured as Figure 6.2, showing the results on the more simple datasets. In this case the results of Synchronization and Voting methods are comparable to the competitors: this result was also confirmed by running a two-sided Wilcoxon rank sum test.

6.4.2. Real case scenarios

Thanks to the Synthetic dataset it was possible to test the behaviour of each method by varying the parameters of the Ensemble. From this experiments we found that the three methods we presented are faster than the competitors based on graph partitioning in case the number of partitions in less than 50, which is a reasonable upper bound for real case scenarios. With a bigger number of partitions instead the time tends to increase more than the graph-based methods. The same considerations can be done by keeping fixed the number of partitions and increasing the number of cluster labels of each partition.

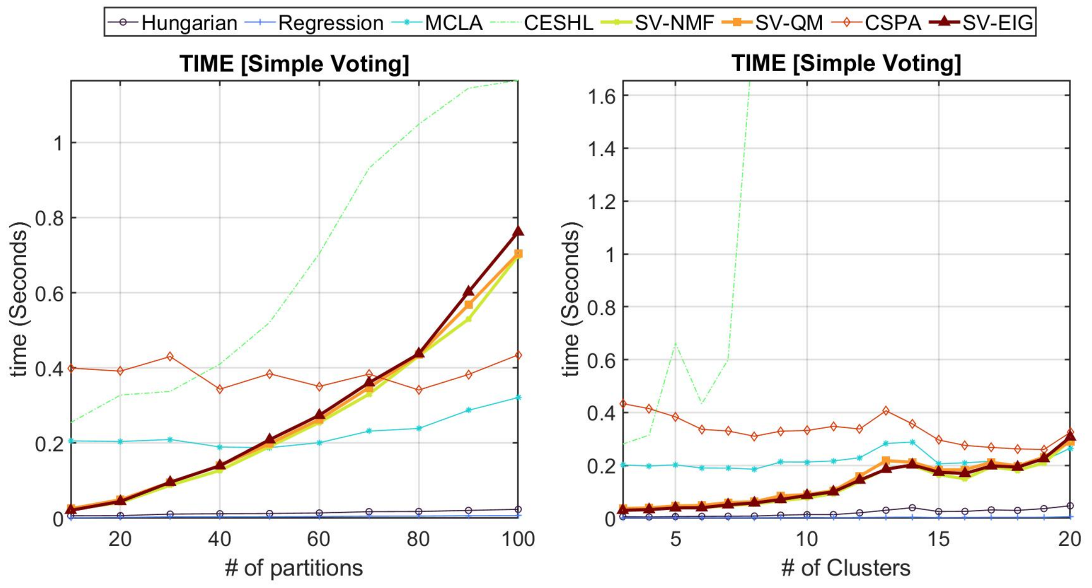


Figure 6.4: Execution time with increasing number of clusters and the number of partitions

Figure 6.4 shows the execution time of the methods over the synthetic dataset varying the number of partitions and the number of clusters. Each method has a differently styled and colored line, and the Synchronization and Voting ones are also in bold. In the left chart it is clear how under 50 partitions the Synchronization and Voting methods perform faster than the graph-based ones. The methods based on Relabeling and Voting are always faster due to the low number of relabelings required to compute a solution. In the right chart the number of partitions was fixed to 20 (a plausible number for a real-case scenario), and also in this case the execution time of Synchronization and Voting was better than the graph-based ones.

6.4.3. Further considerations

The framework of Synchronization and Voting compared to the competitors we shown, offers the possibility to customize the performance and execution time by selecting an appropriate parameter which regulates the completeness of the graph generated for synchronization, so that it can be suitable to different use cases, especially when execution time is a sensitive issue. In our experiment setup, with our specific ensemble generation, we could exploit a reduced number of computations for the edges without having a significant loss in performance: there isn't high heterogeneity in the partitions, so the information provided by the edges of the graph is redundant and it's possible to select a subset of them without losing significant information.

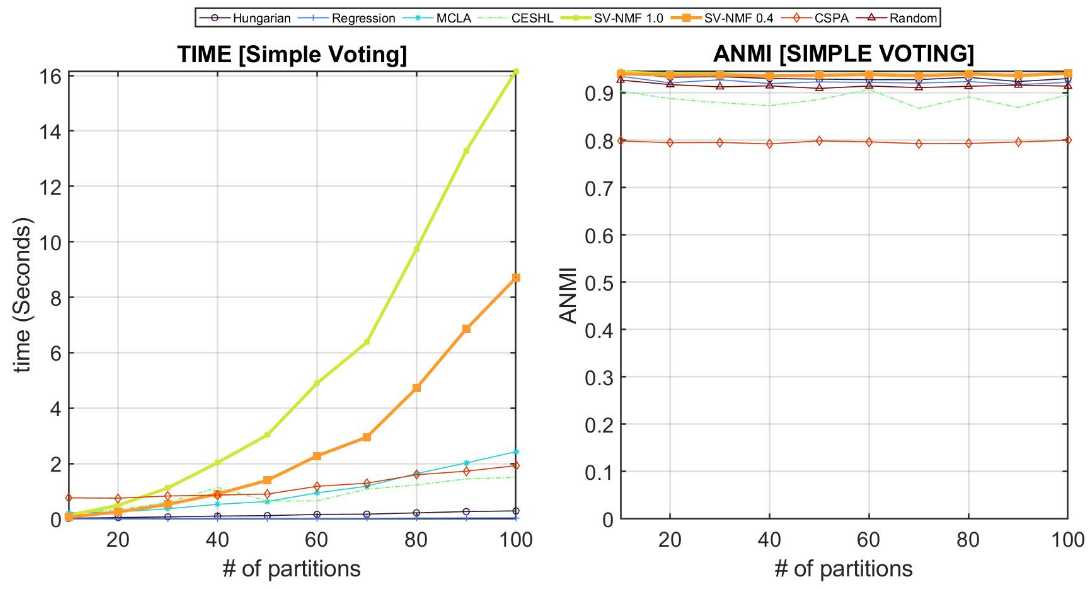


Figure 6.5: Comparison between usage of all the edges and only the 40 %

In Figure 6.5 the benefit of this customizability is shown: We analyzed the execution time and the ANMI of SV-NMF using all the edges available (presented with a dash-dotted line) and selecting only the 40% of them (presented with a solid line), to verify if there was any difference. We tested this specific type of Synchronization and Voting because it is usable in most scenarios. Furthermore all of the three methods we presented have the same graph generation and similar performance, so the results obtained in this experiment can be extended to SV-EIG and SV-QM. In the left the execution time is shown: it's clear that the version of SV-NMF that use less edges is faster than the full-edges one: there is an overall average improvement of 48%. On the right the ANMI is presented, showing that the two version of SV-NMF have similar values. Also, they are among the better performing methods.

7 | Conclusions

The Cluster Ensemble Problem can be hard to solve and there exist a variety of methods that try to solve it using a multitude of approaches, depending on the usage-scenario and the knowledge required on the data. In this chapter we summarize our discoveries and present possible future work to further investigate our results.

7.1. Contributions

We presented a novel approach for the cluster ensemble problem, called Synchronization and Voting, that takes advantage of the graph synchronization technique to assess the problem of relabeling between partitions. This approach belongs to the framework of relabeling and voting solutions, and this is the first time in literature that this type of formulation is applied to these types of methods.

From the results in Chapter 6 it's possible to see clear advantages of Synchronization and Voting in different use cases: the execution time is lower with respect to comparable methods that make use of a graph to assess the cluster ensemble problem, and can be furthermore optimized by using a subset of edges for the creation of the graph. This choice in specific scenarios doesn't alter the performance in a significant way, and it's strongly dependent on the specific technique used for the generation of the ensemble. Furthermore the overall performance in terms of NMI, ARI and accuracy is higher, particularly on datasets that present a sizable number of data points, clusters and features.

There are specific assumptions to take into account regarding the number of partitions in the ensemble: synchronization is computationally intensive, so it's not possible to use an extremely high number of partitions. There is a clear cut-off point where it's more convenient to utilize other approaches to solve the problem, but it is usually over the range of partitions used in the vast majority of real-world scenarios.

7.2. Future developments

A strong limitation for the Synchronization and Voting methods is that they can handle a limited number of partitions while keeping a reasonable and competitive range of execution time. For this reason a natural evolution is trying to tackle this limit, while remaining comparable in terms of precision and time. A possible solution could be choosing a faster relabeling algorithm that could solve the label assignment problem in a quicker way by employing the appropriate optimizations, for example the Auction algorithm [6]. In this way it's achievable to address the problem of relabeling with a different approach, possibly removing assumptions and limitations on the inputs: for example, handling partitions that are obtained with soft clustering algorithms. Another possible research is trying to adopt synchronization to boost the performance of other famous categories of cluster ensemble algorithm such as graph based algorithms, choosing a suitable group. An expansion for our method inspired by the synchronization literature is obtained using multi graphs: this approach could expand the possibility of this technique in ensemble clustering, for example by computing the permutations using different methods and then use all of this measurements at the same time in the synchronization phase.

Bibliography

[1] S. Aeberhard and M. Forina. Wine. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C5PC7J.

[2] F. Arrigoni and A. Fusiello. Synchronization problems in computer vision with closed-form solutions. *International Journal of Computer Vision*, 128:26–52, 2020.

[3] H. Ayad and M. Kamel. Finding natural clusters using multi-clusterer combiner based on shared nearest neighbors. *Multiple Classifier Systems*, pages 166–175, 2003.

[4] H. G. Ayad and M. S. Kamel. On voting-based consensus of cluster ensembles. *Pattern Recognition*, 43:1943–1953, 2010.

[5] F. Bernard, J. Thunberg, J. Goncalves, and C. Theobalt. Synchronisation of partial multi-matchings via non-negative factorisations. *Pattern Recognition*, 92:146–155, 2019.

[6] D. P. Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, pages 105–123, 1988.

[7] T. Boongoen and N. Iam-On. Cluster ensembles: A survey of approaches with recent extensions and applications. *Computer Science Review*, 28:1–25, 2018.

[8] C. Boulis and M. Ostendorf. Combining multiple clustering systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3202:63–74, 2004.

[9] M. V. Breukelen, R. P. W. Duin, D. M. J. Tax, and J. E. D. Hartog. Handwritten digit recognition by combined classifiers. *Kybernetika*, 34:381–386, 1998.

[10] L. Carlone, R. Tron, K. Daniilidis, and F. Dellaert. Initialization techniques for 3d slam: A survey on rotation estimation and its use in pose graph optimization. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4597–4604, 2015.

[11] R. Cole and M. Fanty. ISOLET. UCI Machine Learning Repository, 1994. DOI: https://doi.org/10.24432/C51G69.

[12] R. C. de Amorim, A. Shestakov, B. Mirkin, and V. Makarenkov. The minkowski central partition as a pointer to a suitable distance exponent and consensus partitioning. *Pattern Recognition*, 67:62–72, 2017.

[13] C. Domeniconi and M. Al-Razgan. Weighted cluster ensembles: Methods and analysis. *ACM Trans. Knowl. Discov. Data*, 2, 2009.

[14] R. Duin. Multiple Features. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5HC70.

[15] N. Fachada and A. C. Rosa. generatedata—a 2d data generator. *Software Impacts*, page 100017, 2020.

[16] M. Fanty and R. Cole. Spoken letter recognition. In *Advances in Neural Information Processing Systems*, volume 3, 1990.

[17] X. Z. Fern and C. E. Brodley. Solving cluster ensemble problems by bipartite graph partitioning. page 36. Association for Computing Machinery, 2004.

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

[19] R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

[20] A. Giridhar and P. R. Kumar. Distributed clock synchronization over wireless networks: Algorithms and analysis. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 4915–4920, 2006.

[21] V. M. Govindu. Lie-algebraic averaging for globally consistent motion estimation. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I, 2004.

[22] D. Greene and P. Cunningham. Efficient ensemble methods for document clustering. *University of Dublin, Trinity College, Dublin 2, Ireland*, 2013.

[23] Z. Hong and J. Yang. Lung Cancer. UCI Machine Learning Repository, 1992. DOI: https://doi.org/10.24432/C57596.

[24] Z. Q. Hong and J. Y. Yang. Optimal discriminant plane for a small number of samples and design method of classifier on the plane. *Pattern Recognition*, 24:317–324, 1991.

[25] S. Hou, L. Chen, E. Tas, I. Demihovskiy, and Y. Ye. Cluster-oriented ensemble classifiers for intelligent malware detection. *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing, IEEE ICSC 2015*, pages 189–196, 2015.

[26] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

[27] J. J. Hull. Usps, 1994. https://jundongl.github.io/scikit-feature/datasets.html.

[28] J. J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:550–554, 1994.

[29] N. Iam-on, T. Boongoen, and S. Garrett. Refining pairwise similarity matrix for cluster ensemble problem with cluster relations. In J.-F. Jean-Fran, M. R. Berthold, and T. Horváth, editors, *Discovery Science*, pages 222–233. Springer Berlin Heidelberg, 2008.

[30] N. Iam-on, T. Boongoen, and S. Garrett. Lce: a link-based cluster ensemble method for improved gene expression data analysis. *Bioinformatics*, 26:1513–1519, 2010.

[31] X. Ji, S. Liu, L. Yang, W. Ye, and P. Zhao. Clustering ensemble based on approximate accuracy of the equivalence granularity. *Applied Soft Computing*, 3:109492, 2022.

[32] R. Karp, J. Elson, D. Estrin, and S. Shenker. Optimal and global time synchronization in sensornets. *Center for Embedded Networked*, 2003.

[33] G. Karypis and V. Kumar. Kumar, v.: A fast and high quality multilevel scheme for partitioning irregular graphs. siam journal on scientific computing 20(1), 359-392. *Siam Journal on Scientific Computing*, 20, 11 1999.

[34] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in vlsi domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7:69–79, 1999.

[35] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[36] Y. LeCun and C. Cortes. Mnist handwritten digit database. *AT & T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 7, 2010.

[37] P. Mowforth and B. Shepherd. Statlog (Vehicle Silhouettes). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5HG6N.

[38] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[39] D. Pachauri, R. Kondor, and V. Singh. Solving the multi-way matching problem by permutation synchronization. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[40] C. Shao and S. Ding. Link-based cluster ensemble method for improved meta-clustering algorithm. *IFIP Advances in Information and Communication Technology*, 581 AICT:14–25, 2020.

[41] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

[42] A. Strehl. Cluster analysis and cluster ensemble software. `http://strehl.com/soft.html`.

[43] A. Strehl and J. Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617, 2003.

[44] A. Topchy, A. K. Jain, and W. Punch. A mixture model for clustering ensembles. *Proceedings of the 2004 SIAM International Conference on Data Mining (SDM)*, pages 379–390, 2004.

[45] A. P. Topchy, M. H. C. Law, A. K. Jain, and A. L. Fred. Analysis of consensus partition in cluster ensemble. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 225–232, 2004.

[46] R. Tron, X. Zhou, C. Esteves, and K. Daniilidis. Fast multi-image matching via density-based clustering. *IEEE International Conference on Computer Vision*, 2017.

[47] B. Vandeginste. Parvus: An extendable package of programs for data exploration, classification and correlation. *Journal of Chemometrics*, pages 191–193, 1990.

[48] S. Vega-Pons and J. Ruiz-Shulcloper. A survey of clustering ensemble algorithms. *IJPRAI*, 25:337–372, 2011.

[49] P. Wattuya, K. Rothaus, J. S. Praßni, and X. Jiang. A random walker based approach to combining multiple segmentations. *Proceedings - International Conference on Pattern Recognition*, 2008.

[50] Z. Yu, H.-S. Wong, and H. Wang. Graph-based consensus clustering for class discovery from gene expression data. *Bioinformatics*, 23:2888–2896, 2007.

[51] Z. Yu, L. Li, H. S. Wong, J. You, G. Han, Y. Gao, and G. Yu. Probabilistic cluster structure ensemble. *Information Sciences*, 267:16–34, 2014.

[52] L. Zhang, W. Zhou, C. Wu, J. Huo, H. Zou, and L. Jiao. Center matching scheme for k-means cluster ensembles. In M. Ding, B. Bhanu, F. M. Wahl, and J. Roberts, editors, *MIPPR 2009: Pattern Recognition and Computer Vision*, volume 7496, page 749614. SPIE, 2009.

[53] P. Zhou. Clustering ensemble via structured hypergraph learning. https://doctor-nobody.github.io/publication/paper18.

[54] P. Zhou, X. Wang, L. Du, and X. Li. Clustering ensemble via structured hypergraph learning. *Information Fusion*, 78:171–179, 2022.

[55] Z. H. Zhou and W. Tang. Clusterer ensemble. *Knowledge-Based Systems*, 19:77–83, 2006.

List of Figures

List of Tables

Acknowledgements

Vorrei ringraziare tutte le persone che mi sono state accanto in tutti gli anni di università, a partire dai miei genitori che hanno sempre fornito il loro sostegno, in tutti i sensi. Vorrei ringraziare i miei nonni, che sarebbero stati davvero felici di vedermi qui ora. Vorrei ringraziare tutte le altre persone care, amici e non, che mi sono state accanto, sia a Milano che in altre città, e con il loro supporto mi hanno regalato gioia e forza.
F.A.

Un profondo grazie alla mia famiglia per il supporto che mi ha dato in questi anni di studi universitari. In particolare ai miei genitori, per l'amore, il supporto e l'infinita pazienza che hanno avuto, aiutandomi ad ogni oscatolo e credendo sempre in me.

Un grazie a tutti i colleghi, amici e non, che mi hanno accompagnato in questi anni e con cui ho condiviso gioie e fatiche, spingendomi sempre a dare il massimo.

Un grazie agli amici di una vita con cui sono cresciuto e un grazie agli amici trovati lungo il percorso, che mi hanno sostenuto nei momenti di difficoltà e mi hanno regalato momenti di gioia.

Un ultimo grazie ai miei nonni, che sebbene lontani sono sicuro essere felici nel vedere dove sono arrivato.
Michele

Infine ringraziamo il professore Luca Magri e la professoressa Federica Arrigoni per averci accompagnato in questo percorso, per la loro disponibilità e la loro pazienza che ci hanno permesso di realizzare questa tesi.