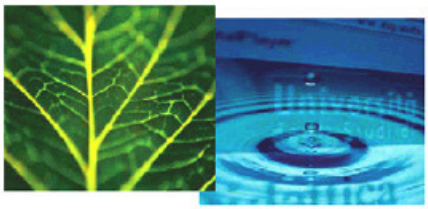
PhD Dissertation



International Doctorate School in Information and Communication Technologies

DIT - University of Trento

Non-Redundant Overlapping Clustering: Algorithms and Applications

Duy Tin Truong

Advisor:

Prof. Roberto Battiti

University of Trento

December 2013

Abstract

Given a dataset, traditional clustering algorithms often only provide a single partitioning or a single view of the dataset. On complex tasks, many different clusterings of a dataset exist, thus alternative clusterings which are of high quality and different from given trivial clusterings are asked to have complementary views. The task is therefore a clear multi-objective optimization problem. However, most approaches in the literature optimize these objectives sequentially (one after another one) or indirectly (by some heuristic combination). This can result in solutions which are not Pareto-optimal. The problem is even more difficult for high-dimensional datasets as clusters can be located in various subspaces of the original feature space. Besides, many practical applications require that subspace clusters can still overlap but the overlap must be below a predefined threshold. Nonetheless, most of the state-of-the-art subspace clustering algorithms can only generate a set of disjoint or significantly overlapping subspace clusters.

To deal with the above issues, for full-space alternative clustering, we develop an algorithm which fully acknowledges the multiple objectives, optimizes them directly and simultaneously, and produces solutions approximating the Pareto front. As for non-redundant subspace clustering, we propose a general framework for generating K overlapping subspace clusters where the maximum overlap between them is guaranteed to be below a predefined threshold. In both cases, our algorithms can be applied for several domains as different analyzing models can be used without modifying the main parts of the algorithms.

Keywords

alternative clustering; non-redundant overlapping clustering; subspace clustering; multi-objective optimization

Acknowledgements

First of all, I would like to thank Professor Roberto Battiti for being my research advisor during my career as a PhD student at University of Trento. He showed me how to do research and how to look at problems from different perspectives by which I has been got excited about doing research during my PhD study. While giving me a lot of freedom in shaping the research direction I desire to follow, he continuously provided me with valuable advices when I felt lost in keeping it up.

A special thank goes to Mauro Brunato and Andrea Passerini for their elaborated discussions and fruitful collaboration which were critical in many situations and are spread throughout this work. During my time as a graduate student at University of Trento, I had the opportunity to get acquainted with some of the finest people here at campus. I would like to thank all the past and present members of the LION lab with whom I have shared all these years of professional life. In particular, I would like to thank Umut Avci, Stefano Teso, Daniil Mirylenka, Paolo Campigotto for sharing with me my research ideas and difficulties in PhD life.

Last but not least, I am indebted to my family, especially my wife Hoa, for their support and encouragement to me during all my years of doing research. I was fortunate to have her by my side and to share with her every memorial moment of both academic and social life during all these years. The accomplishment of this work would certainly be impossible without them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The recent advances in sensing and storage technologies, and the growth of applications like internet search, video or image sharing, etc., have opened the opportunity to collect very large amounts of complex data. In addition, the variety and dimensions of available data have also increased significantly. While storing these datasets can be done efficiently, analyzing them is a challenging task. This requires the development of automatic techniques to support users in extracting knowledge from collected data.

The term of Knowledge Discovery in Database (KDD) has been introduced for this task. It involves three main steps as in Fig.1.1: preprocessing, data mining, and presenting. First, the KDD process cleanses the raw data by removing noise, irrelevant features, etc., to improve quality of the data which is used as the input for the next mining step. In the data mining step, different algorithms can be used to discover useful and hidden patterns from the data. Finally, the patterns are presented to users to help them understand the data characteristics.

This thesis focuses on developing efficient algorithms for the mining step of the KDD process. More specifically, we concentrate on the clustering problem, one of the most important tasks in data mining. Clustering is a very useful unsupervised machine learning technique for detecting the

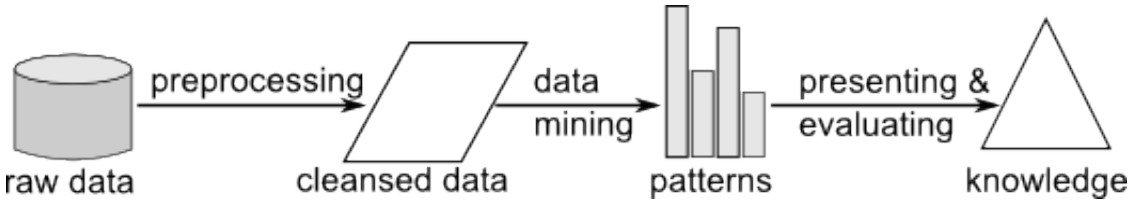


Figure 1.1: Knowledge Discovery in Database (KDD) process.

important patterns or groups without supervised information like example labels. The main goal of clustering is to partition a set of objects into different clusters, such that the objects in the same cluster are very similar to each other whereas the objects in different clusters are very dissimilar from each other. Some types of objects can be documents, images, or nodes in graphs. The feature vectors of objects or the similarities between objects (represented as a similarity graph) in a dataset can used as the input for clustering algorithms. The former type is referred to the internal representation and the latter one is called the external representation.

1.1 Non-Redundant Clustering

1.1.1 Alternative Clustering

Because of being completely unsupervised, clustering is suitable for detecting unknown patterns in areas where little expert knowledge is present. For more demanding areas where the goal is to find patterns consistent with background knowledge, the standard clustering algorithms often do not meet this goal [27, 94]. Therefore, some researchers propose to use additional information to improve the accuracy of clustering or to guide the clustering algorithm towards the desired results. This approach is named *semi-supervised* clustering (SSC). In the last ten years, SSC has received significant attention from researchers because of its success in many applications like document clustering [8, 9, 43, 52, 100], and image clustering

[44, 94]. SSC has been shown to improve the clustering performance substantially with additional instance-level constraints (declaring whether two objects should be in the same cluster or not) or object labels as side information which are provided by a supervisor. Although labels can give more information than constraints, in practical applications, constraints are easier to obtain than labels, especially if the supervisor does not know the correct label of a data object. Because of this reason, most studies in the literature focus on exploiting the knowledge from instance-level constraints to improve the clustering performance, a research branch named as *constrained clustering*.

An interesting variant of constrained clustering is *alternative clustering* or *non-redundant clustering* which imposes the non-redundancy or diversity constraints on the partitioning results [3, 42, 74]. The side information is given as a set of known clusterings, called *negative clusterings*. The goal is to find new clusterings which are different from the given negative set but still of high quality. The motivation behind alternative clustering is that each object in complex datasets can belong to many clusters, therefore each clustering only provides a single view of the dataset whereas there may exist other meaningful views. Therefore, several clustering solutions should be considered when analyzing such datasets, and the clusterings should be different from each other, and providing additional knowledge from the datasets. As users are often not good at describing what they want but easily present what they are not interested in, users can supply known clusterings as input and ask the algorithms to produce novel ones [42].

Although alternative clustering is clearly a multi-objective optimization problem, most approaches in the literature only optimize the two objectives *sequentially* (optimizing one objective first and then optimizing the other one) [25, 74] or *indirectly* by some heuristics [3, 22]. Other methods

combine the two objectives into a single one and then optimize this single objective [40, 92]. Solving a multi-objective optimization problem in the above ways can result in solutions which are not Pareto-optimal, or in a single or a very limited number of solutions on the Pareto front. The user flexibility is thus limited because the tradeoff between the different objectives is decided *a priori*, before knowing the possible range of solutions. The tradeoff can be decided in a better way *a posteriori*, by generating a large set of representative solutions along the Pareto front and having the user pick the favorite one among them. More practical approaches are *interactive* and incorporate *machine learning*: some initial information is given but "intelligent optimization" schemes collect user feedback about the initial solutions and direct the software to explore the most interesting areas of the objective space [10, 11]. In addition, most current alternative clustering algorithms can only accept one negative clustering and generate an alternative clustering. Therefore, one cannot generate a second alternative clustering by rerunning these algorithms. In order to generate a sequence of alternative clusterings, where each one is different from the other ones, a more complex algorithm which can accept a *set* of negative clusterings is required.

1.1.2 Non-Redundant Overlapping Subspace Clustering

In traditional clustering approaches, the similarity between two objects is computed by taking into account *all* features representing them, by using the Euclidean metric or generalizations thereof. However, this approach is not suitable for several real-world cases with high-dimensional feature spaces where interesting clusters can be observed in different subspaces, and where the similarity between items is not based on traditional p-norm distances like Manhattan or Euclidean.

As different clusters can exist in different subspaces, preprocessing the

datasets by feature selection techniques does not solve the problem. This motivates *subspace clustering* algorithms which aim at identifying simultaneously *both* objects in each cluster *and* the subspace of features associated with it. For this reason, subspace clusters are also termed *biclusters* [19].

Several algorithms for biclustering are based on p-norm distances [58, 68]. Three approaches can be distinguished: grid-based, density-based, and projected-subspace methods [68]. In grid-based approaches, the feature space is discretized and each subspace cluster is defined as a set of connected grid cells where each cell contains a number of objects greater than a threshold [98]. Other researchers extend the notion of density-connectivity of DBSCAN [34] for subspace clustering: clusters are specified as dense regions separated from sparse ones, and the distance between objects is computed only on the relevant dimensions of those clusters [55]. Unlike the two previous schemes searching for individual clusters, the projected-subspace approaches aim at discovering a whole set of clusters at once, by optimizing the clustering quality based on some criteria [1]. These algorithms either produce *disjoint* biclusters or *redundant* biclusters with significant overlap. In other words, these algorithms do not allow users to specify the maximum overlap between the biclusters.

However, in many practical applications like gene expression analysis in bioinformatics, a gene can be grouped into different functional categories, thus the biclusters extracted from a gene expression matrix can have overlap but the overlap should be below a predefined threshold. Otherwise, we can have redundant or very similar biclusters. This problem is termed as *non-redundant overlapping clustering*. Another issue is that the methods are often designed and optimized for specific distance metrics like Manhattan [1, 98], or Euclidean [34]. Therefore, it is difficult to apply them to new datasets or domains with different underlying models, among which gene expression analysis is a particularly prominent application. In contrast,

specialized algorithms for biclustering on gene expression data also use very particular models which cannot be generalized to other applications based p-norm distances.

1.2 Contributions and Structure of the Thesis

To deal with the suboptimal problem of alternative clustering in Section 1.1.1, we propose an explicit multi-objective algorithm with the following advantages:

- Optimizing directly and simultaneously the predefined objectives (clustering quality and dissimilarity).
- Generating a sequence of alternative clusterings where each newly generated clustering is guaranteed to be different from previous alternative clusterings.

To the best of our knowledge, this is the first time alternative clustering is solved directly as an multi-objective optimization problem. As our algorithm does not rely on special characteristics of the objective functions while searching for optimal solutions, any objective function measuring clustering quality or dissimilarity can be used. Besides, we also propose techniques for analyzing the Pareto front returned by our algorithm to help users select the preferred solutions.

For the non-redundant overlapping problem in Section 1.1.2, we introduce a flexible biclustering algorithm which can be applied to different domains without modifying its overall structure. In addition, it can sequentially generate multiple overlapping subspace clusters where the maximum overlap is below a predefined threshold. It also allows users to control bicluster shapes by adjusting the relative importance of rows and columns.

Experiments on real-world datasets demonstrate that our algorithm performs significantly better than the other state-of-the-art algorithms.

The rest of the thesis is organized as follows. We first summarize the clustering background in Chapter 2. Then, we discuss the multi-objective optimization problem of alternative clustering and introduce an algorithm for solving it in Chapter 3. The redundancy and overlap problem of subspace clustering and our proposed algorithms are presented in Chapter 4.

1.3 List of Publications

The results during my Ph.D. have been published in the following journals and papers:

- Duy Tin Truong and Roberto Battiti. **A flexible cluster-oriented alternative clustering algorithm for choosing from the Pareto front of solutions**. *Machine Learning* [ERA rank A*], 2013. [**PDF Link**]

- Duy Tin Truong, Roberto Battiti and Mauro Brunato. **Discovering Non-Redundant Biclustering on Gene Expression Data.** In *Proceedings of the 13th IEEE International Conference on Data Mining (ICDM)* [ERA rank A*, acceptance rate 11.6% for a regular paper], 2013. [**PDF Link**]

- Duy Tin Truong, Roberto Battiti and Mauro Brunato. **A Repeated Local Search Algorithm for BiClustering of Gene Expression Data**. In *Proceedings of 2nd International Workshop on Similarity-Based Pattern Analysis and Recognition (SIMBAD)* [acceptance rate 33% for a regular paper], 2013. [**PDF Link**]

- Duy Tin Truong. **A Fast and Adaptive Local Search Algorithm for Multi-Objective Optimization**. In *Proceedings of the 7th Conference of Learning and Intelligent Optimization (LION)* [acceptance rate: 48%], 2013. [**PDF Link**]

- Duy Tin Truong and Roberto Battiti. **A Cluster-Oriented Genetic Algorithm for Alternative Clustering**. In *Proceedings of the 12th IEEE International Conference on Data Mining (ICDM)* [ERA rank A*, acceptance rate: 20%], 2012. [**PDF Link**]

- Duy Tin Truong and Roberto Battiti. **A Survey of Semi-Supervised Clustering Algorithms: from a priori scheme to interactive scheme and open issues**, Technical Report number DISI-13-003, University of Trento, Italy. 2013. [**PDF Link**]

- Duy Tin Truong and Roberto Battiti. **Pareto Local Search for Alternative Clustering**, Technical Report number DISI-13-002, University of Trento, Italy. 2013. [**PDF Link**]

Chapter 2

Clustering Background

The main topic of this thesis is related to semi-supervised clustering; therefore, we summarize the traditional clustering approaches and their extensions, semi-supervised clustering techniques for integrating side information.

2.1 Hierarchical Clustering

Hierarchical clustering constructs a cluster hierarchy or a tree of clusters, also known as *dendrogram*, from data objects. Hierarchical clustering algorithms are mainly categorized into: agglomerative (bottom-up) and divisive (top-down) approach. The agglomerative approach starts with singleton clusters (each singleton cluster is a data object) and recursively merges the two most similar clusters until the desired number of clusters is achieved. In contrast, the divisive approach starts with a cluster consisting of all data objects, and successively splits each cluster into smaller clusters until a stopping condition is satisfied.

The main advantages of hierarchical clustering are:

- Ability to work with arbitrary attribute types [12].
- Ability to work with different similarity or distance functions [12].

- The hierarchical representation is informative and understandable [96].

The common disadvantages of hierarchical clustering are:

- Lack of robustness: it is sensitive to noise and outliers [96].
- It is unable to correct previous misclassification: once an object is assigned to a cluster, the hierarchical clustering algorithms will not consider this object again [12, 96].
- The computational complexity of the *classic* hierarchical clustering algorithms is at least $O(N^2)$ [12].

In the next sections, the agglomerative and divisive clustering will be discussed in detail.

2.1.1 Agglomerative Clustering

The general framework of the agglomerative clustering algorithms is summarized in Algorithm 1.

Algorithm 1: Agglomerative Clustering

Start with singleton clusters.
repeat
  Calculate the similarity between clusters.
  Merge the two closest (most similar) clusters into a new cluster.
until *the desired number of clusters is achieved*;

As can be seen from Algorithm 1, the agglomerative clustering algorithms differ from each other by their distance functions. There are two main strategies to compute the distance between two clusters: *graph methods* and *geometric methods*.

In graph methods, the distance $D(C_i, C_j)$ between two clusters C_i, C_j is calculated by considering the minimal (*single linkage*), average (*average linkage*), or maximal (*complete linkage*) distance of all object pairs (x, y)

where ($x \in C_i, y \in C_j$). SLINK [80], Voorhees' algorithm [93], CLINK [30] are one of the first algorithms in data mining which implement single linkage, average linkage, and complete linkage, respectively. In geometric methods, each cluster is represented by its geometric center and the distance between clusters is calculated based on cluster centers. This results in centroid, minimum variance linkage metrics. These all metrics are generalized by the Lance-Williams recurrence formula [60]:

$$D(C_l, (C_i \cup C_j)) = \alpha_i D(C_l, C_i) + \alpha_j D(C_l, C_j) + \beta D(C_i, C_j) + \gamma |D(C_l, C_i) - D(C_l, C_j)| \quad (2.1)$$

where $D(C_i, C_j)$ is the distance between two clusters C_1, C_2 and $\alpha_i, \alpha_j, \beta, \gamma$ are the coefficients deciding which metric to be used. Let $d(x, y)$ be the distance between two objects x, y. Some special cases of the Equation (2.1) are:

- When $\alpha_i = \alpha_j = 1/2, \beta = 0, \gamma = -1/2$, Equation (2.1) becomes the single linkage metric:

$$D(C_l, (C_i \cup C_j)) = \min(D(C_l, C_i), D(C_l, C_j)) \quad (2.2)$$
$$= \min_{x \in C_l, y \in C_i \cup C_j} d(x, y) \quad (2.3)$$

- When $\alpha_i = \alpha_j = \gamma = 1/2, \beta = 0$, Equation (2.1) becomes the complete linkage metric:

$$D(C_l, (C_i \cup C_j)) = \max(D(C_l, C_i), D(C_l, C_j)) \quad (2.4)$$
$$= \max_{x \in C_l, y \in C_i \cup C_j} d(x, y) \quad (2.5)$$

- When $\alpha_i = |C_i|/(|C_i|+|C_j|), \alpha_j = |C_j|/(|C_i|+|C_j|), \beta = -\alpha_i \alpha_j, \gamma = 0$,

Equation (2.1) becomes the centroid linkage metric:

$$
\begin{aligned}
D(C_l, (C_i \cup C_j)) &= \frac{|C_i|}{|C_i| + |C_j|} D(C_l, C_i) + \frac{|C_j|}{|C_i| + |C_j|} D(C_l, C_j) \\
&\quad - \frac{|C_i||C_j|}{(|C_i| + |C_j|)^2} D(C_i, C_j) && (2.6) \\
&= \frac{|C_i|}{|C_i| + |C_j|} (\overline{c_l} - \overline{c_i})^2 + \frac{|C_j|}{|C_i| + |C_j|} (\overline{c_l} - \overline{c_j})^2 \\
&\quad - \frac{|C_i||C_j|}{(|C_i| + |C_j|)^2} (\overline{c_i} - \overline{c_j})^2 && (2.7) \\
&= (\overline{c_l} - \frac{|C_i|\overline{c_i} + |C_j|\overline{c_j}}{|C_i| + |C_j|})^2 && (2.8) \\
&= (\overline{c_l} - \overline{c_{ij}})^2 && (2.9)
\end{aligned}
$$

where $\overline{c_i}$, $\overline{c_j}$, $\overline{c_l}$, $\overline{c_{ij}}$ are the centers of clusters C_i, C_j, C_l, $(C_i \cup C_j)$, respectively.

Surveys of linkage metrics of more complicated agglomerative clustering algorithms based on the Lance-Williams formula including group average linkage, median linkage, centroid linkage, minimum variance linkage, etc., can be found in [28, 70].

2.1.2 Divisive Clustering

The general framework of the divisive clustering algorithms is summarized in Algorithm 2. As can be seen from Algorithm 2, the main problems

Algorithm 2: Divisive Clustering

Start with a cluster consisting of all objects.

repeat

  Select a cluster in a set of current clusters to split.

  Split the selected cluster into smaller clusters.

until *the desired number of clusters is achieved*;

of divisive algorithms are to decide which cluster to split next and which

method to use for splitting it. For the first problem, simple strategies are often used: a) split every cluster at each level, b) split the clusters with the largest number of elements, c) split the clusters with the highest variance with respect to their centroids. Savaresi et al. [77] discussed the disadvantages of these strategies and proposed a better method based on cluster shapes. For the second problem, often a cluster is split into two smaller clusters and this technique is called *bisecting*. The hierarchical clusters obtained by the bisecting divisive algorithms are organized as a binary tree or a binary taxonomy. This is a very useful result in many applications like document clustering, indexing, etc., [77]. Two of the most widely used algorithms are *Bisecting K-means* [81], a special version of the standard K-means [62], and *Principal Direction Divisive Partitioning (PDDP)* [15].

The bisecting K-means algorithm is shown in Algorithm 3. In detail, given a data matrix $\boldsymbol{D} = [\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_N] \in \mathcal{R}^{M \times N}$ where each column of $\boldsymbol{D}$, $\boldsymbol{d}_i \in \mathcal{R}^M$ is a data object, and we want to split $\boldsymbol{D}$ into two sub-matrices (or sub-clusters) $\boldsymbol{D}_L$ and $\boldsymbol{D}_R$ with N_L and N_R elements and $N_L + N_R = N$. Let $\boldsymbol{c}$ be the centroid of $\boldsymbol{D}$ and be computed as $\boldsymbol{c} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{d}_i$. The K-means algorithm first picks randomly a point $\boldsymbol{c}_L$ and another point $\boldsymbol{c}_R = \boldsymbol{c} - (\boldsymbol{c}_L - \boldsymbol{c})$ in the opposite site of $\boldsymbol{c}_L$ through the centroid $\boldsymbol{c}$. Then each data point $\boldsymbol{d}_i \in \boldsymbol{D}$ is classified into the left sub-cluster $\boldsymbol{D}_L$ if $\boldsymbol{d}_i$ is closer to $\boldsymbol{c}_L$ than $\boldsymbol{c}_R$. Otherwise, that point belongs to the right sub-cluster D_R. $\boldsymbol{c}_L$ and $\boldsymbol{c}_R$ are then assigned as the centroid of $\boldsymbol{D}_L$ and $\boldsymbol{D}_R$. The algorithm repeats these two steps until $\boldsymbol{c}_L$ and $\boldsymbol{c}_R$ are not changed anymore. In contrast, the PDDP algorithm approaches the cluster selection problem by projecting all data objects in $\boldsymbol{D}$ to the principal direction, the direction on which the variance of the data object projections is maximal. The data objects are then split into two groups based on their projection values (positive or non-positive). The PDDP algorithm is presented in Algorithm

Algorithm 3: Bisecting K-means algorithm

Pick randomly a point $\boldsymbol{c}_L$, and compute $\boldsymbol{c}_R = \boldsymbol{c} - (\boldsymbol{c} - \boldsymbol{c}_L)$.
Let $\boldsymbol{c'}_L = \boldsymbol{c}_L, \boldsymbol{c'}_R = \boldsymbol{c}_R$.
repeat
  Let $\boldsymbol{c}_L = \boldsymbol{c'}_L, \boldsymbol{c}_R = \boldsymbol{c'}_R$.
  if $|\boldsymbol{d}_i - \boldsymbol{c}_L| \leq |\boldsymbol{d}_i - \boldsymbol{c}_R|$ **then**
    assign $\boldsymbol{d}_i$ to $\boldsymbol{D}_L$
  else
    assign $\boldsymbol{d}_i$ to $\boldsymbol{D}_R$
  end
  $\boldsymbol{c'}_L = \frac{1}{N_L} \sum_{\boldsymbol{d}_i \in \boldsymbol{D}_L} \boldsymbol{d}_i, \boldsymbol{c}'_R = \frac{1}{N_R} \sum_{\boldsymbol{d}_i \in \boldsymbol{D}_R} \boldsymbol{d}_i$
until $\boldsymbol{c}_L = \boldsymbol{c'}_L$ *and* $\boldsymbol{c}_R = \boldsymbol{c'}_R$;

4. A comparative study of the bisecting K-means and PDDP algorithm

Algorithm 4: Principal Direction Divisive Partitioning (PDDP) algorithm

1. Compute the centroid $\boldsymbol{c}$ of $\boldsymbol{D}$: $c = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{d}_i$.
2. Let $\boldsymbol{D'} = \boldsymbol{D} - \boldsymbol{ce}$ where $\boldsymbol{e}$ is a N-dimensional row vector ones, i.e. $\boldsymbol{e} = [1, 1, .., 1]$.
3. Compute the SVD of $\boldsymbol{D'} = \boldsymbol{U\Sigma V^T}$.
4. Let $\boldsymbol{u}$ be the first column of $\boldsymbol{U}$.

$\forall \boldsymbol{d}_i \in \boldsymbol{D} : \boldsymbol{d}_i$ is assigned to $\boldsymbol{D}_L$ if $\boldsymbol{u^T}(\boldsymbol{d}_i - \boldsymbol{c}) \leq 0$. Otherwise, $\boldsymbol{d}_i$ belongs to $\boldsymbol{D}_R$.

can be found in [76].

2.2 Partitioning Clustering

Instead of providing a tree of clusters as the hierarchical clustering, the partitioning clustering splits the set of data objects into K disjoint clusters where K is provided by the user. Depending on the representation of data objects (*internal representation* or *external representation*), there are different approaches for partitioning clustering like *objective function*, *probabilistic*, or *graph-theoretic* approach.

In the *internal representation*, each data object d is represented by an M-dimensional vector $\boldsymbol{x}_d$ and based on this representation, the similarity or dissimilarity between data objects can be derived. In this case, one common approach is to define an objective function to measure the clustering quality. The clustering problem is now transformed into an optimization problem where the goal is to search for the cluster partitioning with the minimal (or maximal) objective value. An alternative approach in this representation is the probabilistic approach in which each cluster is modeled by a probabilistic distribution. The probabilistic approach assumes that the data objects are generated from a mixture of probabilistic models whose distributions are unknown and the data partitioning problem becomes the problem of identifying such distributions. In the case of *external representation*, there is no vector representation for each data object, only the similarity or dissimilarity between data objects is available. Therefore, the graph-theoretic approach is more suitable in this case and the clustering result is an undirected and weighted graph of data objects.

Some representative algorithms of three above approaches will be discussed in the next sections.

2.2.1 Optimization-Based Clustering

In this approach, the objective function is the most important factor for the success of the process. One of the most widely used objective function is the *squared error* function. In detail, given a set of N data objects $\boldsymbol{x}_j \in \mathcal{R}^d, j = 1, ..., N$ and the goal is to split them into K clusters $C = \{C_1, ..., C_K\}$. The within-cluster S_W and between-cluster square distance S_B [33] are defined as:

$$S_W(C) = \sum_{i=1}^{K} \sum_{\boldsymbol{x}_j \in C_i} |\boldsymbol{x}_i - \boldsymbol{c}_i|^2 \tag{2.10}$$

$$S_B(C) = \sum_{i=1}^{K} N_i(\boldsymbol{c}_i - \boldsymbol{c})(\boldsymbol{c}_i - \boldsymbol{c})^T \tag{2.11}$$

where $\boldsymbol{c}_i = \frac{1}{N_i} \sum_{\boldsymbol{x}_j \in C_i} \boldsymbol{x}_j$, $\boldsymbol{c} = \frac{1}{N} \sum_{j=1}^{N} \boldsymbol{x}_j$, $N_i = |C_i|$. The clustering algorithms now try to minimize the within-cluster square distance S_W and maximize the between-cluster square distance S_B.

The most well-known and simplest partitioning clustering algorithm is *K-Means* [62]. The K-Means algorithm searches for K disjoint clusters such that the within-cluster square distance is minimized. The K-Means algorithm is presented in Algorithm 5.

Algorithm 5: K-Means

Initialize K cluster centers randomly or based on some heuristics.

repeat

Assign each data object $\boldsymbol{x}_i$ to the nearest center.

Update each cluster center $\boldsymbol{c}_i$ as the mean of all data object belongs to that cluster, i.e. $\boldsymbol{c}_i = \frac{1}{N_i} \sum_{\boldsymbol{x}_j \in C_i} \boldsymbol{x}_j$.

until *there is no change for each cluster or a maximum number of iterations are executed*;

Firstly, K-Means initialize K cluster centers randomly or based on heuristics. Then it assigns each data object $\boldsymbol{x}_i$ to the nearest center among K centers. Next, each center is updated as the mean of all data objects assigned to that cluster. K-Means repeats the two previous steps until there is no change in each cluster or until a maximum number of iterations are executed. Because K-Means is very simple and easy to implement, it has been used in many practical problems [96]. The time complexity of K-Means is $NKdL$ where L is the number of iterations. Usually, K and d are much smaller compared to N, hence K-Means can work well with large datasets. Also, the parallel version of K-Means was developed by Stoffel et al. in [82]. However, K-Means has the following disadvantages:

1. K-Means strongly depends on the cluster initialization. The general strategy is to run K-Means many times and select the best clustering. Some classical cluster initialization methods are:

   - Random: pick randomly K points as K centers.
   - Forgy's method [37]: pick randomly K data objects as K centers.
   - Macqueen's method [64]: pick randomly K data objects as K centers. Then following the data object order, assign each data object to the cluster with the nearest center. After each assignment, the center is recalculated as the mean of the data objects in that cluster.
   - Kaufman's method [56]: in this method, K centers are selected sequentially. The first cluster is chosen as the most centrally located data object in the dataset. Then, each next center is one of the remaining data objects with the highest number of data objects around it estimated by a heuristic function.

   The experimental results on the above four methods [73] show that the random and Kaufman's method outperform the other two methods with respect to the effectiveness and robustness of the K-Means algorithm. Also, considering the converge speed, the Kaufman's method is recommended by the authors.

2. K-Means requires the number of clusters K is known in advance, but identifying an effective K is non-trivial. One of the attempts trying to determine the optimal K is the ISODATA algorithm [5]. The ISODATA algorithm merges and splits the intermediate clusters based on some predefined thresholds. However, this means that the problem of identifying the number of clusters turns into the problem of tuning the threshold parameters.

3. K-Means suffers from the problem of local minima. A solution is to formulate the clustering problem as a combinatorial optimization problem and apply directly well-known combinatorial optimization techniques like simulated annealing [18], genetic algorithms [59].

4. K-Means is sensitive to outliers. Because every data object is assigned to a cluster and a cluster center is the mean of all data objects in that cluster, some noise data objects which are very far from the correct cluster center can destroy the K-Means algorithm. The K-Medoids algorithms [35, 56, 91] solve this problem by choosing the data objects in the dataset as cluster prototypes.

2.2.2 Probabilistic Clustering

In probabilistic clustering, the data objects $\{x_1, ..., x_N\}$ are assumed to be generated from a mixture of probabilistic models. The generation process of each data object is as follows: first, a model (or a class) $c_j, 1 \leq j \leq K$ is picked with the probability of p_j and then a data object x_i is drawn from the model c_j. Let θ_j be the parameters of the model c_j and $\Theta = (p_1, ..., p_K, \theta_1, ..., \theta_K)$ be the parameters of the mixture model. The likelihood of the dataset is the probability that the dataset is drawn from that given mixture model:

$$P(X|C) = \prod_{i=1}^{N} \sum_{j=1}^{K} P(x_i|c_j, \Theta) p_j \tag{2.12}$$

Usually, the log-likelihood $L(X|C) = \log P(X|C)$ is used as an objective function and the goal is to find the parameter Θ^* such that the log-likelihood is maximum. The popular algorithm for searching such parameter Θ^* is the *Expectation-Maximization* (EM) algorithm [32, 66]. First, the EM algorithm initializes the parameter Θ as Θ^0 and then it repeats the two steps E and M until the convergence conditions are satisfied. The

two steps are as follows. Step E computes $P(c_j|x_i, \Theta^t)$ given Θ^t. Step M updates Θ^{t+1} based on $P(c_j|x_i, \Theta^t)$ evaluated at Step E.

Let's assume that the probability distributions $p(x_i|c_j, \Theta)$, $p(c_j|x_i, \Theta)$ are normal and each model c_j is parameterized by its mean μ_j and variance Σ_j, then $\Theta = (p_1, ..., p_K, \mu_1, ..., \mu_K, \Sigma_1, ..., \Sigma_K)$. The EM algorithm of Gaussian mixtures [14] is illustrated as in Algorithm 6. Because EM

Algorithm 6: The EM algorithm for Gaussian Mixtures

Initialize Θ as Θ^0 and calculate the log-likelihood.

repeat

Step E: Evaluate the probabilities:

$$w_{ji} = P(c_j|x_i, \Theta^t) = \frac{P(x_i|c_j, \Theta^t)p_j}{\sum_{k=1}^{K} P(x_i|c_k, \Theta^t)p_k} \tag{2.13}$$

Step M: update the parameters:

$$\mu_j = \frac{1}{N_j} \sum_{i=1}^{N} w_{ji} x_i \tag{2.14}$$

$$\Sigma_j = \frac{1}{N_j} \sum_{i=1}^{N} w_{ji} (x_i - \mu_j)(x_i - \mu_j)^T \tag{2.15}$$

$$p_j = \frac{N_j}{N} \tag{2.16}$$

where

$$N_j = \sum_{i=1}^{N} w_{ji} \tag{2.17}$$

Calculate the log-likelihood and check convergence conditions.

until *the convergence conditions are satisfied*;

iterates through every data object-class pair and each data object is a M-dimensional vector, therefore its complexity is $O(LM^2NK)$ where L is the number of iterations that EM performed. In order to reduce such computational cost, Moore [67] suggested using the KD-Tree data structure to organize data objects. The root node keeps all data objects and at each

non-leaf-node, data objects are divided into two its children by splitting at the center of the widest attribute. A node is considered as a leaf and left un-split if the widest attribute of the data objects in that node is less than some threshold. The E and M steps are performed at the root node after accumulating sufficient statistic information (e.g., w_{ij}) from its two children. Each child node recursively sums up the statistic information from its children. At leaf nodes, the statistic information of all data objects are approximated as the statistic information of the centroid. This causes the reduction in the computational cost. Also, a non-leaf-node is pruned and considered as a leaf node if the difference between the lower bound and upper bound of the statistic information of all data objects in this node is less than some threshold.

2.2.3 Graph-Theoretic Clustering

One of the well-known graph theoretic clusterings is Zhan's algorithm [99] which first builds the minimum spanning tree of the dataset and then removes the longest edges to split the minimum spanning tree into sub-trees where each sub-tree forms a cluster. Similarly, Hartuv and Shamir [49] consider clusters as highly-connected sub-graphs with the edge-connectivity (the minimum number of edges whose removal makes the graph disconnected) greater than half of the vertex number of the graph. Recursively, a graph is checked if it is highly connected. If this condition holds, the graph is returned, otherwise it is split into two sub-graphs by a minimum cut procedure which seeks for the smallest number of edges to remove in order to disconnect the graph. Another algorithm called CLICK [78] also uses minimum weight cut to partition data objects into clusters. CLICK first builds a weighted graph of data objects where edge weight w_{ij} between two data objects i and j reflects the probability that i and j are in the same cluster (i and j in this case are called *mates*) and it is defined as

follows:

$$w_{ij} = \log \frac{p_{mates} p(S_{ij}|\mu_T, \sigma_T)}{(1 - p_{mates}) p'(S_{ij}|\mu_F, \sigma_F)} \tag{2.18}$$

where p_{mates} is the probability that two randomly picked data objects are in the same cluster. $p(S_{ij}|\mu_T, \sigma_T)$ and $p'(S_{ij}|\mu_F, \sigma_F)$ are the probability that i and j are mates and not mates, respectively, given the similarity S_{ij} between them. The mean and deviation $(\mu_T, \sigma_T), (\mu_F, \sigma_F)$ of these normal distributions are estimated by prior knowledge or experiments. Then, CLICK recursively splits the graph by a minimum weight cut procedure into two sub-graphs until some stopping conditions are satisfied. Besides, as the number of sub-graphs can be different from the desired number of clusters, CLICK merges similar sub-graphs to generate the requested number of clusters.

2.3 Semi-Supervised Clustering

Semi-supervised clustering (SSC) is the problem of clustering unlabelled data with the support of *side information* provided by a supervisor (who can be an expert or an oracle system). Because of its great success in recent years, SSC has received significant attention from researchers. The side information has been shown to guide the clustering algorithms towards the desired clustering solutions or to help the clustering algorithms escape from local minima. The side information does contribute not only to the performance improvement but also to the complexity reduction. An example is the car land identifying problem from GPS data where the goal is to cluster data points into different lanes [94]. This is a difficult clustering problem for the well-known clustering algorithm K-Means because the lane clusters have a very special shape which is very elongated and parallel to the road centerline. In some experiments, the accuracy of K-Means inte-

grating constraints is 98.6% whereas that of K-Means without constraints is only 58% [94]. Other applications of SSC can be found in [27].

The SSC algorithms can be classified into one of the following two schemes: the a-priori scheme, the interactive scheme. In the a-priori scheme, the side information is given once before executing the SSC algorithm while in the interactive scheme, the side information is collected iteratively by interacting with the supervisor.

2.3.1 A Priori Scheme

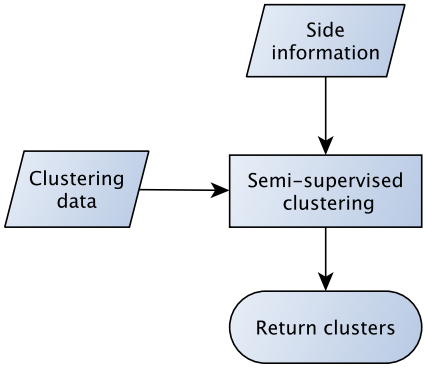


Figure 2.1: A Priori Scheme

In the a priori scheme (shown in Fig. 2.1), the SSC algorithm reads all side information once and uses these information to improve the clustering performance. Many works following this scheme have been done in literature and split into different types of side information like labelled data, instance-level or cluster-level constraints.

Several techniques utilizing the side information in the form of labeled data are proposed like:

- *Seeded K-Means* uses labeled examples to initialize the cluster centers [7].
- *Constrained K-Means* also initializes the cluster centers by labelled data like *Seeded K-Means* but it keeps the labels of examples in the

side information unchanged in the assignment step of the clustering process [7].

The algorithms which uses instance-level constraints provided by users to improve the clustering performance are classified into three groups:

- Constraint-Based Clustering: in this group, the original clustering algorithms are modified to integrate the constraints, e.g., in a constrained agglomerative clustering algorithm, two clusters are only merged if the merging does not violate constraints [27], or by adding the penalty of violating the constraints into the objective function of K-Means [8, 26, 72]. The clustering solutions must satisfy completely the constraints [79, 94] or some constraints can be violated [8, 26, 72]. Also, two dominant approaches of the algorithms in this group consist of extending the objective function of K-Means for integrating constraints [8, 26, 72] or of adding constraints into prior distributions of probabilistic clustering frameworks. The clustering solutions which satisfy the constraints are given higher scores to be selected [9, 63, 79].

- Distance-Based Clustering: in this group, only the distance metric is changed such that if two points are constrained to be in the same cluster, their distance should be smaller than the distance of two points constrained to be in different clusters [50, 95].

- A unified framework for constraint-based and distance-based clustering is also proposed by Basu et al. [9].

Finally, cluster-level constraint based algorithms can be divided into two main classes:

- Balanced-Clustering: where the constraint is that the variance of cluster sizes is as small as possible. Two algorithms for this problem are proposed by Banerjee et al. [6] and Demiriz et al. [31].

- Non-Redundant or Alternative Clustering: in this formulation, the constraint is given as a clustering result and the goal is to find the new clustering result which is high-quality and different from the given one. This problem was first introduced by Goldek et al. [41, 42].

2.3.2 Interactive Scheme

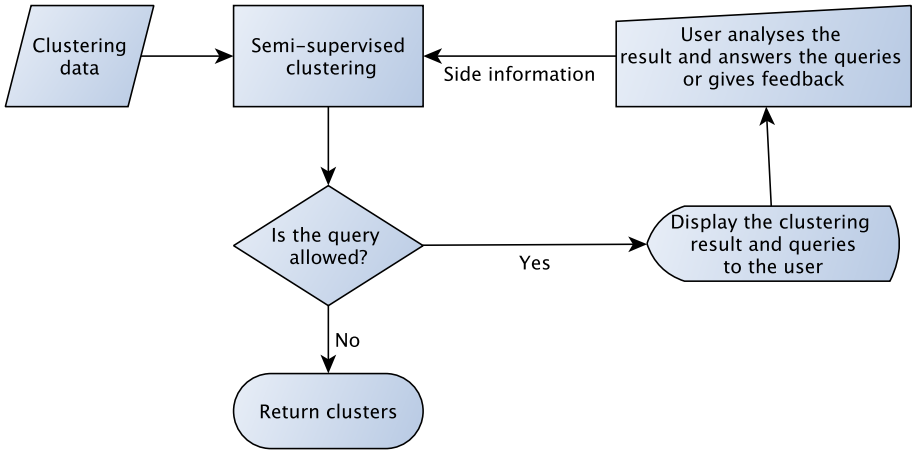


Figure 2.2: Interactive Scheme

In the interactive scheme (illustrated in Fig. 2.2), the SSC algorithm presents the clustering result and a query to a supervisor who can be a user or an oracle system. Then the supervisor provides feedback to the SSC algorithm. The SSC algorithm analyses the feedback and adapts this information to bias the clustering process. The interaction between the SSC algorithm and the supervisor is stopped when some convergence condition is satisfied. The feedback can be collected in two following ways based on the role of the supervisor and the SSC algorithm. If the supervisor plays the active role, then he actively provides the constraints to the SSC algorithm. In the case that the SSC algorithm has the active role, the SSC algorithm will pose queries to the supervisor, and the supervisor is supposed to answer. The second approach has been shown to outperform the first approach in literature. The reason is that the first approach

requires that the supervisor must know which are the most informative constraints to supply for the SSC algorithm while in the second approach, this difficult task is on the side of the SSC algorithm, and it is better if the SSC algorithm is allowed to ask what it is not clear than passively receives irrelevant feedback from the supervisor. The algorithms in the first approach will be referred as the *passive* SSC algorithms, while the ones in the second approach will be called the *active* SSC algorithms.

So far, only few works have been done in this scheme. Interactive SSC algorithms can integrate the constraints by changing the distance metric [20, 21] and use the farthest distance [8], information gain [52], density [100] and co-association confidence [43] to select the most informative constraints.

Chapter 3

Alternative Clustering

Supervised alternative clusterings is the problem of finding a set of clusterings which are of high quality *and* different from a given *negative* clustering. The task is therefore a clear multi-objective optimization problem. Optimizing two conflicting objectives at the same time requires dealing with trade-offs. Most approaches in the literature optimize these objectives sequentially (one objective after another one) or indirectly (by some heuristic combination of the objectives). Solving a multi-objective optimization problem in these ways can result in solutions which are dominated, and not Pareto-optimal. We develop a direct algorithm, called **COGNAC**, which fully acknowledges the multiple objectives, optimizes them directly and simultaneously, and produces solutions approximating the Pareto front. **COGNAC** performs the recombination operator at the *cluster level* instead of at the object level, as in the traditional genetic algorithms. It can accept arbitrary clustering quality and dissimilarity objectives and provides solutions dominating those obtained by other state-of-the-art algorithms. Based on **COGNAC**, we propose another algorithm called **SGAC** for the sequential generation of alternative clusterings where each newly found alternative clustering is guaranteed to be different from all previous ones. The experimental results on widely used benchmarks

demonstrate the advantages of our approach. The source codes, datasets and other supplementary materials of our experiments can be found at `http://lion.disi.unitn.it/intelligent-optimization/cognac/`.

The content of this chapter is extracted from our published paper [87].

3.1 Introduction

Given a dataset, traditional clustering algorithms often only provide a single set of clusters, a single view of the dataset. On complex tasks, many different ways of clustering exist, therefore a natural requirement is to ask for *alternative* clusterings, to have complementary views. Clustering flowers depending on colors and aesthetic characteristics can be suitable for a florist, but not for a scientist, who usually prefers different organizations.

Recently, many techniques have been developed for solving the *alternative clustering* problem. They can be split into two groups: unsupervised or supervised. In unsupervised alternative clustering, the algorithm automatically generates a set of alternative clusterings of high quality and different from each other [13, 23, 24, 45, 54, 71]. Unsupervised alternative clustering is useful if users do not know what they want and need some initial options. In other cases, users already know some trivial or *negative* clusterings, and they ask for different and potentially more informative clusterings. These algorithms are called supervised because the user is directing the alternative clustering by explicitly labeling some clusterings as undesired, or *negative*.

This chapter focuses on *supervised alternative clustering*, the problem of finding new clusterings of good quality *and* as different as possible from a given negative clustering. Supervised alternative clustering is a multi-objective optimization problem with two objectives of clustering quality and dissimilarity, and the goal is to find a representative set of Pareto-

optimal solutions. A Pareto-optimal solution is a solution such that there is no solution which improves at least one objective without worsening the other objectives. The Pareto front is the set of all Pareto-optimal solutions in the objective space. Most approaches in the literature only optimize the two objectives *sequentially* (optimizing one objective first and then optimizing the other one) [25, 74] or *indirectly* by some heuristics [3, 22]. Other methods combine the two objectives into a single one and then optimize this single objective [40, 92].

Solving a multi-objective optimization problem in the above ways can result in solutions which are not Pareto-optimal, or in a single or a very limited number of solutions on the Pareto front. The user flexibility is thus limited because the tradeoff between the different objectives is decided *a priori*, before knowing the possible range of solutions. The tradeoff can be decided in a better way *a posteriori*, by generating a large set of representative solutions along the Pareto front and having the user pick the favorite one among them. More practical approaches are *interactive* and incorporate *machine learning*: some initial information is given but "intelligent optimization" schemes collect user feedback about the initial solutions and direct the software to explore the most interesting areas of the objective space [10, 11].

Some approaches are developed for specific clustering algorithms, and are therefore limited in their applications, or require setting parameters which influence the preference between clustering quality and dissimilarity. Parameter tuning by the final user is a difficult task since some dominating solutions can be lost because of improper settings.

In addition, most current alternative clustering algorithms can only accept one negative clustering $\overline{\mathbf{C}}$ and generate an alternative clustering $\mathbf{C}_1$ different from $\overline{\mathbf{C}}$. Therefore, one cannot generate a second alternative clustering $\mathbf{C}_2$ by simply rerunning the algorithm with $\mathbf{C}_1$ as the negative

clustering. The second alternative clustering $\mathbf{C}_2$ will be different from $\mathbf{C}_1$ but often very similar to $\overline{\mathbf{C}}$, because $\overline{\mathbf{C}}$ is not considered when computing $\mathbf{C}_2$. In order to generate a sequence of alternative clusterings, where each one is different from the other ones, a more complex algorithm which can accept a *set* of negative clusterings is required.

To deal with the above issues, we propose an explicit multi-objective algorithm, called **C**luster-**O**riented **G**e**N**etic algorithm for **A**lternative **C**lusterings (**COGNAC**), with the following advantages:

- Optimizing directly and simultaneously the predefined objectives (clustering quality and dissimilarity).
- Generating a sequence of alternative clusterings where each newly generated clustering is guaranteed to be different from previous alternative clusterings.

In [86], we introduced the basic version of **COGNAC** which is limited to the case where the number of clusters in both negative clusterings and alternative clusterings are the same. In this chapter, we extend **COGNAC** flexibility to handle cases where the number of clusters can be different and propose a new algorithm called **SGAC** for the sequential generation of alternative clusterings. We also propose techniques for analyzing the Pareto front returned by **COGNAC** to help users select the preferred solutions. Moreover, we present detailed experiments with a thorough analysis on the Pareto solutions and compare them with the *ground truth* alternative solutions.

The rest of this chapter is organized as follows. We first formally define the alternative clustering problem in Section 3.3.1. In Section 3.3.2, we summarize the non-dominated sorting framework of Deb et al. [29] and then detail our algorithm **COGNAC**. We then propose the new algorithm **SGAC** in Section 3.3.3. Some techniques for analyzing and visualizing the

Pareto front returned by our algorithms are presented in Section 3.3.4. We describe the test datasets in Section 3.4.1 and the experiments to compare the performance of our algorithm with that of other state-of-the-art proposals on the first alternative clustering in Section 3.4.2. Then, we illustrate the ability of our algorithm for generating a sequence of different alternative clusterings in Section 3.4.3. Finally, we analyze the parameter sensitivity of our method in Section 3.4.4.

3.2 Related Work

Among the first algorithms in supervised alternative clustering is Conditional Information Bottleneck (**CIB**) [40], based on the information bottleneck (IB) method [85]. Their approach in modelling the clustering problem is similar to that of data compression. Given two variables X representing objects and Y representing features, and a negative clustering Z, the **CIB** algorithm finds an alternative clustering C which is different from Z but still good in quality by maximizing the mutual information $I(C;Y|Z)$ between C and Y given Z under the constraint that the mutual information (or information rate) $I(C;X)$ between C and X is less than a threshold R. However, this approach requires an explicit joint distribution between the objects and the features which can be very difficult to estimate.

Bae et al.[3] show that the clusterings found by **CIB** are not of high quality if compared with those found by their **COALA** algorithm, which extends the agglomerative hierarchical clustering algorithm. Let d_1 be the smallest distance between two arbitrary clusters and d_2 be the smallest distance between two clusters where merging them does not violate the constraints (generated by the negative clustering). If the ratio $\frac{d_1}{d_2}$ is less than a threshold, then two nearest clusters are merged to preserve the clustering quality. Otherwise, two clusters with the distance of d_2 are

merged to find dissimilar clusterings. A drawback is that it only considers *cannot-link* constraints, hence useful information which can be obtained through *must-link* constraints is lost. In addition, the application scope of the method is limited because it was developed particularly for the agglomerative clustering algorithms.

To overcome the scope limitation, Davidson et al.[25] propose a method, called **AFDT** which transforms the dataset into a different space where the negative clustering is difficult to be detected and then uses an arbitrary clustering algorithm to partition the transformed dataset. However, transforming the dataset into a different space can destroy the characteristics of the dataset. Qi et al.[74] fix this problem by finding a transformation which minimizes the Kullback-Leibler divergence between the probability distribution of the dataset in the original space and the transformation space, under the constraint that the negative clustering should not be detected. This approach requires specifying user preference at the clustering quality and the clustering dissimilarity. In this thesis, we will refer this approach as **AFDT2**.

Alternative clustering can also be discovered by two orthogonalization algorithms proposed in [22]. The algorithms first project the dataset into an orthogonal subspace and then apply an arbitrary clustering algorithm on the transformed dataset. However, when the objects of the dataset are in low-dimensional spaces, the orthogonal subspace may not exist [25]. In addition, a requirement of Cui et al.'s algorithms which is the number of clusters must be smaller than the number of dimensions in the original dataset may not be satisfied in many practical cases. In fact, Davidson et al.[25] show that **AFDT** outperforms Cui et al.'s algorithms.

The above algorithms can only accept one negative clustering. Recently, Nguyen et al. [92] propose **MinCEntropy**$^{++}$, which can accept a set of $N_{\overline{C}}$ negative clusterings. **MinCEntropy**$^{++}$ finds an alternative clustering

$\mathbf{C}_*$ by maximizing the weighted sum:

$$N_{\overline{C}}\mathbb{I}(\mathbf{C};\mathbf{X}) - \lambda \sum_{i=1}^{N_{\overline{C}}} \mathbb{I}(\mathbf{C};\overline{\mathbf{C}}_i)$$

where $\mathbb{I}(\mathbf{C};\mathbf{X})$ is the mutual information between a clustering $\mathbf{C}$ and the dataset $\mathbf{X}$; $\mathbb{I}(\mathbf{C};\overline{\mathbf{C}}_i)$ is the mutual information between a clustering $\mathbf{C}$ and a negative clustering $\overline{\mathbf{C}}_i$; λ is the parameter trading-off the importance between clustering quality and dissimilarity.

Dasgupta et al.[24] propose an unsupervised algorithm for generating a set of alternative clusterings. In this thesis, we will refer this algorithm as **Alter-Spect** because it is based on the spectral clustering algorithm. **Alter-Spect** first forms the Laplacian matrix L of the dataset and then computes the second through $(m+1)$-th eigenvectors of L where m is the number of alternative clusterings that users want to generate. Then, it runs **K-Means** on these eigenvectors to produce m different alternative clusterings. The main intuition of **Alter-Spect** and other subspace clustering approaches [13, 45, 71] is that different alternative clusterings can exist in different subspaces. In contrast, our algorithm considers all features and optimizes two objectives in parallel. In detail, Nui et al. suggest an algorithm which learns low-dimensional subspaces for different views by optimizing a fixed single objective which is the combination of two alternative clustering objectives [71], i.e., the quality of all clustering is as high as possible, and the redundancy between them is as low as possible. Günnemann et al. also propose a Bayesian framework for determining different views in subspaces [45]. The authors generalize the dataset by using multiple mixture models where each mixture of Beta distributions presents a specific view. As these mixtures can compete against each other in the data generation process, their framework can handle overlapping views and subspace clusters. Instead of generating alternative clusterings as in other approaches, De Bie [13] describes an algorithm which generates a sequence

of different clusters. In each iteration, the algorithm searches for the next cluster surprising the user most, given the previous clusters. The user surprise is inversely proportional to the probability that the new cluster appears under a predefined data distribution. However, as this framework only focuses on the surprise aspect of the next cluster but does not consider its quality, a poor quality but highly-surprising cluster can be returned.

For optimizing directly the clustering objectives, Handl et al. [47] propose an evolutionary algorithm for multi-objective clustering, called **MOCK**, which uses the graph-based representation to encode the clustering solutions. Each clustering solution $\mathbf{C}_t$ of N data objects $\{\mathbf{x}_i\}_{i=1}^{N}$ is represented as a N-dimensional vector and the i-th element $\mathbf{C}_t(i)$ stores the index of the object $\mathbf{x}_j$ to which the i-th object $\mathbf{x}_i$ is connected. All data objects in the same connected components are extracted to form clusters. The clustering solution returned by **MOCK** can have an arbitrarily large number of clusters because the number of connected components can be varied from 1 to N. The number of clusters in alternative clustering is often fixed to make it easier to compare different clusterings. Actually, this is necessary when comparing two clusterings on quality objectives like Vector Quantization Error (VQE) of **K-Means** [62], because the value of VQE decreases when the number of clusters increases. However, fixing the number of clusters decreases **MOCK** performance in a radical way, because many clustering solutions become invalid and are discarded when applying the standard procedures of initialization, recombination, and mutation. Therefore, it is difficult to extend **MOCK** for the alternative clustering problem to compare with our algorithm. This is the reason why we will only compare our algorithm against **COALA**, **AFDT**, **AFDT2**, **MinCEntropy**$^{++}$ and **Alter-Spect**.

3.3 A Cluster-Oriented Genetic Algorithm for Alternative Clustering

In this section, we first formally define the problem of alternative clustering in Section 3.3.1 and then detail our algorithm **COGNAC** to address such problem in Section 3.3.2. Then, based on **COGNAC**, we propose another algorithm (**SGAC**) for generating a set of different alternative clusterings in Section 3.3.3. Some techniques for analyzing and visualizing the Pareto front returned by our algorithm are also presented in Section 3.3.4.

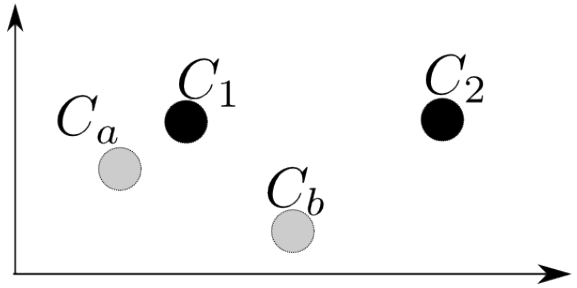
3.3.1 The Alternative Clustering Problem

Given a dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ of N objects, the traditional clustering problem is to partition this dataset into K disjoint clusters such that the clustering quality is as high as possible. Let $\mathbf{C}$ be a clustering solution where $\mathbf{C}(i)$ is the index of the cluster that $\mathbf{x}_i$ belongs to, $\mathbb{D}(\mathbf{C}, \overline{\mathbf{C}})$ be the dissimilarity between two clusterings $\mathbf{C}$ and $\overline{\mathbf{C}}$, and $\mathbb{Q}(\mathbf{C})$ be the inner quality of a clustering $\mathbf{C}$. We defer the definition of $\mathbb{D}(\mathbf{C}, \overline{\mathbf{C}})$ and $\mathbb{Q}(\mathbf{C})$ to Section 3.3.2 where we also present other components of our algorithm and define in this section the dissimilarity between a clustering and a set of clusterings, the overall quality of a clustering and the dominance relation between two clusterings.

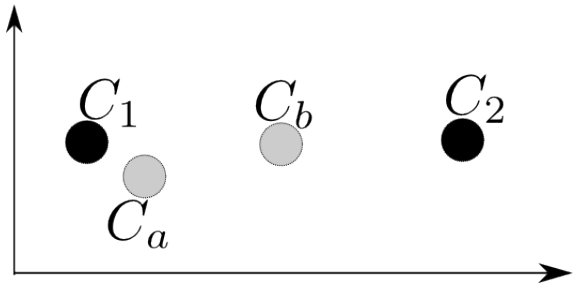
Definition 1 *(Dissimilarity) The dissimilarity between a clustering* $\mathbf{C}$ *and a set* $\overline{\mathbf{S}}$ *of clusterings is the minimum dissimilarity between* $\mathbf{C}$ *and all clusterings* $\overline{\mathbf{C}} \in \overline{\mathbf{S}}$*:*

$$\mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) = \min_{\overline{\mathbf{C}} \in \overline{\mathbf{S}}} \mathbb{D}(\mathbf{C}, \overline{\mathbf{C}}) \tag{3.1}$$

In Fig.3.1a and Fig.3.1b, we illustrate the benefit of using minimum dissimilarity over maximum dissimilarity and average dissimilarity in defining



(a) Maximum dissimilarity prefers solution $\mathbf{C_a}$ while minimum dissimilarity prefers solution $\mathbf{C_b}$.



(b) Average dissimilarity prefers solution $\mathbf{C_a}$ while minimum dissimilarity prefers solution $\mathbf{C_b}$.

Figure 3.1: The benefit of using minimum dissimilarity over maximum and average dissimilarity.

the dissimilarity between a clustering and a clustering set. Assume that the clustering set is $\overline{\mathbf{S}} = \{\mathbf{C_1}, \mathbf{C_2}\}$ and we want to select a clustering between two clusterings $\mathbf{C_a}$ and $\mathbf{C_b}$ based on the dissimilarity. As it can be seen in Fig.3.1a, although $\mathbf{C_a}$ is very similar (or can be equal to) $\mathbf{C_1}$ but its maximum dissimilarity to the clustering set $\overline{\mathbf{S}}$ (which is the dissimilarity between $\mathbf{C_a}$ and $\mathbf{C_2}$) is greater than the maximum dissimilarity of $\mathbf{C_b}$ to $\overline{\mathbf{S}}$ (which is the dissimilarity between $\mathbf{C_b}$ and $\mathbf{C_2}$). Therefore, based on the maximum dissimilarity, $\mathbf{C_a}$ will be selected. However, from human interpretation, $\mathbf{C_b}$ is more different from the clustering set $\overline{\mathbf{S}}$ than $\mathbf{C_a}$. Similarly, in Fig.3.1b, $\mathbf{C_a}$ has a higher average dissimilarity to the clustering set $\overline{\mathbf{S}}$ than $\mathbf{C_b}$ and $\mathbf{C_a}$ will be selected if the average dissimilarity is used. However, $\mathbf{C_b}$ is clearly more different from the clustering set $\overline{\mathbf{S}}$ than $\mathbf{C_a}$.

Definition 2 *(Overall Clustering Quality) The overall quality of a clustering* $\mathbf{C}$ *is characterized by the following bi-objective function* $\mathbb{F}(\mathbf{C}, \overline{\mathbf{S}})$*:*

$$\mathbb{F}(\mathbf{C}, \overline{\mathbf{S}}) = [\mathbb{Q}(\mathbf{C}), \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}})] \tag{3.2}$$

$$\textit{where} \begin{cases} \overline{\mathbf{S}} & \textit{is a given negative clustering set.} \\ \mathbb{Q}(\mathbf{C}) & \textit{is the quality of a clustering } \mathbf{C}. \\ \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) & \textit{is the dissimilarity between } \mathbf{C} \textit{ and } \overline{\mathbf{S}}. \end{cases}$$

Definition 3 *(Clustering Dominance) Given a set* $\overline{\mathbf{S}}$ *of negative clusterings, a clustering* $\mathbf{C}$ *dominates another clustering* $\mathbf{C}'$ *w.r.t* $\overline{\mathbf{S}}$*, written* $\mathbf{C} \succ_{\overline{\mathbf{S}}} \mathbf{C}'$ *iff one quality objective of* $\mathbf{C}$ *is better than or equal to that of* $\mathbf{C}'$ *and the other objective of* $\mathbf{C}$ *is strictly better than that of* $\mathbf{C}'$*. Formally,* $\mathbf{C} \succ_{\overline{\mathbf{S}}} \mathbf{C}'$ *if and only if the following conditions hold:*

$$\begin{aligned} (\mathbf{C} \neq \mathbf{C}') \wedge \big[\big(\mathbb{Q}(\mathbf{C}) > \mathbb{Q}(\mathbf{C}') \wedge \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) \geq \mathbb{D}(\mathbf{C}', \overline{\mathbf{S}})\big) \vee \\ \big(\mathbb{Q}(\mathbf{C}) \geq \mathbb{Q}(\mathbf{C}') \wedge \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) > \mathbb{D}(\mathbf{C}', \overline{\mathbf{S}})\big) \big] \end{aligned} \tag{3.3}$$

Finally, the alternative clustering problem is defined as follows:

Definition 4 *(Alternative Clustering) Given a set* $\overline{\mathbf{S}}$ *of negative clusterings, alternative clustering is the problem of finding a representative set of clusterings* $\mathbf{C}$ *along the Pareto front defined by the bi-objective function* $\mathbb{F}(\mathbf{C}, \overline{\mathbf{S}})$*.*

3.3.2 A Cluster-Oriented Algorithm for Alternative Clustering

In multi-objective optimization problems, usually there are efficient optimal solutions which cannot be compared. Therefore, one of the main goals of multi-objective optimization is to approximate the Pareto front. Evolutionary algorithms (EAs) possess several characteristics that are well suitable for multi-objective optimization. EAs approximate the Pareto

front in a single run by maintaining a solution set or population. In each iteration, this solution set **Q** is modified by two basic steps: selection and variation. The selection step chooses only well-adapted candidates from **Q** to form a set **P** of parent solutions. Then, in the variation step, the parent candidates in **P** are used to produce the next generation through recombination and mutation. The two steps are repeated until a number of generations is reached. We adapt one of the most popular EAs, **NSGAII** [29], for the alternative clustering problem defined in Section 3.3.1.

Applying EAs to the clustering problem is not straightforward because the traditional recombination and mutation operators of EAs are not very suitable for the clustering problem. The first reason is that they often cannot produce offspring solutions with good characteristics inherited from their parents [51]. Besides, in the case of fixed number of clusters, these operators can also produce invalid solutions. Therefore, when applying **NSGAII**, we replace these operators by two new operators called *Cluster-Oriented Recombination* and *Neighbor-Oriented Mutation* which can produce a valid offspring with good properties inherited from its parents. We defer the discussion of the deficiencies of the traditional genetic operators in Section 3.3.2 and 3.3.2, where we also describe our new genetic operators in detail. In the next sections, we summarize the **NSGAII** mechanism and present our modifications for the alternative clustering problem.

Fast NonDominated Sorting Algorithm (NSGAII)

The pseudo code of **NSGAII** is shown in Algorithm 7. Let P be the fixed size of populations generated in the algorithm. The algorithm first creates an initial parent population $\mathbf{P_0}$ and then produces an offspring population $\mathbf{Q_0}$ from $\mathbf{P_0}$ by the usual binary tournament selection, recombination, and mutation operators. The binary tournament selection procedure picks randomly two solutions and returns the nondominated one.

The set of non-dominated solutions $\mathbf{P_{best}}$ is initialized as the set of non-dominated solutions in $\mathbf{P_0} \cup \mathbf{Q_0}$. Then for each generation t, the procedure *FastNonDominatedSort*($\mathbf{R_t}$) classifies all solutions in the combined population $\mathbf{R_t} = \mathbf{P_t} \cup \mathbf{Q_t}$ into different dominance fronts (sorted in the ascending order of dominance depth where the first front is the set of non-dominated solutions). The pseudo code of the *FastNonDominatedSort* procedure is presented in Algorithm 8. We denote n_p the number of solutions that dominate a solution $\mathbf{p} \in \mathbf{P}$ and $\mathbf{S_p}$ the set of solutions that $\mathbf{p}$ dominates. The solutions $\mathbf{p}$ with $n_p = 0$ are placed in the first nondominated front. Then, for each solution $\mathbf{p}$ with $n_p = 0$, we visit each member $\mathbf{q}$ of its dominated set $\mathbf{S_p}$ and reduce its domination count by one. When doing so, if $n_q = 0$ then we put $\mathbf{q}$ in a separate list $\mathbf{Q}$. These members will belong the second nondominated front. The above process is repeated until all solutions are classified. The complexity of this procedure is $O(P^2\Lambda)$ where P is the population size, Λ is the number of objectives. In the case of alternative clustering with two objectives, the complexity of the *FastNonDominatedSort* procedure is $O(P^2)$. Because the combined population includes all solutions of the previous parent population $\mathbf{P_t}$ and the offspring population $\mathbf{Q_t}$, the non-dominated solutions found in previous generations are always kept in the next generations.

The algorithm sequentially adds the solutions of the first fronts $\mathbb{F}_\mathbf{i}$ to the next parent population $\mathbf{P_{t+1}}$ if after adding $\mathbb{F}_\mathbf{i}$, the size of $\mathbf{P_{t+1}}$ is still less than or equal to P. Otherwise, the remaining vacancies of $\mathbf{P_{t+1}}$ are filled by $P - |\mathbf{P_{t+1}}|$ solutions of $\mathbb{F}_\mathbf{i}$ with the largest crowding distances. The crowding distance of a solution in a population is an estimate of the solution density around that solution. The crowding distance of a solution $\mathbf{p}$ is measured as the sum of the normalized distances of two solutions on the left and right side of that solution along each objective. As illustrated in Fig.3.2, the crowding distance of the solution $\mathbf{p}$ is the sum of side lengths of the cuboid

(shown with a dashed box). The larger the crowding distance of a solution is, the less the solution density surrounding that solution is. Therefore, adding the largest crowding distance points encourages the diversity of the next parent population $\mathbf{P_{t+1}}$. The parent population $\mathbf{P_{t+1}}$ is now used to create a new offspring population $\mathbf{Q_{t+1}}$ by the regular evolutionary operators like binary tournament selection, recombination, mutation. In order to create a new solution $\mathbf{p}$, **NSGAII** selects two parents $\mathbf{p_1}$ and $\mathbf{p_2}$ by the binary tournament selection and then apply the recombination operator on $\mathbf{p_1}$ and $\mathbf{p_2}$ to produce $\mathbf{p}$. With probability of α, a mutation operator can be applied on $\mathbf{p}$ to increase the perturbation level. Then, the set $\mathbf{P_{best}}$ of non-dominated solutions obtained so far is updated by the non-dominated solutions in $\mathbf{Q_{t+1}}$. The whole process is repeated for the next generations. The complexity of generating a new offspring population $\mathbf{Q_{t+1}}$ from its parent population $\mathbf{P_{t+1}}$ is $O(P\Omega)$ where Ω is the complexity of computing Λ objectives. In alternative clustering problem, we have two objectives, therefore, the total complexity of **NSGAII** is $O(T(P^2 + P\Omega))$ where T is the number of generations.

In each generation, the number of nondominated solutions is bounded by the population size P. Therefore, when running **NSGAII** with T generations, the size of the result set $\mathbf{P_{best}}$ is bounded by PT. However, when the populations are moved towards the true Pareto front, the solutions at generation t mostly dominate the solutions of previous generation $t-1$. Therefore, in practice, the size of $\mathbf{P_{best}}$ is around P and much smaller than PT. Only when **NSGAII** has converged at generation $t < T$, but it is still running for other $T - t$ generations, then the size of $\mathbf{P_{best}}$ can grow up larger than P because of very similar nondominated solutions produced by **NSGAII** after converging.

The application of **NSGAII** to the alternative clustering problem requires the following choices:

Algorithm 7: NSGAII

Input : The number of generations T, the objective functions f_i
Output: The approximate Pareto front $\mathbf{P_{best}}$
begin
  Initialize the parent population $\mathbf{P_0}$.
  Create the population $\mathbf{Q_0}$ from $\mathbf{P_0}$.
  $\mathbf{P_{best}}$ = non-dominated solutions in $\mathbf{P_0} \cup \mathbf{Q_0}$.
  for $t = 1$ *to* T **do**
    // Selection phase
    $\mathbf{R_t} = \mathbf{P_t} \cup \mathbf{Q_t}$
    $\mathbb{F} = \mathit{FastNonDominatedSort}(\mathbf{R_t})$
    $\mathbf{P_{t+1}} = \emptyset$
    $i = 1$
    while $|\mathbf{P_{t+1}}| + |\mathbb{F}_\mathbf{i}| \leq P$ **do**
      $\mathbf{P_{t+1}} = \mathbf{P_{t+1}} \cup \mathbb{F}_\mathbf{i}$
      $i = i + 1$
    end
    if $|\mathbf{P_{t+1}}| < P$ **then**
      Sort $\mathbb{F}_\mathbf{i}$ in the descending order of crowding distances.
      $\mathbf{P_{t+1}} = \mathbf{P_{t+1}} \cup \mathbb{F}_\mathbf{i}[1 : (P - |\mathbf{P_{t+1}}|)]$
    end
    // Variation phase
    Create the population $\mathbf{Q_{t+1}}$ from $\mathbf{P_{t+1}}$.
    Update $\mathbf{P_{best}}$ with non-dominated solutions in $\mathbf{Q_{t+1}}$.
  end
  return $\mathbf{P_{best}}$
end

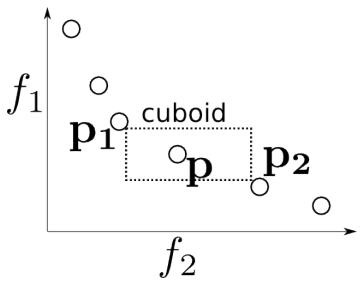


Figure 3.2: Crowding distance

Algorithm 8: FastNondominatedSort

```
Input  : A solution set P
Output: Classified fronts F_i of P
begin
    for p ∈ P do
        S_p = ∅
        n_p = 0
        for q ∈ P do
            if p ≻ q then
                S_p = S_p ∪ {q}
            else
                if q ≻ p then
                    n_p = n_p + 1
                end
            end
        end
        if n_p = 0 then
            p_rank = 1
            F_1 = F_1 ∪ {p}
        end
    end
    i = 1
    while F_i ≠ ∅ do
        Q = ∅
        for p ∈ F_i do
            for q ∈ S_p do
                n_q = n_q − 1
                if n_q = 0 then
                    q_rank = i + 1
                    Q = Q ∪ {q}
                end
            end
        end
        i = i + 1
        F_i = Q
    end
end
```

- Two objective functions.
- A genetic encoding of clusterings.
- Recombination and mutation operators to generate a new offspring population from a parent population.
- An effective initialization scheme.

In the next sections, we present the above components.

Objective Functions

We consider the Vector Quantization Error (VQE) – normally used in **K-Means** [62]– for measuring the clustering quality, because the base clustering algorithm used in **AFDT** and **AFDT2** is also **K-Means**. This objective has been shown to be very robust for noisy datasets. The VQE of a clustering solution $\mathbf{C}_t$ is the sum of the square distances from each data object $\mathbf{x}_i$ to the centroid of the cluster $\mathbf{C}_t^k$ where $\mathbf{x}_i$ belongs to. The VQE function is:

$$VQE(\mathbf{C}_t) = \sum_{\mathbf{C}_t^k \in \mathbf{C}_t} \sum_{\mathbf{x}_i \in \mathbf{C}_t^k} ||\mathbf{x}_i - \boldsymbol{\mu}_t^k||^2 \tag{3.4}$$

where $\mathbf{C}_t^k$ is a cluster in the clustering solution $\mathbf{C}_t$, $\boldsymbol{\mu}_t^k$ is the centroid of $\mathbf{C}_t^k$, and $||\mathbf{x}_i - \boldsymbol{\mu}_t^k||^2$ is the squared Euclidean distance between an item and its centroid. However, in the text datasets, the cosine distance is more suitable than the Euclidean distance. Therefore, when measuring the performance on the text datasets, we replace the Euclidean distance by the cosine distance. The Cosine VQE is:

$$CosineVQE(\mathbf{C}_t) = \sum_{\mathbf{C}_t^k \in \mathbf{C}_t} \sum_{\mathbf{x}_i \in \mathbf{C}_t^k} (1 - cosine(\mathbf{x}_i, \boldsymbol{\mu}_t^k)) \tag{3.5}$$

where $cosine(\mathbf{x}_i, \boldsymbol{\mu}_t^k)$ is the cosine similarity between $\mathbf{x}_i$ and $\boldsymbol{\mu}_t^k$. The smaller the VQE is, the better the quality of a clustering is. The cost

of computing VQE for a clustering $\mathbf{C}_t$ is $O(ND)$ where N is the dataset size and D is the number of dimensions of data objects.

To measure the similarity between two clusterings, we use the popular Adjusted Rand Index (ARI) [53]. ARI is a normalized version of the Rand Index (RI) proposed by Rand et al. [75]. The Rand Index $RI(\mathbf{C}_1, \mathbf{C}_2)$ between two clusterings $\mathbf{C}_1$ and $\mathbf{C}_2$ is simply defined as $\frac{n_{11}+n_{00}}{n_{11}+n_{10}+n_{01}+n_{00}}$ where n_{11} is the number of object pairs that are in the same cluster in both two clusterings; n_{00} is the number of pairs that are in different clusters in both clusterings; n_{10} is the number of pairs that are assigned in the same cluster by the clustering $\mathbf{C}_1$ and in different clusters by the clustering $\mathbf{C}_2$; n_{01} is the number of pairs that are assigned in different clusters by the clustering $\mathbf{C}_1$ and in the same cluster by the clustering $\mathbf{C}_2$. A problem with RI is that the expected value for two random clusterings is not constant. Hubert et al. [53] fix this issue by introducing a normalized version of RI, called ARI. The ARI between two solutions $\mathbf{C}_1$ and $\mathbf{C}_2$ is defined as follows:

$$ARI(\mathbf{C}_1, \mathbf{C}_2) = \frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex}$$

$$ARI(\mathbf{C}_1, \mathbf{C}_2) = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{n_{i.}}{2} \sum_j \binom{n_{.j}}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}\right] - \left[\sum_i \binom{n_{i.}}{2} \sum_j \binom{n_{.j}}{2}\right] / \binom{n}{2}} \quad (3.6)$$

where n_{ij} is the number of common data objects of two clusters $\mathbf{X}_i$ and $\mathbf{X}_j$ produced by the clustering solutions $\mathbf{C}_1$ and $\mathbf{C}_2$, and $n_{i.} = \sum_j n_{ij}$, $n_{.j} = \sum_i n_{ij}$. The maximum value of ARI is 1 when two clusterings are identical. The value of ARI is around zero, or even a negative value, when two clusterings are very different. As we prefer different alternative clusterings, the smaller the ARI is, the better the dissimilarity between two clusterings is. In other words, we minimize the maximum similarity between the alternative clustering and the negative clustering set. Therefore, we define the similarity between an alternative clustering and a negative clustering set

(similarly to the dissimilarity definition in Equation 3.1) as the maximum similarity between that alternative clustering and the clusterings in the negative clustering set. The complexity of computing ARI between two clusterings is $O(N)$ where N is the dataset size. Therefore, the total complexity of **COGNAC** when optimizing VQE and ARI is $O(T(P^2+PND))$ where T is the number of generations and P is the population size. In other words, fixing the number of generations and the population size, the complexity of **COGNAC** increases *linearly* with the dataset size N and the number of dimensions D of data objects.

Genetic Encoding of Clusterings

We use the cluster-index based representation to encode clustering solutions. In detail, a clustering solution $\mathbf{C}_t$ of N data objects $\{\mathbf{x}_i\}_{i=1}^N$ is a N-dimensional vector where $\mathbf{C}_t(i)$ is the index of the cluster where the data object $\mathbf{x}_i$ belongs to. The index of each cluster is in the range of 1 to K with K is the fixed number of clusters. For example, with a dataset of 10 objects, and the number of clusters is 3, the clustering solution $\mathbf{C}_t = [1113331122]$ represents there clusters: $\mathbf{X_1} = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_7}, \mathbf{x_8}\}$, $\mathbf{X}_2 = \{\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$, $\mathbf{X}_3 = \{\mathbf{x}_9, \mathbf{x}_{10}\}$.

Cluster-Oriented Recombination Operator

Although the cluster-index encoding is popular in the literature, its main disadvantage is that the traditional recombination operators often do not produce offspring solutions which inherit good properties from their parents. The first problem is that the traditional recombination operators are performed on the *object level* whereas the clustering meaning is considered on the *cluster level*. In other words, the clusters are the smallest units containing information about the quality of the clustering solution to which they belong [36]. Another drawback is that one clustering can

be represented by many chromosomes, e.g. two chromosomes [123144] and [314322] represents the same solution of four clusters $\mathbf{C}^1 = \{\mathbf{x}_1, \mathbf{x}_4\}, \mathbf{C}^2 = \{\mathbf{x}_2\}, \mathbf{C}^3 = \{\mathbf{x}_3\}, \mathbf{C}^4 = \{\mathbf{x}_5, \mathbf{x}_6\}$. Therefore, performing recombination operators without a correct matching of clusters can return invalid solutions as in the following example:

$$[1|\overline{23}|144]$$
$$[\overline{3}|14|\overline{322}]$$
$$\overline{[3|23|322]}$$

The offspring [3|23|322] not only has an invalid number of clusters but also is very different from its parents. In this case, the offspring should be identical to their parents because they represent the same clustering solution.

To solve the above problems, we propose a *cluster-oriented* recombination operator where recombination is performed on clusters rather than on separate objects. The idea of performing recombination on clusters was first proposed by Falkenauer et al. [36] for the bin packing problem. However, their recombination operator cannot be applied to the clustering problem as it assumes special characteristics of the bin packing problem. In addition, their recombination does not perform a matching before merging clusters of two parents, therefore invalid solutions can still be returned.

The pseudo-code of our cluster-oriented recombination operator is presented in Algorithm 9. We first find a perfect matching $\mathbf{M}$ between clusters of two parents such that the number of common objects between them is largest. In this chapter, we use the perfect matching algorithm proposed by Munkres et al.[69]. The complexity of the matching algorithm is $O(K^4)$ (or $O(K^3)$ if optimized) where K is the number of clusters. Often, K is very small compared to the dataset size N, therefore the overhead of computing a perfect matching is relatively small.

Then, we perform uniform crossover on clusters as follows. First, we select a set $\mathbf{I}$ of $K/2$ random positions in $\{1, .., K\}$ to copy clusters $\mathbf{C}^i_{p_1}$ (where $i \in \mathbf{I}$) of the first parent $\mathbf{C}_{p_1}$ to the offspring $\mathbf{C}_o$. Let $\mathbf{U}$ be the set of all unassigned objects. Then, for each remaining position $i \in \{1, .., K\} \setminus \mathbf{I}$, we compute the set $\mathbf{C}^{\mathbf{M}(i)}_{p_2} \cap \mathbf{U}$ of unassigned objects in the cluster $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$ of the second parent $\mathbf{C}_{p_2}$. If all objects in $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$ are assigned, it means that $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$ is strictly included in some cluster of the first parent. Therefore, we move all objects in $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$ to cluster i to avoid empty clusters. Otherwise, we simply assign the unassigned objects in $\mathbf{C}^{\mathbf{M}(i)}_{p_2} \cap \mathbf{U}$ to cluster i. After merging clusters from two parents, there are still unassigned (or orphan) objects. These orphan objects will be assigned to the clusters of one randomly chosen parent. We assign the orphan objects to the clusters of only one parent to preserve good characteristics from that parent.

An example of a dataset with 12 objects is in Fig.3.3a. The number of clusters is 3. The clusters of two parents are as in Fig.3.3b, 3.3c. The perfect matching $\mathbf{M}$ will match: $\mathbf{C}^1_{p_1} \rightarrow \mathbf{C}^{\mathbf{M}(1)=3}_{p_2}$, $\mathbf{C}^2_{p_1} \rightarrow \mathbf{C}^{\mathbf{M}(2)=1}_{p_2}$, $\mathbf{C}^3_{p_1} \rightarrow \mathbf{C}^{\mathbf{M}(3)=2}_{p_2}$ as in Fig.3.3d. Assume that $\mathbf{I} = \{1\}$. We copy cluster $\mathbf{C}^1_{p_1}$ from $\mathbf{C}_{p_1}$, and move the unassigned objects in two clusters $\mathbf{C}^{\mathbf{M}(2)=1}_{p_2}$, $\mathbf{C}^{\mathbf{M}(3)=2}_{p_2}$ from $\mathbf{C}_{p_2}$ to the offspring, as in Fig.3.3e. Then, we assign the orphan object 5 to the cluster $\mathbf{C}^2_o$ as in the first parent to obtain the offspring as in Fig.3.3f. As it can be seen, the offspring inherits all good properties from its parents and converges to a correct clustering solution.

Neighbor-Oriented Mutation Operator

In the traditional mutation operators, usually some objects are selected and moved randomly to different clusters. However, moving an object $\mathbf{x}_i$ to a *random* cluster $\mathbf{C}^k$ can radically decrease the clustering quality when $\mathbf{x}_i$ is too far from $\mathbf{C}^k$. Also, if only few objects are moved to new clusters, the resulting perturbation can be too small for escaping local minima. But

Algorithm 9: Cluster-Oriented Recombination Operator

Input : Two parent solutions $\mathbf{C}_{p_1}, \mathbf{C}_{p_2}$, the number of clusters K, the dataset $\mathbf{X}$.
Output: An offspring solution $\mathbf{C}_o$.
begin
  Initialize $\mathbf{C}_o$: $\forall i \in \{1, .., N\} : \mathbf{C}_o(i) = -1$.
  Let $\mathbf{C}^i_{p_j}$ be the i-th cluster of parent $\mathbf{C}_{p_j}$.
  Find a maximum perfect matching $\mathbf{M}$ where $\mathbf{C}^i_{p_1}$ is matched to $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$.
  Let $\mathbf{I}$ be the set of $K/2$ indices selected randomly from $\{1, .., K\}$.
  // Perform uniform crossover on clusters.
  Copy clusters $\mathbf{C}^i_{p_1}$ where $i \in \mathbf{I}$ to the offspring: $\forall \mathbf{x}_t \in \mathbf{C}^i_{p_1} : \mathbf{C}_o(t) = i$.
  for $i \in \{1, .., K\} \setminus \mathbf{I}$ **do**
    $\mathbf{U} = \{\mathbf{x}_j : \mathbf{x}_j \in \mathbf{X} \wedge \mathbf{C}_o(j) = -1\}$.
    if $\mathbf{C}^{\mathbf{M}(i)}_{p_2} \cap \mathbf{U} = \emptyset$ **then**
      $\forall \mathbf{x}_t \in \mathbf{C}^{\mathbf{M}(i)}_{p_2} : \ \mathbf{C}_o(t) = i$.
    else
      $\forall \mathbf{x}_t \in \mathbf{C}^{\mathbf{M}(i)}_{p_2} \cap \mathbf{U} : \ \mathbf{C}_o(t) = i$.
    end
  end
  // Assign orphan objects.
  if *rand() % 2 = 0* **then**
    for $i \in \{1, .., K\} \setminus \mathbf{I}$ **do**
      $\forall \mathbf{x}_j. \ \mathbf{x}_j \in \mathbf{C}^i_{p_1} \wedge \mathbf{C}_o(j) = -1 : \mathbf{C}_o(j) = i$.
    end
  else
    for $i \in \mathbf{I}$ **do**
      $\forall \mathbf{x}_j. \ \mathbf{x}_j \in \mathbf{C}^{\mathbf{M}(i)}_{p_2} \wedge \mathbf{C}_o(j) = -1 : \mathbf{C}_o(j) = i$.
    end
  end
  return $\mathbf{C}_o$
end

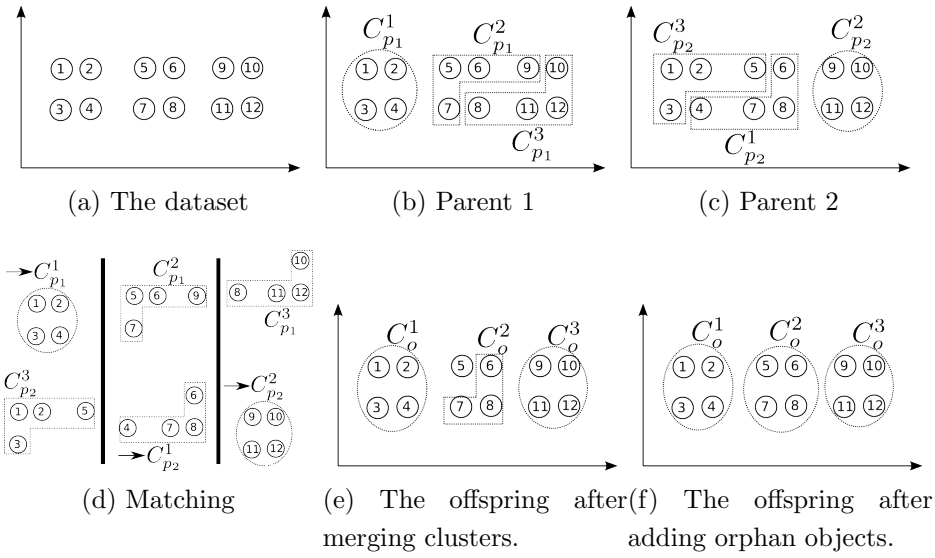


Figure 3.3: Cluster-Oriented Recombination Operator Example.

if many objects are moved to random clusters, the quality of the offspring can be strongly degraded. Therefore, determining the set of items to move is a difficult task. As it was the case for recombination, the traditional mutation operators can also produce invalid solutions when moving an object in a singleton cluster to a new cluster without checking the validity. To solve these problems, we replace the traditional mutation operators with a new operator called the *Neighbor-Oriented Mutation* operator.

The pseudo-code of the new mutation operator is presented in Algorithm 10. In detail, with the probability of ρ, each data object $\mathbf{x}_i$ in a cluster with size greater than 1 is moved to the cluster of one of its γ nearest neighbors $\mathbf{x}_j$. In other words, a proportion ρ of the data objects will be selected randomly and moved to the clusters of one of their nearest neighbors. Besides, we do not move objects of singleton clusters, therefore the validity of the offspring solution is guaranteed. Moving an item in this manner avoids assigning it to a very far cluster, therefore the search space is reduced significantly, but the clustering solution is still of good quality. Besides, this operator can also produce arbitrarily-shaped clusters by linking near

objects, e.g., a long cluster can be formed as a chain of near objects. Note that moving an object to one of the clusters of its nearest neighbors is different from moving that object to the nearest cluster: the first strategy allows arbitrary-shape clusters whereas the second one favors spherical clusters. To reduce the computational cost, the nearest neighbors of all objects will be computed only once before calling the neighbor-oriented mutation operator. For high-dimensional datasets, the distance between the objects can be computed by a suitable kernel. In this chapter, we simply use the Euclidean distance.

The larger the value of ρ is, the more perturbed the offspring is. On the contrary, with small values of ρ, the search space is restricted, and as a result, the probability of remaining stuck in local minima increases. A similar issue also regards the number of nearest neighbors (parameter γ). A large value of γ allows moving an object $\mathbf{x}_i$ to far clusters and a small value of γ permits moving $\mathbf{x}_i$ only to near clusters. Setting γ too large results in random moves of data objects and thus wasting a lot of computing resources because very far objects can be assigned to the same cluster. But setting γ too small can limit the search space too much and keep the algorithm confined close to local minima.

To solve the above problems, we propose a method inspired by Simulated Annealing [57]. At the beginning, both parameters are assigned large values to allow the algorithm to explore potential solution regions. Then, these parameters are gradually decreased to help the algorithm exploit the most promising regions. This scheme automatically shifts in a gradual manner from diversification to intensification. In detail, in the first generation, ρ is assigned to ρ_{max} and then in the next generations, ρ is decreased sequentially by multiplying by a decay factor ρ_{dec} such that the value of ρ in the last generation is ρ_{min}. Mathematically, the probability ρ_t of moving an object at the t-th generation is computed as $\rho_t = \rho_{max}\rho_{dec}^t$ and

Algorithm 10: Neighbor-Oriented Mutation Operator

Input : A solution $\mathbf{C}_o$, the perturbation probability ρ, the number of nearest neighbors γ.

Output: The perturbed solution $\mathbf{C}_o$.

begin

  for *each object* $\mathbf{x}_i \in \mathbf{X}$ **do**

    Let $\mathbf{X}_{nn}$ be the set of the first γ nearest neighbors of the object $\mathbf{x}_i$.

    Pick randomly a nearest neighbor $\mathbf{x}_j \in \mathbf{X}_{nn}$.

    if *the number of objects in the cluster of* $\mathbf{x}_i$ *is greater than 1* **then**

      With the probability of ρ, moving $\mathbf{x}_i$ to the cluster of $\mathbf{x}_j$ by assigning: $\mathbf{C}_o(i) = \mathbf{C}_o(j)$.

    end

  end

  return $\mathbf{C}_o$

end

$\rho_{max}\rho_{dec}^{T} = \rho_{min}$ where T is the number of generations. Formally, ρ_{dec} is computed based on ρ_{min} and ρ_{max} as follows:

$$\rho_{dec} = \sqrt[T]{\frac{\rho_{min}}{\rho_{max}}} \tag{3.7}$$

Similarly, the number of nearest neighbours γ is first set to γ_{max} and then sequentially decreased by multiplying by a decay factor γ_{dec}. However, we only decrease γ in the first $T/2$ generations and keep γ as γ_{min} in the remaining generations to guarantee a large enough perturbation for the algorithm to escape from local minima. In detail, for the t-th generation where $t < T/2$, $\gamma_t = \gamma_{max}\gamma_{dec}^{t}$ where γ_{dec} is computed such that $\gamma_{max}\gamma_{dec}^{T/2} = \gamma_{min}$. In other words, γ_{dec} is computed as:

$$\gamma_{dec} = \sqrt[T/2]{\frac{\gamma_{min}}{\gamma_{max}}} \tag{3.8}$$

For the t-th generation where $t \geq T/2$, γ_t is set to γ_{min}. In the implementation, γ_t is a double variable and rounded to an integer by a built-in ceiling function when calling the *Neighbor-Oriented Mutation* operator.

Initialization

The initialization phase plays an important role in guiding the algorithm towards the true Pareto front. If the initial population contains only random solutions which are very different from the negative clusterings, then the algorithm explores well the region where the dissimilarity of the alternative clusterings and the negative ones is high. Analogously, if solutions similar to the negative clusterings are included into the initial set, then the algorithm often produces high-quality clusterings but similar to the negative ones. Here we assume that the negative clusterings are of high quality because they are usually obtained from single objective clustering algorithms. From this observation, we generate the initial population such that half of them are dissimilar clusterings and the rest are high-quality clusterings as follows.

Generating dissimilar clusterings: Let P be the initial population size and K_{neg} and K_{alter} be the number of clusters in negative clusterings and alternative clusterings, respectively. We generate $P/2$ dissimilar solutions from pairs of negative clusterings and individual negative clusterings as follows.

For each pair of two negative clustering solutions $\mathbf{C}_1$ and $\mathbf{C}_2$, we first find a perfect matching $\mathbf{M}$ between their clusters. Then, for each pair of matched clusters $\mathbf{C}_1^i \rightarrow \mathbf{C}_2^{\mathbf{M}(i)}$, we compute a common cluster $\mathbf{C}_{com}^i = \mathbf{C}_1^i \cap \mathbf{C}_2^{\mathbf{M}(i)}$, and a xor cluster $\mathbf{C}_{xor}^i = (\mathbf{C}_1^i \cup \mathbf{C}_2^{\mathbf{M}(i)}) \setminus \mathbf{C}_{com}^i$. Then we randomly merge two nearest common clusters or xor clusters until their total number equals K_{alter} to generate a dissimilar offspring $\mathbf{C}_o$. The distance between two common clusters or two xor clusters is computed as the Euclidean distance between their centroids. The offspring is very dissimilar from its parents because in their parents, the objects in two common or two xor clusters are in different clusters, but in the offspring they are placed into

the same cluster. Note that we do not merge a common cluster and a xor cluster because they can reproduce a cluster which equals one of parents' clusters. If the number of clusters in two negative solutions are different, before matching, we sequentially split the largest cluster of the solution with the smaller number of clusters into two sub-clusters by **K-Means** until the number of clusters in two solutions are the same.

For individual negative clustering $\mathbf{C}_1$, we first extract its K_{neg} clusters $\{\mathbf{C}_1^i\}_{i=1}^{K_{neg}}$. Next, for each cluster $\mathbf{C}_1^i$, we use **K-Means** [62] to partition this cluster into K_{alter} sub-clusters $\{\mathbf{C}_1^{i_j}\}_{j=1}^{K_{alter}}$. The remaining objects in $\mathbf{X} \setminus \mathbf{C}_1^i$ are assigned to the j-th nearest sub-cluster $\mathbf{C}_1^{i_j}$, with probability $\alpha^{-j} / \sum_{t=1}^{K} \alpha^{-t}$ to form a dissimilar offspring $\mathbf{C}_o$. The parameter α is a factor determining the perturbation level of the offspring solution. In other words, the probability of assigning an unassigned object to its (j+1)-th nearest sub-cluster is α times smaller than the probability of assigning that object to the j-th nearest sub-cluster. The smaller α is, the more perturbed the offspring is, therefore we vary α from $\alpha_{min} = 2$ to $\alpha_{max} = 10$ to generate a diverse set of dissimilar solutions. In detail, from each cluster $\mathbf{C}_1^i$ and a value $\alpha \in \{\alpha_{min}, .., \alpha_{max}\}$, we generate $\frac{P/2 - N_{\overline{C}}(N_{\overline{C}}-1)/2}{N_{\overline{C}} K_{neg}(\alpha_{max}-\alpha_{min}+1)}$ dissimilar offspring where $N_{\overline{C}}$ is the number of negative clusterings. The distance between an object and a sub-cluster is computed as the Euclidean distance between that object and the sub-cluster centroid. The offspring generated by the above strategy is very dissimilar to their parents because the objects in each cluster $\mathbf{C}_1^i$ of their parents $\mathbf{C}_1$ are now assigned to different clusters. This strategy is similar to the ensemble clustering algorithm proposed by Gondek et al.[42], but different because of the perturbation parameter α to diversify the offspring set.

Generating high-quality clustering: We generate $\frac{P/2}{N_{\overline{C}}}$ high quality offspring from each negative clustering $\mathbf{C}_1$ as follows. First, we extract its K_{neg} clusters $\{\mathbf{C}_1^i\}_{i=1}^{K_{neg}}$. If $K_{neg} > K_{alter}$, we merge sequentially two

nearest clusters until the number of clusters is K_{alter}. In the case where $K_{neg} < K_{alter}$, we split iteratively the largest cluster into two sub-clusters by **K-Means** until the number of clusters equals K_{alter}. Then, we compute the cluster centroids $\{\boldsymbol{\mu}_1^i\}_{i=1}^{K_{alter}}$ and assign each object to its i-th nearest centroid with the probability $\alpha^{-i} / \sum_{t=1}^{K_{alter}} \alpha^{-t}$ to obtain a new offspring. Similar to the procedure of generating dissimilar offspring, we also vary α from $\alpha_{min} = 2$ to $\alpha_{max} = 10$ for diversifying the high-quality offspring set.

3.3.3 Sequential Generation of Alternative Clusterings

Based on the **COGNAC** algorithm (presented in Section 3.3.2), we propose the algorithm **SGAC** (the abbreviation of **S**equential **G**eneration of **A**lternative **C**lusterings) to generate a sequence of alternative clusterings as in Algorithm 11. First, the negative clustering set contains only the initial negative clustering and the alternative clustering set is empty. This initial negative clustering is often obtained from popular single objective clustering algorithms like **K-Means** [62], **Hierarchical Clustering** [60]. Then in each iteration, the user will select one of the alternative clusterings returned by **COGNAC**. We defer the detailed discussion of the selection technique in Section 3.3.4. This alternative clustering is added to both sets of negative and alternative clusterings. Therefore, the alternative clustering generated in each next iteration is guaranteed to be different from the previous alternative clusterings. Finally, the set of all different alternative clusterings is returned to the user.

3.3.4 Analyzing the Pareto front

In order to reduce the number of solutions presented to users, we apply a traditional clustering algorithm like **K-Means** to partition the solution set into K clusters. Because the range of dissimilarity and quality are

Algorithm 11: SGAC

Input : The initial negative clustering solution $\overline{\mathbf{C}}$, the number of alternative clusterings M

Output: The set of alternative clusterings

begin

  $\overline{\mathbf{S}} = \{\overline{\mathbf{C}}\}$

  $\mathbf{S}' = \emptyset$

  for $m = 1$ *to* M **do**

    $\mathbf{S}^* = \mathbf{COGNAC}(\overline{\mathbf{S}})$

    The user selects one alternative clustering $\mathbf{C}'$ from $\mathbf{S}^*$.

    $\overline{\mathbf{S}} = \overline{\mathbf{S}} \cup \{\mathbf{C}'\}$

    $\mathbf{S}' = \mathbf{S}' \cup \{\mathbf{C}'\}$

  end

  return $\mathbf{S}'$

end

different, when clustering the solutions, we normalize their dissimilarity and quality as follows:

$$\mathbb{D}'(\mathbf{C}, \overline{\mathbf{S}}) = \frac{\mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) - \mu_{\mathbb{D}}}{\sigma_{\mathbb{D}}} \tag{3.9}$$

$$\mathbb{Q}'(\mathbf{C}) = \frac{\mathbb{Q}(\mathbf{C}) - \mu_{\mathbb{Q}}}{\sigma_{\mathbb{Q}}} \tag{3.10}$$

where $\mu_{\mathbb{D}}, \sigma_{\mathbb{D}}, \mu_{\mathbb{Q}}, \sigma_{\mathbb{Q}}$ are the mean and standard deviation of dissimilarity and quality of the solution set, respectively. For each cluster of solutions $\mathbf{S_i}$, the ranges of its dissimilarity and quality are represented in two border solutions: the one with the highest dissimilarity and lowest quality, and the other one with the highest quality and lowest dissimilarity. Therefore, users only need to consider the two border solutions of each cluster and quickly discard unsuitable clusters. If the user is satisfied with one of the border solutions, the algorithm can stop. Otherwise, the user selects a cluster of solutions with a reasonable range of dissimilarity and quality. Then, he can analyze that cluster more deeply by partitioning it again into

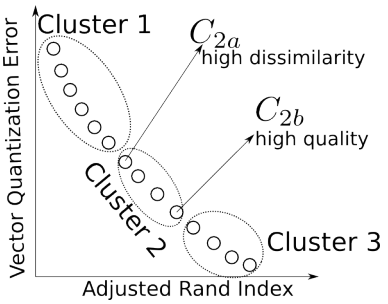


Figure 3.4: Analyzing the Pareto front with clustering.

sub-clusters and repeating the above process until a satisfactory solution is met.

Fig.3.4 shows an example of partitioning the Pareto solutions into 3 clusters. In cluster 2, the solution $\mathbf{C_{2a}}$ is the solution with the highest dissimilarity and lowest quality, and the solution $\mathbf{C_{2b}}$ is the solution with the highest quality and lowest dissimilarity. Assume that the range of dissimilarity and quality of two solutions $\mathbf{C_{2a}}$ and $\mathbf{C_{2b}}$ satisfies the users' requirement. If they satisfy with one of the two border solutions, they can stop the algorithm. Otherwise, if users want to have finer solutions, they can run a traditional clustering algorithm like **K-Means** [62] to partition cluster 2 into three other sub-clusters and repeat the whole process.

Besides, when plotting all solutions, the figures are very difficult to read, therefore, we filter the similar solutions on the Pareto front as follows. First, we sort all solutions in the descending order of the dissimilarity objective. Then, we add the first solution with the highest dissimilarity to the filtered Pareto front $\mathbf{S_{filtered}}$. For each next solution $\mathbf{C}$, we compute the normalized difference on each objective between $\mathbf{C}$ and the previously added solution $\mathbf{C}'$. Denote $\mathbf{S}^*$ the Pareto solution set returned by **COGNAC**. The normalized difference (w.r.t a negative clustering set $\overline{\mathbf{S}}$) in dissimilarity Δ_D and in quality Δ_Q of two solutions $\mathbf{C}$ and $\mathbf{C}'$ are

computed as:

$$\Delta_{\mathbb{D}}(\mathbf{C}, \mathbf{C}') = \frac{|\mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) - \mathbb{D}(\mathbf{C}', \overline{\mathbf{S}})|}{\mathbb{D}_{max} - \mathbb{D}_{min}} \tag{3.11}$$

$$\mathbb{D}_{max} = \max_{\mathbf{C} \in \mathbf{S}^*} \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) \tag{3.12}$$

$$\mathbb{D}_{min} = \min_{\mathbf{C} \in \mathbf{S}^*} \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) \tag{3.13}$$

$$\Delta_{\mathbb{Q}}(\mathbf{C}, \mathbf{C}') = \frac{|\mathbb{Q}(\mathbf{C}) - \mathbb{Q}(\mathbf{C}')|}{\mathbb{Q}_{max} - \mathbb{Q}_{min}} \tag{3.14}$$

$$\mathbb{Q}_{max} = \max_{\mathbf{C} \in \mathbf{S}^*} \mathbb{Q}(\mathbf{C}) \tag{3.15}$$

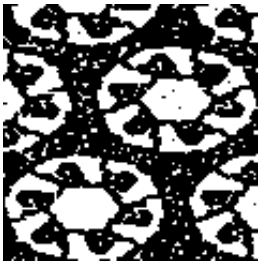
$$\mathbb{Q}_{min} = \min_{\mathbf{C} \in \mathbf{S}^*} \mathbb{Q}(\mathbf{C}) \tag{3.16}$$

If the normalized difference on two objectives $\Delta_{\mathbb{D}}(\mathbf{C}, \mathbf{C}')$ and $\Delta_{\mathbb{Q}}(\mathbf{C}, \mathbf{C}')$ between two solutions $\mathbf{C}$ and $\mathbf{C}'$ are equal to or greater than a threshold δ, then $\mathbf{C}$ is added to the filtered Pareto front $\mathbf{S}_{\mathbf{filtered}}$. The above process is repeated until all solutions are considered. This technique can also be applied before partitioning the approximated Pareto front into K clusters to remove similar solutions.

3.4 Experiments

In this section, we describe the test datasets in Section 3.4.1 and the experiments to compare the performance of our algorithm with that of other state-of-the-art algorithms on the first alternative clustering in Section 3.4.2. Then, we illustrate the ability of our algorithm for generating a sequence of different alternative clusterings and compare the result with that of other two algorithms in Section 3.4.3. Finally, we analyze the parameter sensitivity of **COGNAC** in Section 3.4.4.

(a) The *Flowers* image.



(b) The *Flowers* image obtained by **K-Means**.

(c) The *Birds* image.

(d) The *Birds* image obtained by **K-Means**.

Figure 3.5: Escher images.

3.4.1 Datasets

In order to measure the performance of alternative clustering algorithms, we use four datasets with "ground truth" alternative clusterings. The first and second dataset are two Escher images with multiple interpretations to the human eye as in Fig.3.5a, 3.5c. The *Flowers* and *Birds* images' size are 120×119 and 106×111, respectively. The RGB color space of each image is then converted in the L*a*b* color space (an important attribute of the L*a*b*-model is the device independence: the colors are defined independently of the device that they are displayed on). The difference in the color of two pixels can be computed as the Euclidean distance between their a* and b* values. The negative clustering of each image in Fig.3.5b, 3.5d is obtained by running **K-Means** to partition each image into two clusters.

Dataset	Cardinality	Number of dimensions	Number of clusters
CMUFaces	624	39	20
WebKB	1041	500	4
Birds	11766	2	2
Flowers	14280	2	2

Table 3.1: Dataset characteristics.

The third dataset is the *CMU Face* dataset on the UCI repository [38]. We use all 624 face images of 20 people taken with varying pose (straight, left, right, up), expression (neutral, happy, sad, angry), eyes (wearing sunglasses or not). The size of each image is $960(32 \times 30)$. We then apply PCA as in [92] to reduce the number of dimensions to 39. The labelling of each image by the name of the person in that image is used as the negative clustering of this dataset.

The fourth dataset is the *WebKB*[1] dataset. It contains HTML documents collected mainly from four universities: Cornell, Texas, Washington, Wisconsin, and classified under four groups: course, project, faculty, students. We select 500 features with highest information gain (conditioned on group names) to reduce the number of dimensions. Then, we remove stop words, stemming, and use TF-IDF weighting to construct the feature vectors. The resulting dataset consists of 1041 documents. The labelling on four groups is used as the negative clustering.

Table 3.1 summarizes the datasets.

3.4.2 Comparisons on The First Alternative Clustering

We compare the performance of our algorithm **COGNAC** on the first alternative clustering with that of four other state-of-the-art algorithms, namely **COALA**[3], **AFDT**[25], **AFDT2** [74], **MinCEntropy**$^{++}$[92] on four datasets in Section 3.4.1.

[1] `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/`

Number of generations	200
Population size	200
Mutation rate	0.2
ρ_{max}	0.3
ρ_{min}	0.1
γ_{max}	40
γ_{min}	10

Table 3.2: Parameter setting for **COGNAC**.

Experimental Setup

The parameters of **COGNAC** are set as in Table 3.2. For a specific configuration, **COALA**, and **AFDT2** can only produce one solution. In order to generate a set of different solutions from **COALA**, the parameter declaring user reference on clustering quality and dissimilarity w is changed from 0.1 to 1.0 with step of 0.1. Large w values imply better clustering quality and smaller clustering dissimilarity. Similarly, the trade-off parameter a of **AFDT2** is changed from 1.0 to 2.8 with step of 0.2. Increasing a improves the clustering dissimilarity and decreases the clustering quality. The default values for w and a are set to 0.6 [3] and 2.0 [74], respectively. **AFDT** has no parameters and can only produce one solution. For **MinCEntropy**$^{++}$, we also vary its trade-off parameter m from 1 to 10 for generating a diverse set of solutions.

Except for the *CMUFaces* dataset where the number of clusters in alternative clusterings K_{alter} is set to 4, on other datasets, K_{alter} is set as the number of clusters in negative clusterings. Besides, on each dataset, **COGNAC**, **MinCEntropy**$^{++}$ and the base algorithm **K-Means** of **AFDT** and **AFDT2** are run 10 times to reduce the randomness effect. **COGNAC**[2] and **COALA**[3] are implemented in C++ and Java, respectively. The other

[2]http://lion.disi.unitn.it/intelligent-optimization/cognac/
[3]http://ericbae.com/2011/05/11/clown-clustering-package/

three algorithms **AFDT**, **AFDT2**[4] and **MinCEntropy++**[5] are implemented in Matlab. All algorithms are run on a machine with Intel(R) Xeon(R) CPU E5440 @ 2.83GHz, and the Ubuntu 9.10 Operating System.

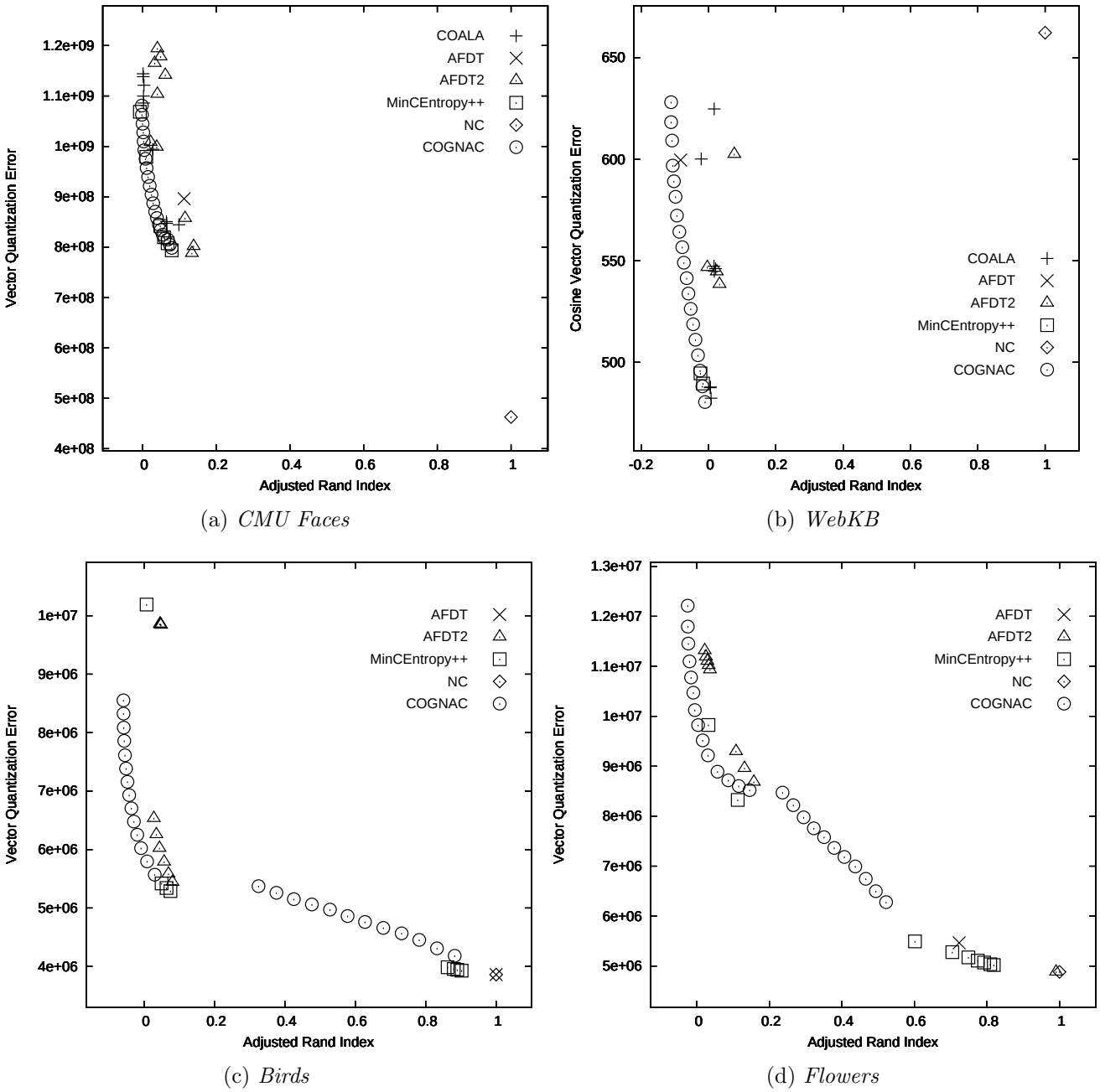
Experimental Results

Fig.3.6 shows the performance of five algorithms on four datasets. In the figures, we also plot the negative clusterings to present the trade-off between clustering quality and dissimilarity. We denote the negative clusterings as **NC** in the plots. Besides, in order to keep the figures readable, we only plot some representative solutions on the Pareto front produced by **COGNAC** (by applying the filter procedure as in Section 3.3.4). On two large datasets *Birds* and *Flowers*, **COALA** cannot finish after 24 hours of CPU time.

As it can be observed, on most datasets our **COGNAC** provides diverse sets of high quality (in both clustering quality and dissimilarity) solutions. All solutions of **COALA**, **AFDT** and **AFDT2** are above the Pareto front of **COGNAC**. Thus, for each clustering solution produced by these three algorithms, there is always a solution produced by **COGNAC** of better quality in both objectives. Especially, on the *WebKB* and *Birds* datasets, our algorithm produces solutions of much higher quality. When comparing **COGNAC** and **MinCEntropy**$^{++}$, on the *Birds* dataset, the solution of **MinCEntropy**$^{++}$ on the left-up corner is outperformed significantly by other solutions of **COGNAC**. On the other datasets, the solutions of two algorithms slightly dominate each other. However, **COGNAC** provides a much more diverse set of solutions compared to that of **MinCEntropy**$^{++}$. In other words, increasing the trade-off parameter of **MinCEntropy**$^{++}$ with equal intervals does not guarantee to obtain a diverse set of solutions.

[4] `http://wwwcsif.cs.ucdavis.edu/~qiz/code/code.html`

[5] `https://sites.google.com/site/vinhnguyenx/publications`



(a) *CMU Faces* (b) *WebKB*

(c) *Birds* (d) *Flowers*

Figure 3.6: Performance comparison of five algorithms on four datasets.

Dataset	**COALA**	**AFDT**	**AFDT2**	**MinCEntropy^{++}**	**COGNAC**
CMU Faces	0.16	0.60	0.01	0.01	2.49
WebKB	60.17	62.24	0.32	0.53	19.58
Birds	NT	3050.70	8.45	13.25	63.43
Flowers	NT	5418.13	12.54	20.50	132.87

Table 3.3: Run-time (in seconds) of five algorithms. NT means "Not Terminate after running for 24 hours".

Table 3.3 shows the average run-time (in seconds) of five algorithms. Although approximating the whole Pareto front, **COGNAC** is much faster than **COALA** and **AFDT**. When comparing with **AFDT2** and **MinCEntropy^{++}**, **COGNAC** is slower than these algorithms because in a single run, **COGNAC** searches for the whole Pareto front whereas the other two algorithms only compute one solution. However, the run-time of **COGNAC** is relatively small and scales up linearly with the dataset size and the number of generations.

Analyzing the Pareto front: We also apply the analysis procedure in Section 3.3.4 to identify meaningful alternative clusterings returned by **COGNAC**. In all datasets, we partition the Pareto front into 5 groups and check the border solutions. On the *CMUFaces* dataset, we perform another step of partitioning on the last group. Please see Appendix A.1 for the plots of all border solutions. As for the other algorithms, we select the solutions returned when running them with their default parameters.

In order to see whether the alternative clusterings discovered by **COGNAC** are similar to the expected alternative clusterings, on the *CMUFaces* and *WebKB* dataset, we compute the ratio of *dominant* poses and university-based documents in each cluster, respectively. Table 3.4a and 3.4b show the cluster statistics of alternative clusterings of five algorithms on the *CMUFaces* and *WebKB* dataset. As it can be seen, with the negative clusterings of 20 people for the *CMUFaces* dataset, and of 4 groups {course, faculty,

Algorithm	Cluster 1	Cluster 2	Cluster 3	Cluster 4
COALA	up(0.32)	left(0.87)	right(0.52)	right(0.58)
AFDT	up(0.30)	straight(0.34)	left(0.33)	right(0.30)
AFDT2	up(0.29)	straight(0.28)	left(0.32)	right(0.71)
MinCEntropy^{++}	up(0.54)	straight(0.75)	left(0.86)	right(0.89)
COGNAC	up(0.63)	straight(0.44)	left(0.70)	right(0.86)

(a) Alternative Clusterings of five algorithms on *CMUFaces*

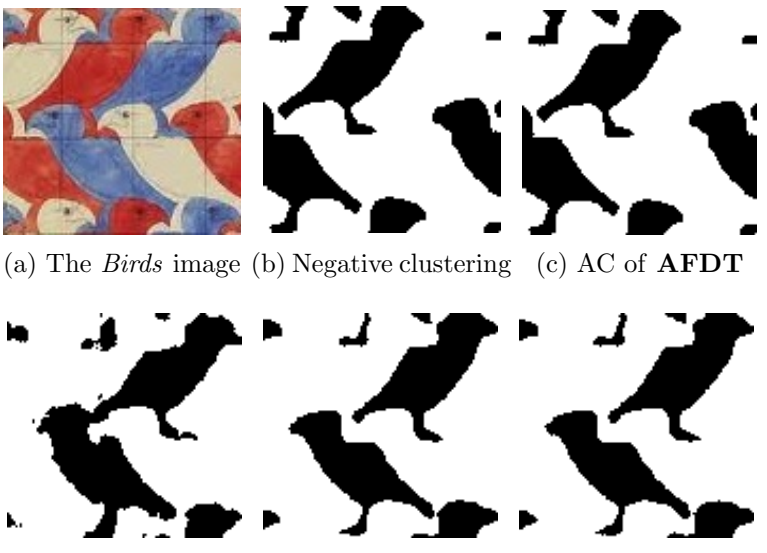
Algorithm	Cluster 1	Cluster 2	Cluster 3	Cluster 4
COALA	texas(0.85)	cornell(0.97)	wisconsin(0.93)	washington(0.94)
AFDT	texas(0.39)	texas(1.00)	cornell(0.98)	wisconsin(0.39)
AFDT2	wisconsin(0.50)	wisconsin(0.29)	wisconsin(0.67)	washington(1.00)
MinCEntropy^{++}	texas(1.00)	cornell(1.00)	wisconsin(0.69)	washington(1.00)
COGNAC	texas(0.98)	cornell(0.99)	wisconsin(0.82)	washington(0.97)

(b) Alternative Clustering on WebKB

Table 3.4: Alternative Clusterings on *CMUFaces* and *WebKB*

students, staff} for the *WebKB* dataset, **COGNAC** and **MinCEntropy**$^{++}$ produce the alternative clusterings matching closely to the expected alternative clusterings of the two datasets. On the *CMUFaces* dataset, **COALA** can only detects three alternative clusters {up, left, right}. Although **AFDT** and **AFDT2** can also discover all four alternative clusters of the *CMUFaces* dataset, their dominant ratios are much smaller than those of **COGNAC** and **MinCEntropy**$^{++}$. On the *WebKB* dataset, **COALA**, **MinCEntropy**$^{++}$, and **COGNAC** return the solutions which are very similar to the expected alternative clustering. In contrast, **AFDT** and **AFDT2** produce poor solutions in this case. Especially, **AFDT2** can only discover two alternative clusters.

As for the *Birds* and *Flowers* dataset, we select the images which are meaningful to human-eye interpretation from the border solution set of **COGNAC**. Fig.3.7f and 3.8f show these alternative clusterings. It can be seen that **COGNAC** has discovered successfully the high-quality al-



(a) The *Birds* image (b) Negative clustering (c) AC of **AFDT**

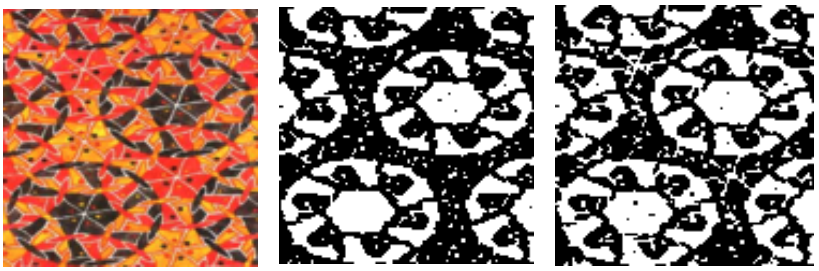
(d) AC of **AFDT2** (e) AC of **minCEntropy**$^{++}$ (f) AC of **COGNAC**

Figure 3.7: The alternative clusterings (AC) of four algorithms on the *Birds* dataset

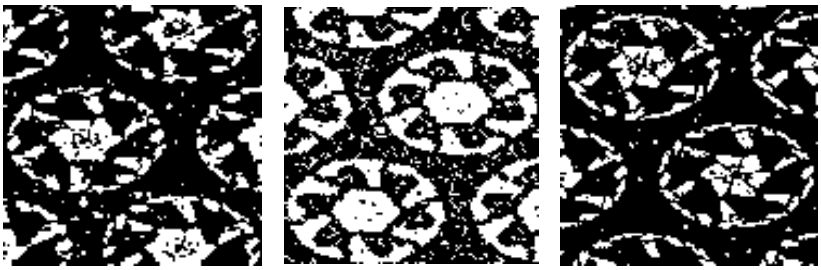
ternative clusterings of these images. Although **AFDT2** also finds two meaningful solutions on these datasets, its solutions are much noisier than those of **COGNAC**. Besides, **AFDT** fails on both datasets as its solutions (in Fig.3.7c and Fig.3.8c) are very similar to the negative clusterings. Likewise, on the *Flowers* dataset, **minCEntropy**$^{++}$ also returns the alternative solution (in Fig.3.8e) which is almost the same as the negative one.

3.4.3 Sequential Generation of Alternative Clusterings

In this section, we use a synthetic dataset with multiple clusterings and the *Flowers* dataset to illustrate the effectiveness of **COGNAC** for generating a set of different alternative clusterings (by applying the **SGAC** procedure in Algorithm 11). We also compare our algorithm with **MinCEntropy**$^{++}$ and **Alter-Spect**.



(a) The *Flowers* image (b) Negative clustering (c) AC of **AFDT**



(d) AC of **AFDT2** (e) AC of **minCEntropy**$^{++}$ (f) AC of **COGNAC**

Figure 3.8: The alternative clusterings of four algorithms on the *Flowers* dataset

Experimental Setup

The synthetic dataset consists of six Gaussian sub-clusters with the centroids at $\{(0,0),(6,0),(8,4),(6,8),(0,8),(-2,4)\}$ and the standard deviation of 0.5 for each coordinate. Each sub-clusters consists of 20 points as plotted in Fig.3.9. Because this dataset has six natural sub-clusters, when

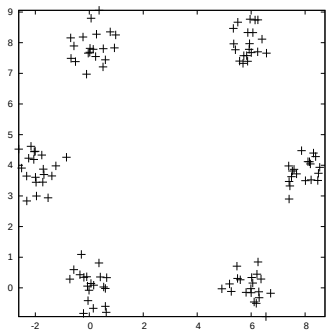


Figure 3.9: The *Six-Gaussians* dataset.

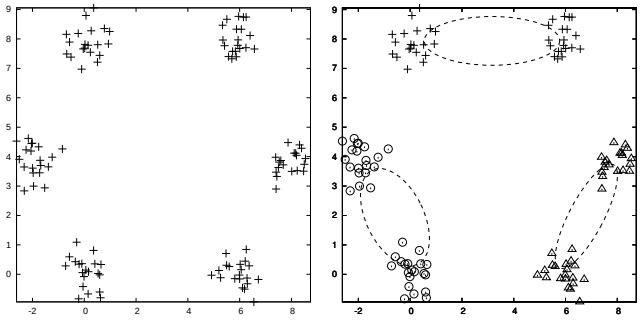
setting the number of clusters to three, there are many possible ways to

partition it. The parameters of **COGNAC** are set as in Table 3.2. For **MinCEntropy**$^{++}$, the parameter m (declaring that quality is m times important than dissimilarity) is set to its default value. We run both **COGNAC** and **MinCEntropy**$^{++}$ 10 times with different random seeds and record the best results. We apply the analysis procedure in Section 3.3.4 to select the best alternative clustering solution.

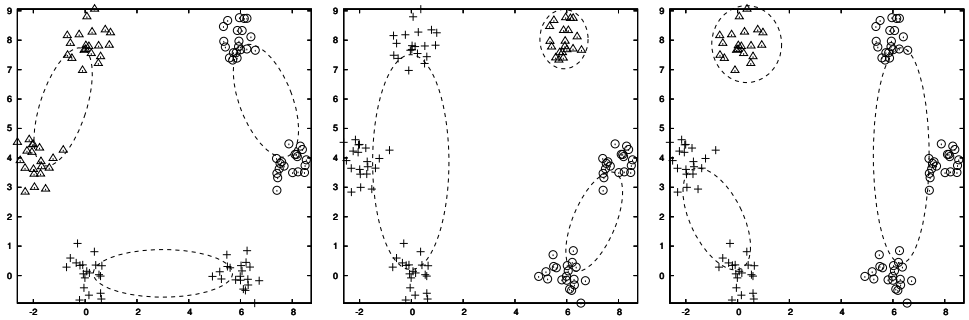
Experimental Results

We run **COGNAC**, **MinCEntropy**$^{++}$ and **Alter-Spect** on the *Six-Gaussians* dataset with the initial negative clustering as in Fig.3.10b. The sets of different alternative clusterings returned by three algorithms are shown in Fig.3.10. We plot one more solution for **Alter-Spect** because its first solution can be very similar to the negative clustering. It can be seen that the alternative clusterings obtained by **COGNAC** are very different from each other and of high quality. On the contrary, **MinCEntropy**$^{++}$ can only generate the first two alternative clusterings in Fig.3.10f, Fig.3.10g. The third solution generated by **MinCEntropy**$^{++}$ in Fig.3.10h is very similar to the first solution in Fig.3.10f with some mistakes. Although **Alter-Spect** can also produce a set of alternative clusterings, these solutions are less meaningful to human interpretation because very far sub-clusters are grouped together, as in the third solution in Fig.3.10k.

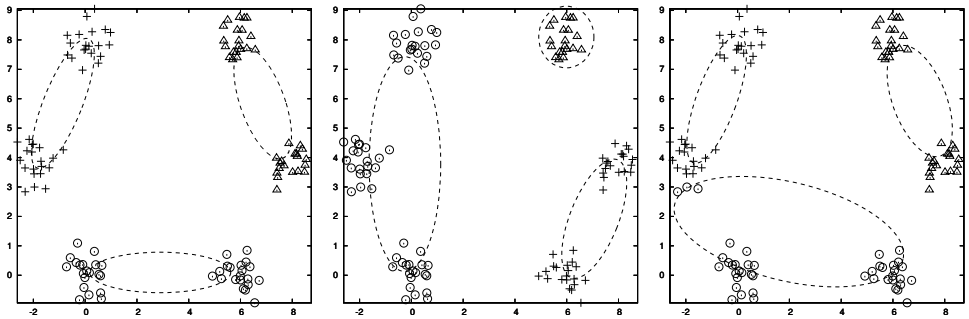
The results on the *Flowers* dataset are plotted in Fig.3.11. **COGNAC** successfully discovers the other two alternative clusterings (with red and yellow colors) as depicted in Fig.3.11c, 3.11d. On the contrary, **minCEntropy**$^{++}$ produces alternative clusterings which are very similar to the negative clustering, as plotted in Fig.3.11e, 3.11f. As for **Alter-Spect**, only its second solution in Fig.3.11h is meaningful because the other two solutions are very similar to the negative clustering.



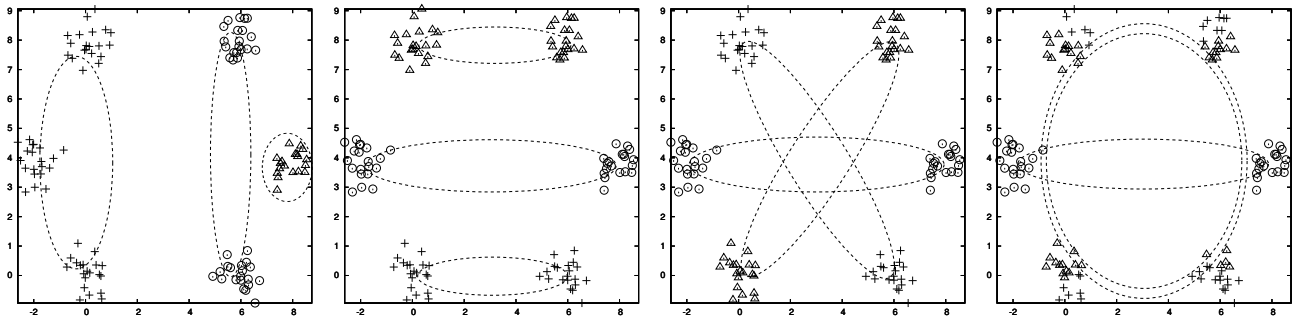
(a) The Six-Gaussians dataset (b) The negative clustering.



(c) 1st AC of **COGNAC** (d) 2nd AC of **COGNAC** (e) 3rd AC of **COGNAC**

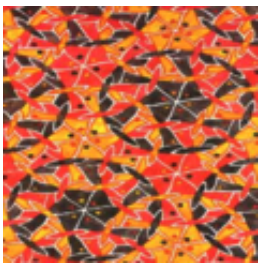


(f) 1st AC of **minCEntropy**$^{++}$ (g) 2nd AC of **minCEntropy**$^{++}$ (h) 3rd AC of **minCEntropy**$^{++}$

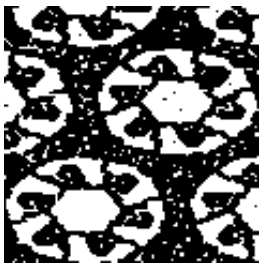


(i) 1st AC of **Alter-Spect** (j) 2nd AC of **Alter-Spect** (k) 3rd AC of **Alter-Spect** (l) 4th AC of **Alter-Spect**

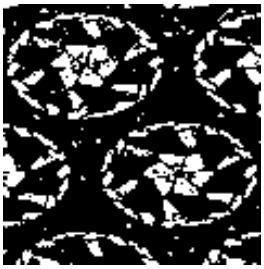
Figure 3.10: Alternative clusterings (AC) of three algorithms on the *Six-Gaussians* dataset.



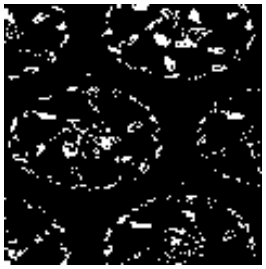
(a) The *Flowers* dataset.



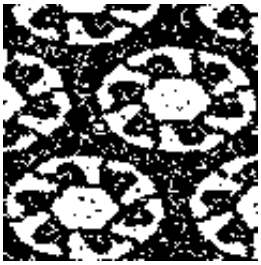
(b) The negative clustering.



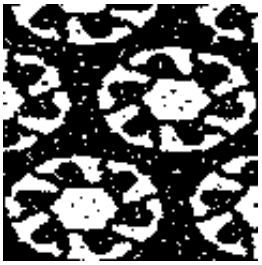
(c) 1st AC of **COGNAC**



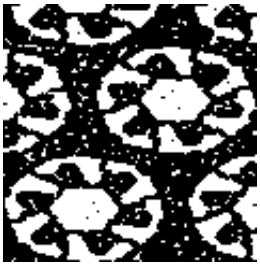
(d) 2nd AC of **COGNAC**



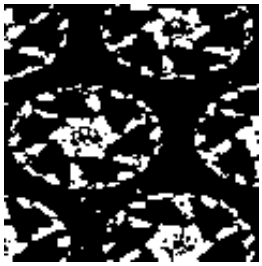
(e) 1st AC of **minCEntropy**$^{++}$



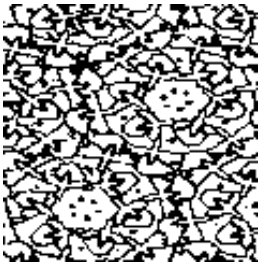
(f) 2nd AC of **minCEntropy**$^{++}$



(g) 1st AC of **AlterSpect**



(h) 2nd AC of **AlterSpect**



(i) 3rd AC of **AlterSpect**

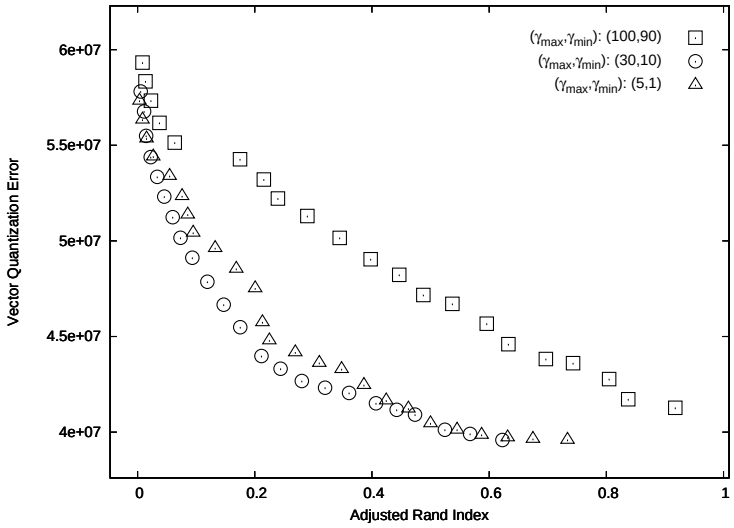
Figure 3.11: Alternative clusterings of three algorithms on the *Flowers* dataset.

3.4.4 Parameter Sensitivity Analysis

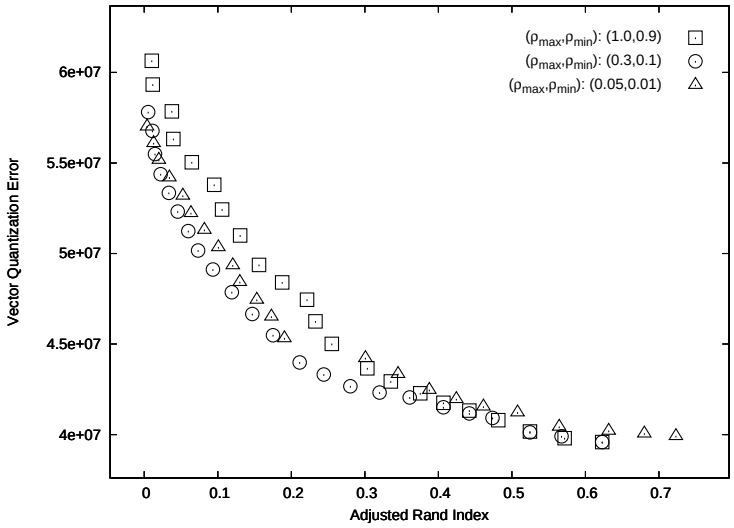
In this section, we perform three experiments to study the sensitivity of **COGNAC** to parameters $\rho_{max}, \rho_{min}, \gamma_{max}, \gamma_{min}$. In all experiments, we fix the number of generations to 100 and run the algorithm on the reduced *CMUFaces* dataset containing images of four people (an2i, at33, boland, ch4f). Besides, there is no parameter in the *Cluster-Oriented Recombination* operator, therefore we only use the *Neighbor-Oriented Mutation* operator to modify the solutions,

We first fix the perturbation probability (ρ_{max}, ρ_{min}) to $(0.3, 0.1)$ and set the maximum and minimum number of nearest neighbors $(\gamma_{max}, \gamma_{min})$ to $(100, 90)$, $(30, 10)$, $(5, 1)$, respectively. The first setting with large values can be considered as the representative for the traditional mutations, where an object can be assigned to arbitrary (or very far) clusters. As it can be seen in Fig.3.12a, setting $(\gamma_{max}, \gamma_{min})$ to very large value like $(100, 90)$ leads to the poorest performance. Because in this case, **COGNAC** can put very far objects to the same cluster but according to local structures of the dataset, the near objects are often partitioned in the same cluster. However, setting $(\gamma_{max}, \gamma_{min})$ to very small value like $(5, 1)$ can make the algorithm stuck at local minima, hence a moderate setting of $(\gamma_{max}, \gamma_{min})$ like $(30, 10)$ results in the best performance.

To study the sensitivity to the perturbation probability, we fix the number of nearest neighbors $(\gamma_{max}, \gamma_{min})$ to $(30, 10)$, and set the perturbation probability (ρ_{max}, ρ_{min}) to $(1.0, 0.9)$, $(0.3, 0.1)$, and $(0.05, 0.01)$. Fig.3.12b shows that setting (ρ_{max}, ρ_{min}) to large values like $(1.0, 0.9)$ results in the poorest performance because large perturbation levels can destroy the good properties of current objects. However, small perturbation levels are not enough for **COGNAC** to escape from local minima, therefore similarly to the case of the number of nearest neighbors $(\gamma_{max}, \gamma_{min})$, a moderate



(a) **COGNAC** performance on the *CMUFaces* dataset with different configurations on the number of nearest neighbors.



(b) **COGNAC** performance on the *CMUFaces* dataset with different configurations on the perturbation probability.

Figure 3.12: **COGNAC** performance on the *CMUFaces* dataset

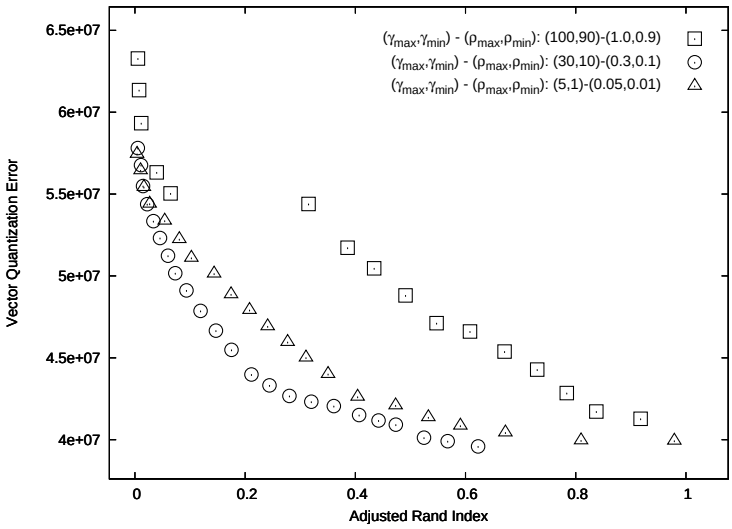


Figure 3.13: **COGNAC** performance on the *CMUFaces* dataset with different configurations.

setting often results in the best performance as presented in Fig.3.12b.

Finally, we compare three configurations of $\{(\gamma_{max}, \gamma_{min}) - (\rho_{max}, \rho_{min})\}$ in the decreasing order of perturbation level which are: $\{(100, 90) - (1.0, 0.9)\}$, $\{(30, 10) - (0.3, 0.1)\}$, $\{(5, 1) - (0.05, 0.01)\}$. The middle configuration $\{(30, 10) - (0.3, 0.1)\}$ outperforms the other configurations, as shown in Fig.3.13.

3.5 Conclusion

In this chapter, we proposed an explicit multi-objective algorithm for alternative clustering, called **COGNAC** and a derived algorithm called **SGAC** for the sequential generation of alternative clusterings. **COGNAC** and **SGAC** not only provide solutions outperforming those produced by other state-of-the-art algorithms, but they also possess attractive features. Firstly, they are very flexible and they can accept arbitrary objectives, therefore they can be used as a baseline when comparing with different alternative clustering algorithms. In addition, **SGAC** can be used to gen-

erate a *sequence* of alternative clusterings by adding the newly found alternative clustering to the negative clustering set and re-running **COGNAC**. Each newly generated clustering is guaranteed to be different from the previously found ones. Finally, **COGNAC** approximates the whole Pareto front in a single run, but its complexity only scales up linearly with the dataset size, when deployed with two objectives VQE and ARI.

Currently, **COGNAC** simply returns the whole Pareto front to the user. Then, the user applies some techniques proposed in this chapter to analyze and visualize the obtained solution set. Although judging the best solution from the Pareto front is a responsibility of the users, helping users to explore the Pareto front *interactively* in a more efficient manner is an interesting and non-trivial task. In the future, we plan to integrate interactive techniques in multi-objective optimization [11, 17] with our algorithm so that users can quickly direct the search to explore the regions of their interest.

Chapter 4

Non-Redundant Overlapping Subspace Clustering

Traditional clustering approaches based on feature selection techniques are often not suitable for high-dimensional datasets, as different feature subsets may be relevant to different clusters. Subspace clustering algorithms aim at the simultaneous identification of a cluster and of a subspace of features associated with it. However, these methods are often designed and optimized for specific cases of L^p spaces, e.g., based on the Manhattan or Euclidean distance, and cannot be easily applied to new domains with different underlying models, like gene expression analysis in bioinformatics. In contrast, specialized algorithms for biclustering on gene expression data also use very particular models which cannot be generalized to other applications on L^p spaces. Besides, most of the state-of-the-art algorithms can either generate a disjoint set of subspace clusters or a very large set of possible clusters that can overlap significantly.

In this chapter, we propose a novel algorithm, **FLEXBIC**, which can be applied to different domains without modifying its overall structure. In addition, it can sequentially generate multiple overlapping subspace clusters where the maximum overlap is below a predefined threshold, and it also allows users to control bicluster shapes by adjusting the relative importance

of rows and columns. Experiments on real-world datasets demonstrate that **FLEXBIC** performs significantly better than the other state-of-the-art algorithms. The source codes, datasets and other supplementary materials of our experiments can be found at `http://lion.disi.unitn.it/intelligent-optimization/flexbic/`.

4.1 Introduction

The goal of clustering is to associate the objects to different sets (clusters) such that the objects in the same cluster are similar and the ones in different clusters are dissimilar. Clustering is a fundamental building block in science and in general human activities (distinguishing group of entities with different names —like for different animal species— is in fact a form of clustering). In traditional clustering approaches, the similarity between two objects is computed by taking into account *all* features representing them, by using the Euclidean metric or generalizations thereof. However, this approach is not suitable for several real-world cases with high-dimensional feature spaces where interesting clusters can be observed in different subspaces, and where the similarity between items is not based on traditional p-norms.

As different clusters can exist in different subspaces, preprocessing the datasets by feature selection techniques does not solve the problem. This motivates *subspace clustering* algorithms which aim at identifying simultaneously *both* objects in each cluster *and* the subspace of features associated with it. For this reason, subspace clusters are also termed *biclusters* [19].

Several algorithms for biclustering are based on p-norm distances like Manhattan or Euclidean [58, 68]. Three approaches can be distinguished: grid-based, density-based, and projected-subspace methods [68]. In grid-based approaches, the feature space is discretized and each subspace cluster

is defined as a set of connected grid cells where each cell contains a number of objects greater than a threshold [98]. Other researchers extend the notion of density-connectivity of DBSCAN [34] for subspace clustering: clusters are specified as dense regions separated from sparse ones, and the distance between objects is computed only on the relevant dimensions of those clusters [55]. Unlike the two previous schemes searching for individual clusters, the projected-subspace approaches aim at discovering a whole set of clusters at once, by optimizing the clustering quality based on some criteria [1]. Again, the methods are often designed and optimized for specific distance metrics like Manhattan [1, 98], or Euclidean [34]. Therefore, it is difficult to apply them to new datasets or domains with different underlying models, among which gene expression analysis in bioinformatics is a particularly prominent application.

Let us summarize the bioinformatics context. Gene expression is the process by which information from a gene is used in the synthesis of proteins. Microarray experiments provide the expression level of a large number of genes under different experimental conditions [4]. From the gene expression data, one would like to find maximal *subsets* of genes and *subsets* of conditions, where the genes exhibit highly similar patterns (*co-regulation* and *co-expression*) under these conditions. This problem is called *biclustering* by Cheng and Church [19]. Several algorithms have been proposed for gene expression biclustering based on special models [19, 61, 65], and are therefore difficult to extend for other applications. Besides, they often only either produce disjoint [19] or redundant biclusters with arbitrary overlap [61, 97]. However, in practice, some genes can belong to multiple functional categories, thus the biclusters extracted from a gene expression matrix should be allowed to overlap, while maintaining the overlap below a predefined threshold to avoid getting too similar —and therefore redundant— biclusters. This problem is similar to *alternative clustering*,

with the goal of generating high-quality clusterings different from a given trivial set [42, 87]. However, alternative clustering algorithms cannot be used directly for non-redundant biclustering because redundancy is measured between whole clusterings, and not between biclusters. Günnemann et al. propose an algorithm to search for non-redundant overlapping biclusters in high-dimensional spaces [46]. However, their model is not suitable for analyzing gene expression data as dissimilarity is measured by the Euclidean distance.

In this chapter, we present a FLEXible BIClustering algorithm, named **FLEXBIC**, which can return multiple biclusters such that the overlap between each pair is less than a user-defined threshold. While other algorithms often search for K biclusters at once [61, 97], **FLEXBIC** produces one bicluster in each of K iterations. Each newly generated bicluster is different from the previous ones but can overlap with them in a controlled manner. This work builds upon basic versions [88, 89] limited to the application to gene expression data, and extends the approach to design a general-purpose, flexible algorithmic framework able to handle more general applications. Besides, this new method also allows users to control bicluster shapes, i.e., to specify the preference for clusters with more rows vs. clusters with more columns, and therefore to avoid suboptimal results if only the bicluster's overall size is considered.

The rest of this chapter is organized as follows. In Section 4.2, we formally define the problem of non-redundant subspace clustering. Then, we describe our algorithm in Section 4.3. Finally, we present the application of our algorithms to the search for disjoint subspace clusters with the Euclidean metric in Section 4.4 and to discover non-redundant overlapping biclusters on gene expression data in Section 4.5.

4.2 Non-Redundant Overlapping Subspace Clustering Problem

Following the notation in [65], let $A = (X, Y)$ be a data matrix with the set of rows X and the set of columns Y, a_{ij} be the value of a cell in the matrix A representing the relation between row i and column j. A subspace cluster is represented by a submatrix (I, J). To abstract a notion of "coherence" of a cell value with the other cells of the cluster we proceed in two steps. First, we introduce modeling functions $\mathbb{M} : \mathbb{R}^{|X||Y|} \times \mathbb{N}^{|I|} \times \mathbb{N}^{|J|} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ predicting the proper value of a coherent cell in the cluster from the other contained cells: $\hat{a}_{ij} = \mathbb{M}(A, I, J, i, j)$, where $\hat{a}_{ij}$ is the estimated value for the cell a_{ij} by the model $\mathbb{M}$. Then we measure the dissimilarity between the modeled value and the actual cell value. Let's note that we permit a slight abuse of notation: the modeling functions depend on the number of rows and columns in the cluster but have the same structural form.

Definition 5 *A* bicluster *or* subspace cluster *is a submatrix* (I, J) *(where* $I \subset X$*,* $J \subset Y$*) whose error* $Err_{\mathbb{M}}(I, J)$ *with respect to predictive model* $\mathbb{M}$ *is below a given threshold* δ*.*

Depending on applications and data, different dissimilarity or error functions can then be used to measure the discrepancy between the actual cell values a_{ij} and the values $\hat{a}_{ij}$ predicted by model $\mathbb{M}$. In this chapter, we consider the error function given as the cell error average:

$$\mathrm{Err}_{\mathbb{M}}(I, J) = \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} \mathrm{cellErr}_{\mathbb{M}}(a_{ij}, \hat{a}_{ij}) \tag{4.1}$$

where $\mathrm{cellErr}_{\mathbb{M}}(a_{ij}, \hat{a}_{ij})$ measures the approximation error of the model $\mathbb{M}$ for the data cell a_{ij}. Some examples of $\mathrm{cellErr}_{\mathbb{M}}(a_{ij}, \hat{a}_{ij})$ are $|a_{ij} - \hat{a}_{ij}|$ or $(a_{ij} - \hat{a}_{ij})^2$. Similarly, given a bicluster (I, J), the error or residue of a row

i or a column j are defined as:

$$\text{rowErr}_{\mathbb{M}}^{IJ}(i) = \frac{1}{|J|} \sum_{j \in J} \text{cellErr}_{\mathbb{M}}(a_{ij}, \hat{a}_{ij}) \tag{4.2}$$

$$\text{colErr}_{\mathbb{M}}^{IJ}(j) = \frac{1}{|I|} \sum_{i \in I} \text{cellErr}_{\mathbb{M}}(a_{ij}, \hat{a}_{ij}). \tag{4.3}$$

Definition 6 *Biclustering or Largest Subspace Cluster Problem — Given a data matrix $A = (X, Y)$, a model $\mathbb{M}$ and an error function $Err_{\mathbb{M}} : 2^X \times 2^Y \to R$, the biclustering problem is the problem of searching for the largest bicluster (I, J):*

$$\begin{aligned} &\textit{Maximize:} && size(I, J) = |I||J| \\ &\textit{subject to:} && Err_{\mathbb{M}}(I, J) \leq \delta \\ & && I \subset X, J \subset Y. \end{aligned} \tag{4.4}$$

Some formulations of the maximum bicluster problem have been proven to be NP-hard, starting from Cheng and Church [19].

Definition 7 *Overlap Between Two Biclusters*
The overlap of a bicluster (I_1, J_1) on a bicluster (I_2, J_2) is:

$$Overlap\big((I_1, J_1), (I_2, J_2)\big) = \frac{|I_1 \cap I_2||J_1 \cap J_2|}{|I_1||J_1|}. \tag{4.5}$$

Definition 8 *Overlap Between A Bicluster and A Set of Biclusters*
The overlap of a bicluster (I_1, J_1) with a set $\boldsymbol{B}$ of biclusters is:

$$Overlap\big((I_1, J_1), \boldsymbol{B}\big) = \max_{(I_2, J_2) \in \boldsymbol{B}} Overlap\big((I_1, J_1), (I_2, J_2)\big). \tag{4.6}$$

Definition 9 *Non-Redundant Subspace Clustering Problem*
Given a data matrix $A = (X, Y)$, a model $\mathbb{M}$ and an error function $Err_{\mathbb{M}} : 2^X \times 2^Y \to \mathbb{R}$, a set of biclusters $\boldsymbol{B}$, the non-redundant subspace clustering

problem is the problem of searching for the largest bicluster (I, J) *such that the overlap with the other biclusters in* $\boldsymbol{B}$ *less than a threshold* γ*:*

$$\begin{aligned} &\textit{Maximize:} && \textit{size}(I, J) = |I||J| \\ &\textit{subject to:} && \textit{Err}_{\mathbb{M}}(I, J) \leq \delta \\ & && \textit{Overlap}\big((I, J), \boldsymbol{B}\big) \leq \gamma \\ & && I \subset X, J \subset Y. \end{aligned} \tag{4.7}$$

Given an algorithm that solves this problem, we can extract a set of K non-redundant biclusters by rerunning it for K times. The set $\boldsymbol{B}$ of constraining biclusters can be empty at the beginning. Then, after each run, the newly discovered bicluster is added to $\boldsymbol{B}$ to guarantee that the next bicluster will be sufficiently different from the members of $\boldsymbol{B}$. As the biclustering problem is NP-hard, we will present a heuristic algorithm which can find reasonably good solutions in polynomial time.

4.3 A Core-Node Search Algorithm for Non-Redundant Subspace Clustering

FLEXBIC is based on a *repeated local search* module for searching for the largest bicluster which is presented in this section. Later, this module will be used to discover K *non-redundant* biclusters.

4.3.1 Repeated Local Search for Largest Bicluster

The pseudocode of our REpeated Local Search module for BIClustering, named **BICRELS**, is shown in Algorithm 12. We first generate the initial bicluster *seeds* by combining rows and column clusters. In detail, we partition the row set into K_r clusters by applying the K-means procedure [62] on the column-normalized data where each row is an instance, and

Algorithm 12: BICRELS

Input : data matrix A, residue threshold δ

Output: A bicluster (I, J)

```
begin
    pool = create a set of initial seed biclusters.
    largestBicluster = ∅
    for i = 1 to numberOfRestarts do
        bicluster = pick randomly a bicluster from pool.
        Remove bicluster from pool.
        bicluster = replaceNodes(bicluster)
        bicluster = deleteNodes(bicluster, δ)
        repeat
            bicluster = replaceNodes(bicluster)
            bicluster = addNodes(bicluster, δ)
        until no change;
        bicluster = deleteNodes(bicluster, δ)
        if |bicluster| > |largestBicluster| then
            largestBicluster = bicluster
        end
    end
end
return largestBicluster
```

each column is a feature. The normalization to eliminate the difference in attribute ranges is obtained by subtracting the mean value from each column and then dividing the results by its standard deviation. Similarly, we divide the columns into K_c clusters on the row-normalized data. Then, we pick randomly a cluster of row and a cluster of column to form a bicluster. In the experiments of this chapter, we create $K_r \times K_c$ seed biclusters.

Algorithm 13: replaceNodes

Input : (I, J) are the sets of rows and columns
Output: (I', J') with smaller or equal residue

begin
  repeat
    // Replace columns
    repeat
      $maxJ = \arg\max_{j \in J} \text{colErr}_{\mathbb{M}}^{IJ}(j)$
      $minJ = \arg\min_{j \in Y \setminus J} \text{colErr}_{\mathbb{M}}^{IJ}(j)$
      $J' = J \cup \{minJ\} \setminus \{maxJ\}$
      if $Err_{\mathbb{M}}(I, J) > Err_{\mathbb{M}}(I, J')$ **then**
        $J = J'$
      end
    until *J is not modified*;
    // Replace rows
    repeat
      $maxI = \arg\max_{i \in I} \text{rowErr}_{\mathbb{M}}^{IJ}(i)$
      $minI = \arg\min_{i \in X \setminus I} \text{rowErr}_{\mathbb{M}}^{IJ}(i)$
      $I' = I \cup \{minI\} \setminus \{maxI\}$
      if $Err_{\mathbb{M}}(I, J) > Err_{\mathbb{M}}(I', J)$ **then**
        $I = I'$
      end
    until *I is not modified*;
  until *I, J are not modified*;
end
return (I, J)

The local search procedure (line 4 to 14) is restarted for a number of runs to explore different local minima. The normalized data is used only to create the initial bicluster set, while local search runs on the original data. In each run, the algorithm first picks randomly a bicluster from the initial set and removes it. Next, it reduces the bicluster's residue by the procedure **replaceNodes**. Function **replaceNodes** shrinks the residue by replacing the column (or row) with the highest residue in that bicluster by an external column (or row) with the smallest residue, if the replacement reduces the residue. The replacement is repeated until a locally minimal residue is obtained. If the residue is still greater than the threshold, some rows or columns are deleted in the **deleteNodes** procedure of Algorithm 14, following the proposal by Cheng and Church [19]. **deleteNodes** keeps deleting the columns and rows with highest mean residues, until the residue drops below the threshold. The parameter θ declares the relative importance between rows and columns. A large value of θ leads to biclusters with more columns, a smaller value to biclusters with more rows. The default value $\theta = 1$, implies equal weights of rows and columns. Although both **replaceNodes** and **deleteNodes** can decrease the residue, **replaceNodes** keeps the bicluster size unchanged whereas **deleteNodes** reduces it.

At this point, the residue is guaranteed to be lower than or equal to the threshold. The algorithm starts repeating two steps: **replaceNodes** and **addNodes** until convergence. The main scheme is that while fixing the bicluster volume, we try to reduce the residue and then while keeping the residue below the threshold, we try to increase the volume. The **addNodes** procedure in Algorithm 15 iteratively adds a column or a row with the smallest residue, until the residue exceeds the threshold. Finally, to guarantee that the residue is less than or equal to the threshold, we perform **deleteNodes** before returning the bicluster. As the number of rows and columns is finite, the loop of two steps **replaceNodes** and **addNodes**

Algorithm 14: deleteNodes

Input : (I, J) are the sets of rows and columns, threshold δ, preference for columns θ

Output: (I', J') with residue smaller than threshold δ

begin
  while $Err_{\mathbb{M}}(I, J) > \delta$ **do**
    $maxJ = \arg\max_{j \in J} \text{colErr}_{\mathbb{M}}^{IJ}(j)$
    $maxI = \arg\max_{i \in I} \text{rowErr}_{\mathbb{M}}^{IJ}(i)$
    if $colErr_{\mathbb{M}}^{IJ}(maxJ) < \theta rowErr_{\mathbb{M}}^{IJ}(maxI)$ **then**
      $I = I \setminus \{maxI\}$
    else
      $J = J \setminus \{maxJ\}$
    end
  end
end
return (I, J)

Algorithm 15: addNodes

Input : (I, J) are the sets of rows and columns, threshold δ

Output: (I', J') with greater or equal size

begin
  while $Err_{\mathbb{M}}(I, J) < \delta$ **do**
    $minJ = \arg\min_{j \in Y \setminus J} \text{colErr}_{\mathbb{M}}^{IJ}(j)$
    $minI = \arg\min_{i \in X \setminus I} \text{rowErr}_{\mathbb{M}}^{IJ}(i)$
    if $colErr_{\mathbb{M}}^{IJ}(minJ) < \theta rowErr_{\mathbb{M}}^{IJ}(minI)$ **then**
      $J = J \cup \{minJ\}$
    else
      $I = I \cup \{minI\}$
    end
  end
end
return I, J

always terminates after a finite number of iterations (which is less than or equal to $(|X| + |Y|)$).

4.3.2 Discovering Non-Redundant Biclusters

Local Minima Distribution

Repeated local search produces locally optimal solutions (local minima), which cannot be further improved by local changes. In many cases, *kicking* the system out of local minima but still searching in their vicinity can be effective to discover even better solutions [10]. This is caused by the "big valley" hypothesis of local search: in many relevant problems, local minima tend to be clustered with a rich structure, and Iterated local Search (ILS) methods aiming at moving between nearby local minima are particularly efficient.

To motivate our proposal for discovering non-redundant biclusters, and to validate the "big valley" hypothesis for our problem, we study the local minimum distribution. We run **BICRELS** for 1000 restarts with random clusters of columns and rows on the *Yeast* dataset [84]. The details of the experimental setup are presented in Section 4.5.3. Let X, Y be the set of columns and rows of the dataset. Each random cluster of rows or columns are sampled uniformly from the power set 2^Y and 2^X of rows and columns. In this case, we use random row and column clusters instead of the ones obtained by K-means to have a uniform distribution of initial solutions. All duplicated local minima are removed from the analysis. We measure the distance between two local minima (I_1, J_1) and (I_2, J_2) by the Jaccard distance:

$$\begin{aligned}&\text{JaccardDistance}\big((I_1, J_1), \big(I_2, J_2\big)\big) = \\ &= 1 - \frac{|I_1 \cap I_2||J_1 \cap J_2|}{|I_1||J_1| + |I_2||J_2| - |I_1 \cap I_2||J_1 \cap J_2|}.\end{aligned}$$

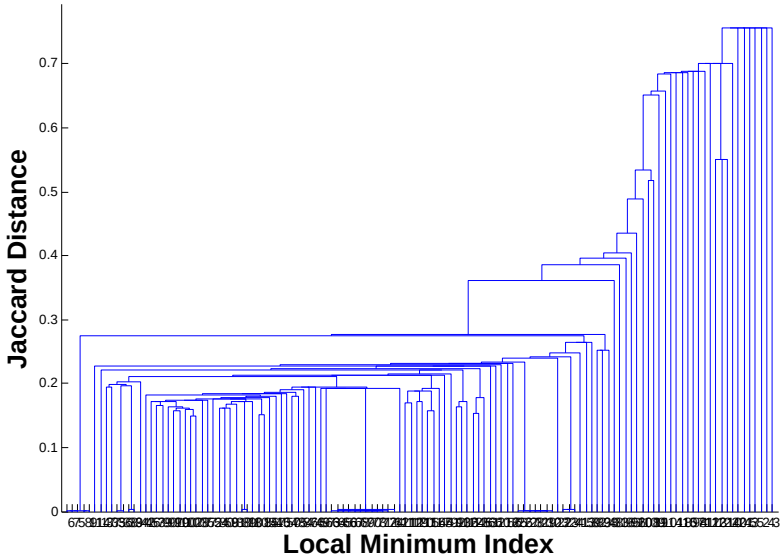


Figure 4.1: Local Minimum Hierarchical Cluster Tree

A small Jaccard distance between two local minima means that the number of overlapping cells between them is large.

For the sake of convenience, we shall refer to a row or a column as a "node" hereafter. The smallest (or largest) node is the one with the lowest (or highest) residue. A node rank is the order in which that node appears in the ascending order of all node residues, i.e., the smallest node is the one having the smallest rank and vice versa. We plot hierarchical cluster tree of the local minima obtained by running a Hierarchical Clustering algorithm (using the Single Linkage method) [80] in Fig.4.1. The second column of Table 4.1 shows the statistics on the percentage of local minima (I, J) sharing at least $(1 - \text{JaccardDistance})|I||J|$ cells with another local minimum where $|I||J|$ is the number of cells of such local minima. The third column presents the average percentage of common nodes whose ranks less than $(1 - \text{JaccardDistance})(|I| + |J|)$. It can be seen that a large portion of local minima have a significant overlap with other local minima, e.g., 76.98% of local minima share at least 70% of their cells with at least another local minimum; and 95.40% of the common nodes having ranks less than

Jaccard Distance	Percentage of Local Minima (%)	Percentage of Smallest Nodes (%)
0.1	26.19	100.00
0.2	63.49	98.68
0.3	76.98	95.40
0.4	79.37	88.42
0.5	81.75	78.87
0.6	85.71	65.81
0.7	92.86	49.70
0.8	100.00	32.52

Table 4.1: The second column shows the percentage of local minima (I, J) sharing at least $(1 - \text{JaccardDistance})|I||J|$ cells with another local minimum where $|I||J|$ is the number of cells of such local minima. The third column presents the average percentage of common nodes whose ranks less than $(1 - \text{JaccardDistance})(|I| + |J|)$.

$0.7(|I| + |J|)$ or the overlap contains mostly the rows and columns with smallest residues.

The above quantitative results demonstrate that an optimal solution is often surrounded by other near locally optimal solutions and shares some common structures with them. To discover such optimal solutions from a local minimum, an effective algorithm should therefore modify the current solution to move to nearby local optima rather than restarting from scratch.

From this observation, we develop our FLEXible BIClustering algorithm, called **FLEXBIC**, based on the notion of core rows or columns in a bicluster.

The core nodes

In a δ-bicluster (I, J), the nodes can be divided into two groups: core nodes and loose nodes. In detail, a row $i \in I$ is a core row if and only if:

$$\text{rowErr}_{\mathbb{M}}^{IJ}(i) \leq \beta_r$$
$$\text{where}$$
$$0 \leq \beta_r < \max_{i \in I}(\text{rowErr}_{\mathbb{M}}^{IJ}(i)).$$

Similarly, a column $j \in I$ is a core column if and only if:

$$\text{colErr}_{\mathbb{M}}^{IJ}(j) \leq \beta_c$$
$$\text{where}$$
$$0 \leq \beta_c < \max_{j \in J}(\text{colErr}_{\mathbb{M}}^{IJ}(j)).$$

For a pair of (β_r, β_c), we can identify the boundary between the core and loose nodes. Fig.4.2 shows an example of core nodes for a specific pair of (β_r, β_c). The blue rows and the columns from 1 to 6 are core nodes because these rows show more similar patterns in these columns. The remaining rows and columns are loose nodes.

For the visual presentation, we often rearrange rows and columns of a δ-bicluster in ascending order of their residues as in Fig.4.3a. After sorting, the core and loose nodes are traversed from left to right and from top to bottom as in Fig.4.3b. Because of the tight relationship between core nodes, i.e., some rows can only show similar behaviors through some columns, the core-node set of a bicluster is often included also in other locally optimal biclusters in the neighborhood. Fig.4.3c and 4.3d illustrate this relationship (assuming that we can rearrange the rows and columns of two biclusters). The first bicluster consists of the red and blue part. The second one is the combination of the red and green part. The two share a core set denoted as the black part. Therefore, when we expand a

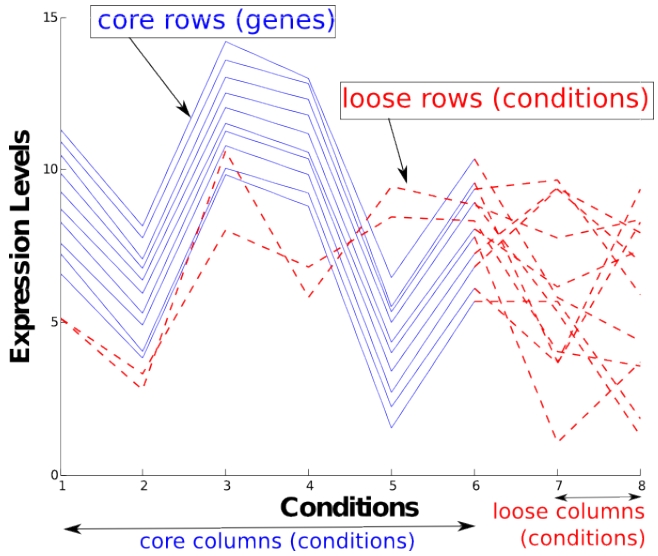


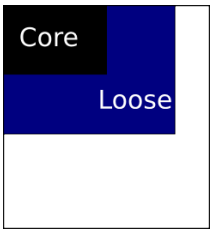
Figure 4.2: A core node example of a δ-bicluster.

bicluster around its core set, we can quickly identify a family of locally-optimal biclusters. We do not have to build or search from scratch. Besides, different core sets can exist in different regions of the data matrix. Thus, discovering these core sets and searching in the neighborhood helps to extract the representative biclusters of the data matrix.

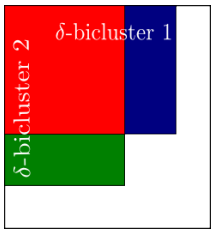
The Flexible Biclustering Algorithm

FLEXBIC has three main steps which are motivated and illustrated in Fig.4.4. At the beginning, local search is used to identify the first locally optimal bicluster denoted by *Core* 1 and *Loose* 1 in Fig.4.4a. From there, it searches for other locally optimal solutions around the core set by expanding columns (and reducing rows) or reducing columns (and expanding rows). After identifying the optimal bicluster around the first core set (denoted by *Core* 1^* and *Loose* 1^* in Fig.4.4b), it restricts the search area in the largest submatrix (the grey area in this case) having no intersection with the first optimal bicluster. The whole process is repeated to identify the next optimal bicluster as in Fig.4.4c and Fig.4.4d. Note that the second

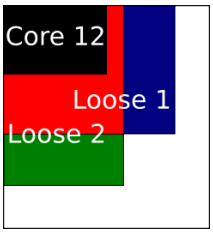
(a) The arrangement of all nodes in a δ-bicluster in a sequential order.



(b) The arrangement of all nodes in a δ-bicluster according to their residues.



(c) The arrangement of all nodes of two neighborhood δ-biclusters in a sequential order.



(d) The arrangement of all nodes of two δ-biclusters according to their residues.

Figure 4.3: The arrangement of all nodes in δ-biclusters.

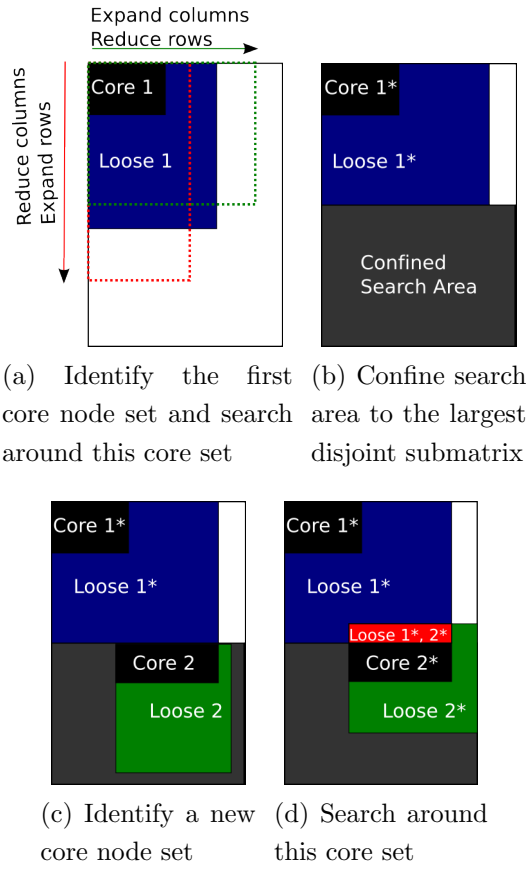
optimal bicluster (consisting of *Loose* $1^*, 2^*$, *Core* 2^*, *Loose* 2^* in Fig.4.4d) *overlaps* with the first optimal one (in the *Loose* $1^*, 2^*$ part) because of the expansion steps.

Intuitively, restricting the search area guarantees that the sub-matrix formed by the new core set is disjoint from those formed by previous core sets. This new sub-matrix thus reveals a non-redundant view of the data matrix. The algorithm subsequently expands around this core set to search for overlapping nodes.

The pseudo-code of **FLEXBIC** is shown in Algorithm 16. In each iteration, the algorithm searches for the largest bicluster satisfying the residue and overlap constraints.

Algorithm 16: FLEXBIC

$\boldsymbol{B} = \emptyset$
for $k = 1$ *to* $numberOfBiclusters$ **do**
  // **Restrict search region**
  $(\hat{I}, \hat{J}) = restrictRegion(A, \boldsymbol{B})$
  // **Identify the core bicluster**
  $(I, J) =$ **BICRELS**$(A_{\hat{I}\hat{J}}, \delta)$
  $(I^*, J^*) = (I, J)$
  // **Search for variants of the core bicluster**
  for $n = (1 - \alpha_1)|J|$ *to* $(1 + \alpha_2)|J|$ **do**
    $(I_1, J_1) = (I, J)$
    if $n < |J|$ **then**
      Remove $|J| - n$ smallest columns from (I_1, J_1)
    else
      Add $n - |J|$ smallest columns to (I_1, J_1)
    end
    $\tau = (1 - \gamma)\frac{n}{|J|}$
    $\overline{I} = \sqrt{\tau}|I|$ smallest rows in I
    $\overline{J} = \sqrt{\tau}|J|$ smallest columns in J
    $(I_1, J_1) =$ **replaceNodes2**$(I_1, J_1, A, \overline{I}, \overline{J})$
    if $Err_{\mathbb{M}}(I_1, J_1) < \delta$ **then**
      Add smallest rows to (I_1, J_1) s.t. adding them does not violate the overlap limit until $\mathrm{Err}_{\mathbb{M}}(I_1, J_1) \geq \delta$ or no such rows exist.
      Remove the last added row from (I_1, J_1) if $\mathrm{Err}_{\mathbb{M}}(I_1, J_1) > \delta$.
    else
      Delete largest rows from (I_1, J_1) s.t. deleting them does not violate the overlap limit until $\mathrm{Err}_{\mathbb{M}}(I_1, J_1) \leq \delta$ or no such rows exist.
    end
    if $|I^*||J^*| > |I_1||J_1|$ *and* $Err_{\mathbb{M}}(I_1, J_1) \leq \delta$ **then**
      $(I^*, J^*) = (I_1, J_1)$
    end
  end
  $\boldsymbol{B} = \boldsymbol{B} \cup \{(I^*, J^*)\}$
end
return $\boldsymbol{B}$



(a) Identify the first core node set and search around this core set

(b) Confine search area to the largest disjoint submatrix

(c) Identify a new core node set

(d) Search around this core set

Figure 4.4: Main steps of the **FLEXBIC** algorithm.

FLEXBIC starts by finding the largest submatrix without no nodes in common with any of previous optimal bicluster in $\boldsymbol{B}$ by calling procedure *restrictRegion* presented in Algorithm 17. After starting from the full matrix, this procedure identifies the bicluster in $\boldsymbol{B}$ having the largest overlap. If the ratio of common rows is greater than the ratio of common columns, the common columns are removed from the restricted matrix. Otherwise, the common rows are removed. The whole process is repeated until there is no overlap. Through incremental computation, the complexity of this procedure is $O(M+N)$ where M and N are the number of rows and columns, respectively.

To find a locally optimal bicluster in the restricted area, we use **BICRELS** as it is effective and efficient for this goal. The characteristic of

Algorithm 17: restrictRegion

Input: data matrix A with rows X and columns Y, bicluster set $\boldsymbol{B}$

$(\hat{I}, \hat{J}) = (X, Y)$

repeat

  $(I_2, J_2) = \arg\max_{(I_1,J_1)\in\boldsymbol{B}} |(I_1, J_1) \cap (\hat{I}, \hat{J})|$

  if $\frac{|I_2\cap\hat{I}|}{|\hat{I}|} > \frac{|J_2\cap\hat{J}|}{|\hat{J}|}$ **then**

    $\hat{J} = \hat{J} \setminus (J_2 \cap \hat{J})$

  else

    $\hat{I} = \hat{I} \setminus (I_2 \cap \hat{I})$

  end

until $I_2 = \emptyset$ *or* $J_2 = \emptyset$;

return $(\hat{I}, \hat{J})$

the bicluster (I, J) returned by **BICRELS** can fall into two cases. First, only a subset of columns in the current biclusters are core nodes. The number of core columns is then identified by each core column threshold β_c. However, setting precisely β_c is difficult, therefore we assume that the number of core columns varies from $(1-\alpha_1)|J|$ to $|J|$. The second situation is where all columns of the current bicluster are core nodes and included in a larger core set. In this case, we assume that the number of core columns can be from $|J|$ to $(1+\alpha_2)|J|$. In other words, the algorithm tries reducing and expanding the column size from $(1-\alpha_1)|J|$ to $(1+\alpha_2)|J|$ where α_1, α_2 are parameters controlling the reduction and expansion size. Large values of (α_1, α_2) mean that the algorithm searches for more potential solutions and consumes more run-time. To guarantee the overlap limit, α_2 must satisfy $\alpha_2 \leq \frac{\gamma}{1-\gamma}$ (where γ is the overlap threshold). To prove this, we observe that the core bicluster has no common cells with other biclusters, and the overlapping cells can only come from adding $\alpha_2|J|$ columns to the current bicluster. When the column size is $(1+\alpha_2)|J|$, the maximum

Algorithm 18: replaceNodes2

Input: a bicluster (I, J), tabu row and column set $\overline{I}, \overline{J}$

$\hat{I} = X \setminus I, \quad \hat{J} = Y \setminus J$

repeat

  // Replace columns

  repeat

    $maxJ = \arg\max_{j \in J \setminus \overline{J}} \text{colErr}_{\mathbb{M}}^{IJ}(j)$

    $minJ = \arg\min_{j \in \hat{J}} \text{colErr}_{\mathbb{M}}^{IJ}(j)$

    $J' = J \cup \{minJ\} \setminus \{maxJ\}$

    if $Err_{\mathbb{M}}(I, J') < Err_{\mathbb{M}}(I, J)$ **then**

      $J = J'$

      $\hat{J} = \hat{J} \setminus \{minJ\}$

    end

  until *J is not modified or* $\hat{J} = \emptyset$;

  // Replace rows

  repeat

    $maxI = \arg\max_{i \in I \setminus \overline{I}} \text{rowErr}_{\mathbb{M}}^{IJ}(i)$

    $minI = \arg\min_{i \in \hat{I}} \text{rowErr}_{\mathbb{M}}^{IJ}(i)$

    $I' = I \cup \{minI\} \setminus \{maxI\}$

    if $Err_{\mathbb{M}}(I', J) < Err_{\mathbb{M}}(I, J)$ **then**

      $I = I'$

      $\hat{I} = \hat{I} \setminus \{minI\}$

    end

  until *I is not modified or* $\hat{I} = \emptyset$;

until *I, J are not modified*;

return (I, J)

overlap between the current bicluster and the other biclusters in $\boldsymbol{B}$ is:

$$\frac{\alpha_2|J||I|}{(1+\alpha_2)|J||I|} = \frac{\alpha_2}{1+\alpha_2} \leq \gamma$$
$$\Leftrightarrow \ \alpha_2 \leq \frac{\gamma}{1-\gamma}.$$

To reduce (or increase) the columns size, our algorithm removes (or adds) the columns with largest residue from (or to) the core bicluster. It then calls **replaceNodes2** in Algorithm 18 to reduce residue. This procedure is an extension of procedure **replaceNodes** in **BICRELS** for controlling the overlap limit. Specifically, the **replaceNodes2** procedure replaces nodes which are not in $\sqrt{\tau}|I|$ rows and $\sqrt{\tau}|J|$ columns with smallest residue where $\tau = (1-\gamma)\frac{n}{|J|} \leq 1$ and n is the number of columns in the new bicluster. As the submatrix formed by these rows and columns is included in the core bicluster, it has no overlap with other biclusters. This guarantees that the overlap is always below the threshold γ. This is because, when the column size is n and we keep $\sqrt{\tau}|I|$ and $\sqrt{\tau}|J|$ smallest rows and columns, the maximum overlap is:

$$1 - \frac{\sqrt{\tau}|I|\sqrt{\tau}|J|}{|I|n} = 1 - \frac{\tau|J|}{n} \leq \gamma$$
$$\Leftrightarrow \tau \geq (1-\gamma)\frac{n}{|J|}.$$

After performing **replaceNodes2**, if the residue is smaller than the threshold δ, our algorithm traverses the rows in the ascending order of residue and adds the ones permitted by the overlap limit. The process is repeated until the residue exceeds the threshold or no such rows exist. If necessary, the last added row is removed to guarantee the residue constraint. Similarly, if the bicluster residue is larger than the threshold δ, the algorithm keeps deleting the largest rows without violating the overlap constraint until the residue is below the threshold or no such rows exist. Finally, if the current

bicluster is valid and better, the best bicluster is updated and added to the set $\boldsymbol{B}$ before moving to the next iteration.

The complexity of **BICRELS** is $O((M+N)MN)$ where M and N are the number of rows and columns. Removing or adding a column (line 11 to 14) requires the computation of all column residues $\text{colErr}_{\mathbb{M}}^{IJ}$ with the complexity of $O(MN)$. Thus, the complexity of this step is $O(MN^2)$. As only one column is modified each time, the incremental update can be used to reduce the complexity from $O(MN^2)$ to $O(N^2)$. The complexity of **replaceNodes2** is $O((M+N)MN)$ because the maximum number of rows and columns which can be replaced is $M+N$. For each replacement step, we need to compute all row or column residues with the complexity of $O(MN)$. Besides, from line 20 to 23, searching for the smallest or largest rows requires the computation of all row residues $\text{rowErr}_{\mathbb{M}}^{IJ}$ with the complexity of $O(MN)$. In the worst case, all rows can be added and therefore complexity of this part is $O(M^2N)$. However, in each step, only one row is added or removed, thus their residues can be updated incrementally to reduce the complexity from $O(M^2N)$ to $O(M^2)$. In total, the complexity of our algorithm is $O((M+N)MN)$.

4.4 Discovering Disjoint Biclusters in Euclidean Space

Most subspace clustering algorithms using p-norm distances are often developed to find a set of disjoint or all valid biclusters. Therefore, in this section, we illustrate the application of our algorithm for searching disjoint biclusters where the dissimilarity between items is measure by the Euclidean distance.

4.4.1 State-of-the-art Benchmark Algorithms

A representative set of the most competitive subspace clustering algorithms have been selected as benchmark for our experiments. Let's summarize their principles [68].

In the grid-based approaches, each subspace cluster is defined as a set of connected grid cells where each cell contains a number of objects greater than a threshold. A cell is specified by intervals I_i per dimension i. If the dimension i belongs to the subspace of that cluster, then the cell interval I_i is strictly a part of the domain $[L_i, U_i]$ of dimension i where L_i, U_i are the lower and upper bound that dimension. On other non-relevant dimensions, the cell intervals are the full domains. The algorithms in this scheme can generate all possible clusters or only high-quality ones, the quality being measured by an external function μ. **MineClus** [98] picks at each iteration a random point p from the dataset and identifies the cluster centered at p. To this end, **MineClus** transforms the subspace clustering problem into the problem of mining frequent itemsets in transactional databases. Each point q is replaced with an itemset as follows. If the difference between the value of q and p in dimension i is less than a threshold w, then dimension i is included into the corresponding itemset of q. Then, **MineClus** runs a frequent itemset mining algorithm to find the best dimensions and objects for the cluster centered at p. The process is repeated for a number of random medoids and the cluster with the highest quality is selected. After a cluster is discovered, the objects in that clusters are removed and the process is applied again to the remaining objects to identify the next cluster. Finally, clusters are merged if users want to have at most K clusters.

Instead of using grids, other researchers extend the notation of density-connectivity of DBSCAN [34] for subspace clustering: clusters are specified as dense regions separated from sparse regions where the distance between

objects is computed only on the relevant dimensions of those clusters. One of the fist density-based subspace clustering algorithms is **SUBCLU** [55], a bottom-up, greedy algorithm. **SUBCLU** first searches for all clusters in 1-D subspaces by running DBSCAN on these subspaces. Then, based on the k-dimensional clusters and subspaces obtained, it builds a list of $(k+1)$-dimensional subspace candidates and runs DBSCAN on this candidate list to generate the $(k+1)$-dimensional clusters. The whole procedure is repeated until no clusters are found. Because of the monotonicity of density-connected sets, the algorithm can quickly eliminate all subspaces that cannot contain clusters. However, as there is no mechanism to control the redundancy, the number of clusters generated by **SUBCLU** is often large and the clusters can overlap significantly.

Unlike the two previous schemes searching for individual clusters, the projected-subspace approaches aim at discovering a whole set of clusters at once. **PROCLUS**, an algorithms in this approach [1], starts by picking a set M of K random points from a superset of βK farthest points to form a set of K medoids where $\beta > 1$ controls the superset size. Then, it identifies the best dimensions for all medoids and assigns each point to the nearest medoid to form clusters. The distance between each point and a medoid is computed under the dimensions assigned to that medoid. The quality of a medoid set is considered as the clustering quality. At the end of an iteration, the bad medoids whose clusters contain fewest points are replaced by random points in the superset of medoid. The whole procedure is repeated for many iterations until the medoid set cannot be improved any more. Although **PROCLUS** is robust, it requires users to specify the average number of dimensions for biclusters and this parameter is often unknown in practice.

4.4.2 Model and Error Function

For the experiments of this section, the distance between objects in I under some subspace J is defined by their Euclidean distance. The model $\mathbb{M}$ is the following one:

$$\hat{a}_{ij} = \frac{1}{|I|} \sum_{i \in I} a_{ij} \tag{4.8}$$

and the squared cell error:

$$\text{cellErr}_{\mathbb{M}}(a_{ij}, \hat{a}_{ij}) = (a_{ij} - \hat{a}_{ij})^2. \tag{4.9}$$

Under this model, the error function $\text{Err}_{\mathbb{M}}(I, J)$ is defined as:

$$\text{Err}_{\mathbb{M}}(I, J) = \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} (a_{ij} - \hat{a}_{ij})^2 \tag{4.10}$$

$$= \frac{1}{|I||J|} \sum_{i \in I} ||\boldsymbol{a}_i^J - \overline{\boldsymbol{a}}_I^J||^2 \tag{4.11}$$

where $\boldsymbol{a}_i^J = \{a_{ij}\}_{j \in J}$ is the vector representing the object i under the subspace J, and $\overline{\boldsymbol{a}}_I^J = \frac{1}{|I|} \boldsymbol{a}_i^J$ is the centroid vector of the subspace cluster (I, J). As can be seen, $\text{Err}_{\mathbb{M}}(I, J)$ in Equation (4.11) is actually the vector quantization error used in the K-means algorithm [62]. In other words, a valid δ-bicluster (I, J) under this model is a set $I \subset X$ of objects under a subspace $J \subset Y$ such that the normalized vector quantization error of this bicluster is less than a threshold δ.

4.4.3 Experiments

In this section, we compare our algorithms **BICRELS** with the other state-of-the-art algorithms: **PROCLUS** [1], **MineClus** [98], and **SUBCLU** [55].

Table 4.2: UCI dataset characteristics

Dataset	Number of Objects	Average number of Dimensions	Number of Clusters
Iris	150	4	3
Wine	178	13	3
Glass	214	9	6
IrisWine	328	9	6
IrisGlass	364	7	9
WineGlass	392	11	9

Experimental Setup: We embed UCI datasets *Iris*, *Wine*, *Glass* and their combinations into high-dimensional space and run all subspace clustering algorithms on these datasets and check whether the *ground truth* biclusters can be recovered.

In detail, all original UCI data sets are normalized (subtracting the mean and dividing by the standard deviation) before embedding them into a high-dimensional space. For a dataset $A = (X, Y)$, we append $(\varphi - 1)|Y|$ dimensions to each object to form a new dataset with $\varphi|Y|$ dimensions. The value of each object under the new dimensions are random values from 0 to 100. The order of objects in the dataset is randomized. Similarly, we embed pairs of datasets into high-dimensional space to test whether the algorithms can discover biclusters with different number of dimensions. For a pair of two datasets $A_1 = (X_1, Y_1)$ and $A_2 = (X_2, Y_2)$, we embed them into a space of $\varphi \max(|Y_1|, |Y_2|)$ dimensions following the previous procedure.

The number of biclusters on each dataset is set as the number of clusters in the original one. To obtain K biclusters, **BICRELS** is run for K times. Following the Cheng and Church suggestion [19], we replace the values of the cells in the subspace cluster found the $(k-1)$-th run by random values from 0 to 100 before the k-th run. The parameters

$numberOfRestarts$ and the residue threshold δ of **BICRELS** are set to 10 and 1, respectively. Parameters K_r and K_c of **BICRELS** are set to $\min(|Y|/10, 100)$ and $\min(|X|/10, 100)$ where $|Y|$, $|X|$ are the number of rows and columns. As for **SUBCLU**, its parameter $minPoints$ and radius ϵ are set to 10 and 1, respectively. Because **SUBCLU** generates a set $C^{\mathbf{SUBCLU}}$ biclusters which can overlap significantly, we select only K biclusters from this set to compare with the other algorithms as follows. For each true bicluster $C_t = (I_t, J_t)$ of a dataset DB, we choose the bicluster $C_p = (I_p, J_p) \in C^{\mathbf{SUBCLU}}$ having the highest overlap with C_t, i.e.:

$$C_p = \underset{C_q=(I_q,J_q)\in C^{\mathbf{SUBCLU}}}{\arg\max} |I_q \cap I_t||J_q \cap J_t|. \tag{4.12}$$

This reduction procedure helps us to check whether the true biclusters are included in the bicluster set $C^{\mathbf{SUBCLU}}$ generated by **SUBCLU**. The parameters of **MineClus** are set as follows: $\alpha = 0.5/K, \beta = 0.1, w = 1, k = K$ where K is the true number of clusters in each dataset. Finally, the average number of dimensions and the number of biclusters in **PROCLUS** are set to the ground truth values on each dataset as in Table 4.2. **BICRELS** is implemented in Matlab while three other algorithms **PROCLUS**, **MineClus** and **SUBCLU** are implemented in Java under the framework described in [68].

The performance of all algorithms is evaluated as the matching between the clusterings produced by these algorithms and the ground truth clusterings, measured by the Adjusted Rand Index (ARI) [53]. ARI computes the matching between two clusterings on full dimensional space as the ratio between the number of object pairs in the same cluster or in different clusters in both clusterings and the total number of pairs. The maximum value of ARI is one when two clusterings are identical and is around zero when they are very different. ARI is popular as it guarantees that the expected matching of two random clusterings is zero. To measure the matching of

two clusterings where their clusters are located in subspaces, we consider each cell in the data matrix as an object, and all cells of a subspace cluster are assigned to the same cluster. The cells which do not belong to any bicluster are grouped to a noise cluster.

Experimental Result: Fig.4.5 shows the adjusted rand index of all subspace clustering algorithms on six datasets. **BICRELS** outperforms the other algorithms on most datasets. On two datasets *Iris* and *Wine*, the performance of **BICRELS** and **PROCLUS** are approximately the same. However, on the *Glass* datasets and on the combined datasets where the number of dimensions in biclusters can be different, **BICRELS** produces much better results than **PROCLUS**. Besides, **BICRELS** performance is more stable when changing the dataset dimensions.

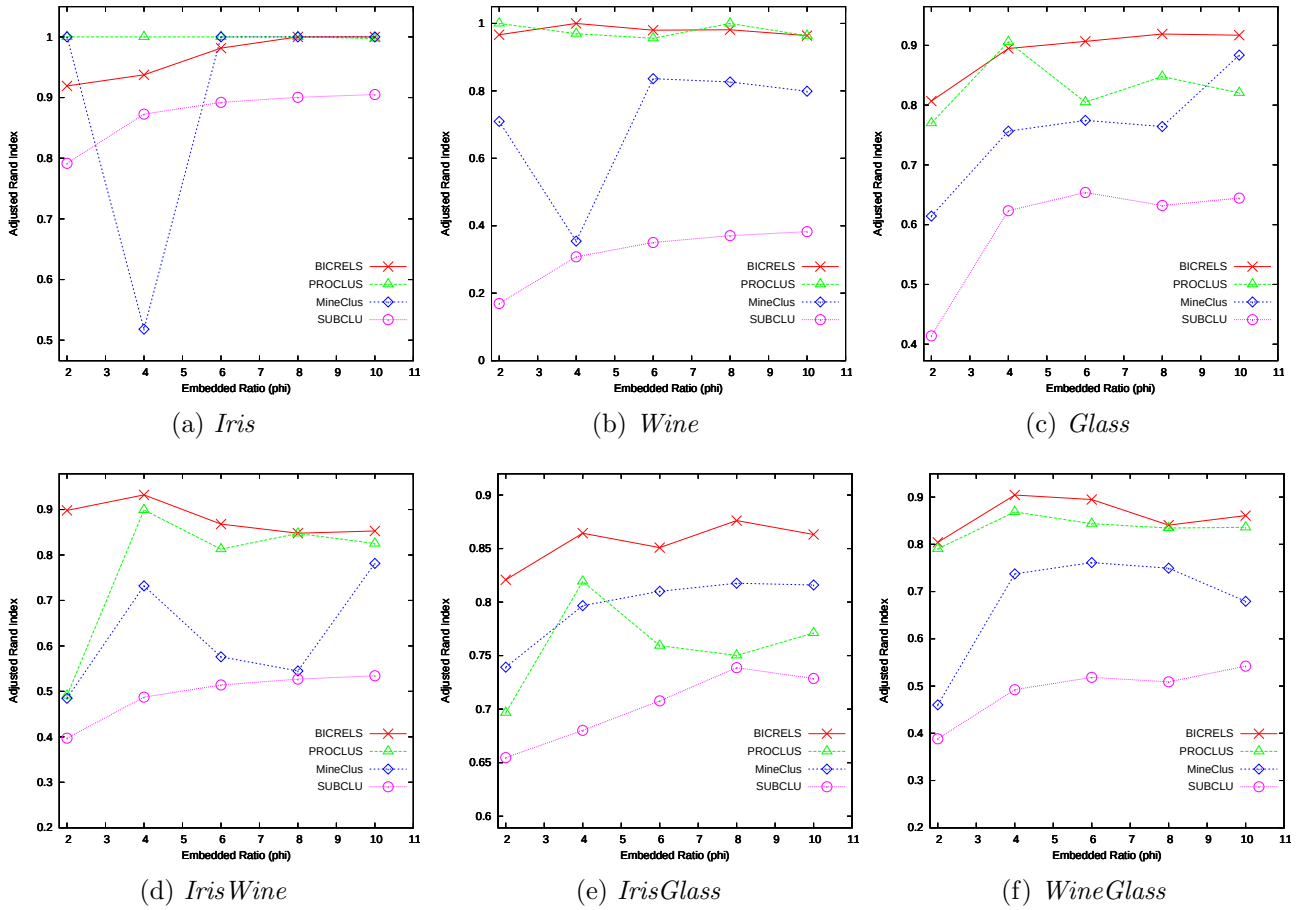
4.5 Discovering Non-Redundant Overlapping Biclusters on Gene Expression Data

In this section, we illustrate the application of our algorithm on gene expression data to generate K overlapping biclusters where the maximum overlap between them is below a predefined threshold.

4.5.1 Related Work on Biclustering of Gene Expression Data

The algorithms proposed for solving the biclustering problem can be classified into different groups [65]:

- Iterative row and column clustering combination: applying the standard clustering methods on rows and columns of the data matrix and then combining the row and column clusters to form biclusters [39].
- Divide and conquer: breaking the problem into smaller problems, solving them recursively, and combining the solutions of sub-problems to



(a) *Iris* (b) *Wine* (c) *Glass*

(d) *IrisWine* (e) *IrisGlass* (f) *WineGlass*

Figure 4.5: Performance comparison of four algorithms on UCI datasets embedded in high-dimensional space. The x-axis corresponds to the ratio φ between the number of dimensions in the high-dimensional space and that of the original space of each dataset.

form the solution for the original problem [48].

- Greedy iterative search: removing rows or columns to reduce the bicluster residue below the threshold and adding rows or columns to increase the bicluster volume while the constraint on residue is still satisfied [19].

- Exhaustive bicluster enumeration: enumerating all possible biclusters to identify the best ones in exponential time [83].

- Distribution parameter identification: assuming the data is generated from a model and trying to fit parameters of that model by minimizing a certain criterion [61].

However, the biclustering algorithms can optimize different "coherence" criteria [65]. Therefore, in the experiments of Section 4.5, we only compare our algorithm, **FLEXBIC**, with those based on the *additive model*.

One of the first additive models is proposed by Cheng and Church [19]. They define a bicluster (I, J) (with I and J are the row and column sets) as a δ-bicluster if and only if its mean squared residue $MSR(I, J)$ below a threshold δ with:

$$MSR(I, J) = \frac{1}{|I|J|} \sum_{i \in I,\ j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2$$

where a_{iJ}, a_{Ij} and a_{IJ} are the *ith* row mean, the *jth* column mean and the bicluster mean. Similarly, the residue of a row or a column is computed as the average residue of all cells in that row or column. Their algorithm to search for the largest δ-bicluster starts from the initial bicluster containing all rows and columns and iteratively deletes a set of nodes (rows or columns) with large residues until the mean squared residue is below the threshold. A set of outside nodes is then added to the bicluster to increase its volume while obeying the residue constraint. This algorithm

is deterministic and very fast, as a set of nodes can be deleted or added at the same time. Its complexity is $O((M+N)MN)$ where M and N are the number of genes and conditions. However, modifying a set of nodes simultaneously can also leads to non-optimal solutions. Besides, in each run, the algorithm can only generate one largest bicluster. To generate a set of disjoint biclusters, the cell values in the previously discovered biclusters are replaced by random values and the algorithm is rerun on the new data. We denote this algorithm as **ChengChurch**.

To overcome its limitation, Yang et al. [97] propose a probabilistic method named **FLOC** (Flexible Overlapped Clusters) which can discover a set of K biclusters in a run. The algorithm starts from a set of random initial biclusters. They are formed by selecting randomly a subset of rows and a subset of columns, such that the bicluster residue is below a threshold. Then **FLOC** iteratively performs the best action for each row and column to improve the bicluster quality. The actions are deleting or adding a row or a column to one of K biclusters. The best action is the one that gives the highest improvement in a gain function which is the sum of the reduction ratio in mean squared residue and the increase ratio in volume. As two objectives (volume and residue) are considered at the same time, **FLOC** can return biclusters with very small volume while their residues are much lower than the threshold. Besides, **FLOC** is very sensitive to the initial biclusters and its complexity is $O((M+N)^2 \times K \times p)$ where M, and N are the number of genes and conditions, K is the number of biclusters, and p is the number of iterations the algorithm runs until convergence.

Lazzeroni and Owen introduce the plaid model to analyze the underlying structure of a gene expression matrix [61] allowing overlap between biclusters. The data matrix is described as the sum of different layers where each layer corresponds to a bicluster. In detail, a gene expression

level a_{ij} of the *ith* gene and the *jth* condition is modeled as:

$$a_{ij} = \sum_{k=0}^{K} (\mu_k + \alpha_{ik} + \beta_{jk}) \rho_{ik} \kappa_{jk} \tag{4.13}$$

where μ_k, α_{ik} and β_{jk} are the mean, the *ith* gene and the *jth* condition effects in the *kth* layer. ρ_{ik} and κ_{jk} are binary variables describing the membership of the *ith* gene and the *jth* condition. Layer zero ($k = 0$) describes the background. It is fitted before adding the next layers one by one until the predefined number of biclusters is reached or no more significant biclusters are found. While fitting the model parameters, the authors relax the membership variables ρ_{ik} and κ_{jk} as real variables shifted towards binary solutions during the optimizing process. Turner et al. find that "the gradual shifting of estimates introduces a discontinuity which may prevent the algorithm converging to a superior solution to the final result" [90]. Therefore, they propose an optimization method using binary bicluster membership to preserve continuity and speed up convergence. We refer to this improved version as **ImpPlaid**.

4.5.2 Model and Error Function

To discover biclusters in a data matrix of co-expression patterns, we use the additive model $\mathbb{M}$ proposed by Cheng and Church [19]:

$$\hat{a}_{ij} = a_{iJ} + a_{Ij} - a_{IJ}, \tag{4.14}$$

where

$$a_{iJ} = \frac{1}{|J|} \sum_{j \in J} a_{ij},$$

$$a_{Ij} = \frac{1}{|I|} \sum_{i \in I} a_{ij},$$

$$a_{IJ} = \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} a_{ij} = \frac{1}{|I|} \sum_{i \in I} a_{iJ} = \frac{1}{|J|} \sum_{j \in J} a_{Ij}.$$

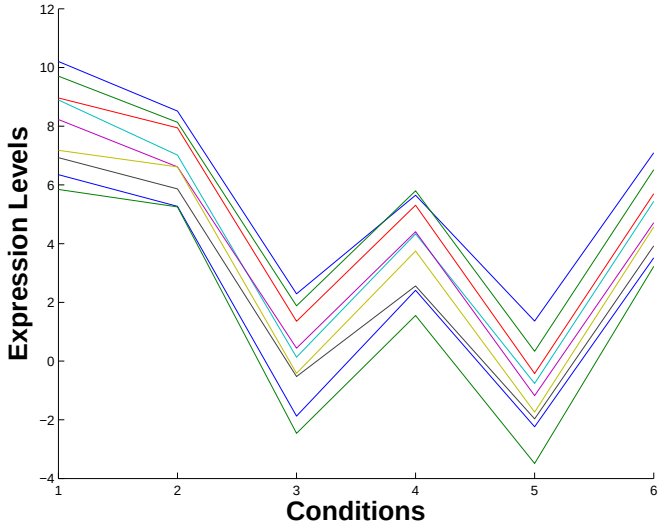


Figure 4.6: An example of a bicluster with 9 genes and 6 conditions.

The biological motivation for the model is that, in a gene regulatory network, the gene expression level is proportional to a sum of a term characterizing the gene plus a term characterizing the experimental condition which is activating the specific network. Let's note that, if logarithms of the original measures are taken, the model is multiplicative in the original measures. Fig.4.6 shows an example of a bicluster with 9 genes and 6 conditions.

The error measure is the squared cell error:

$$\text{cellErr}_{\mathbb{M}}(a_{ij}, \hat{a}_{ij}) = (a_{ij} - \hat{a}_{ij})^2. \tag{4.15}$$

The error function $\text{Err}_{\mathbb{M}}(I, J)$ is defined as:

$$\text{Err}_{\mathbb{M}}(I, J) = \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} (a_{ij} - \hat{a}_{ij})^2 \tag{4.16}$$

$$= \frac{1}{|I||J|} \sum_{i \in I} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2. \tag{4.17}$$

4.5.3 Experiments

In this section, we compare the performance of all algorithms for finding the overlapping biclusters, the largest biclusters and the coverage of K biclusters.

Experimental Setup

In the following experiments, we set the number of biclusters generated by each algorithm to $K = 10$. Parameter α_1, α_2 and γ of **FLEXBIC** are set to 0.5. Parameter α of **ChengChurch** is set to its default value ($\alpha = 1.2$). The number of initial rows and columns in **FLOC** are set to 9 and 2, respectively as in [89]. The number of biclusters produced by **FLOC** in each run is set to 10. **FLOC** is run for 10 times to eliminate the randomness effect. In each run, **BICRELS** is restarted for 10 times ($numberOfRestarts = 10$). As **BICRELS** can only return the largest bicluster in each run, to generate $K = 10$ biclusters, we also apply the random masking process of **ChengChurch**. In the initialization phase of **BICRELS**, the K-means algorithm uses *cosine similarity* to measure the difference between two rows or columns. The reason is that two very similar patterns with large difference in feature magnitude should still have a small distance. Besides, the parameters K_r and K_c of **BICRELS** are set to $\min(|Y|/10, 100)$ and $\min(|X|/10, 100)$ where $|Y|$, $|X|$ are the number of rows and columns, respectively. Because **ImpPlaid** is non-deterministic and the number of biclusters is identified automatically, we run **ImpPlaid** 10 times on each dataset with different random seeds, and select the result from the run which produces at least two biclusters and has the highest coverage. The parameters of **ImpPlaid** are set to the default values in the source code of the *biclust* R package [1].

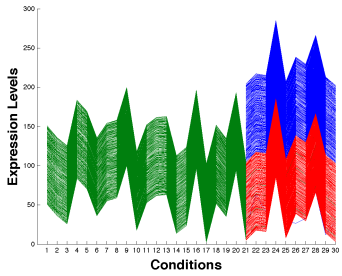
[1] `http://cran.r-project.org/web/packages/biclust/`

In the experiments, we use two real datasets *Yeast* [84] and *Lymphoma* [2]. The *Yeast* dataset consists of 2884 genes and 17 conditions. The *Lymphoma* dataset has 4026 genes and 96 conditions. These datasets are preprocessed by Cheng and Church[2]. The missing values are processed as in [19]. The residue threshold of all algorithms on the *Yeast* and *Lymphoma* dataset are set to 300 and 1200 as in [19, 89].

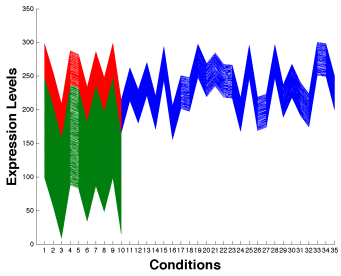
Identifying the overlapping biclusters

Fig.4.7 shows the biclusters found by four algorithms on two synthetic datasets. We denote the first and second bicluster with the green and blue color and the overlap part with the red color. In other to plot the overlap part, we arrange the columns such that these overlapping columns are at the end of the first bicluster and at the beginning of the second bicluster. We plot the overlap part after plotting two biclusters, therefore the overlap part is always visual if any. For each algorithm, we only plot two biclusters with maximum overlap. When there is no overlap between biclusters, we plot the two largest biclusters of that algorithm. It can be seen that **FLEXBIC** has discovered different overlapping biclusters through the expansion steps. Besides, **FLOC** can only produce disjoint biclusters with a very small number of columns. Although **ImpPlaid** can also discover reasonably large biclusters on *Dataset 1*, these biclusters have no overlap. This comes from the fact that after subtracting the first layer corresponding to the first bicluster, the values of all cells in the overlap regions are also removed, i.e., in the remaining residue matrix, these values are almost the same as the background. Therefore, when adding the second layer, **ImpPlaid** cannot include the rows and columns in such overlap region. Similarly, on *Dataset 2*, **ImpPlaid** only produces a pair of biclusters with very few overlapping nodes. In contrast, **ChengChurch** produces very

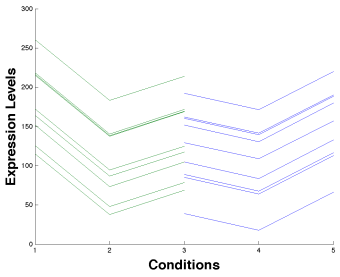
[2]`http://arep.med.harvard.edu/biclustering`



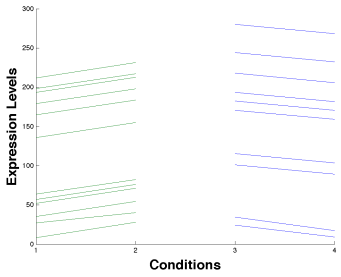
(a) *Dataset 1*, **FLEXBIC** biclusters: 6000(200 × 30), 3960(396 × 10)



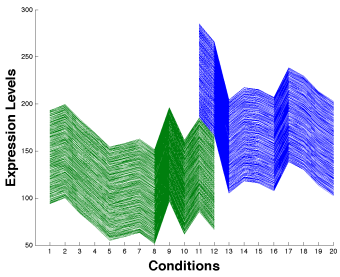
(b) *Dataset 2*, **FLEXBIC** biclusters: 4000(400 × 10), 3500(100 × 35)



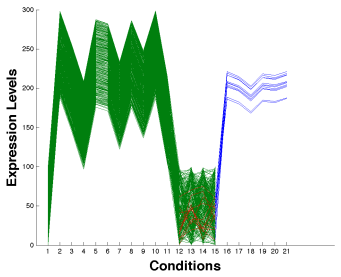
(c) *Dataset 1*, **FLOC** biclusters: 27(9 × 3), 27(9 × 3)



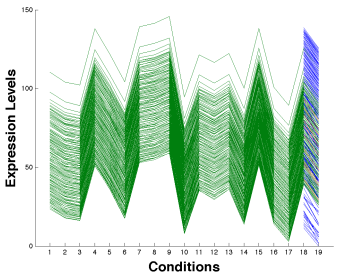
(d) *Dataset 2*, **FLOC** biclusters: 24(12 × 2), 20(10 × 2)



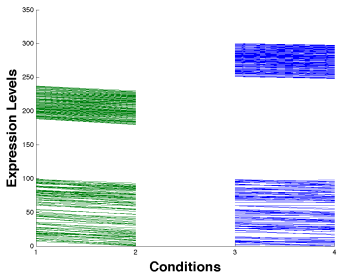
(e) *Dataset 1*, **ImpPlaid** biclusters: 2400(200 × 12), 2000(200 × 10)



(f) *Dataset 2*, **ImpPlaid** biclusters: 3180(212 × 15), 120(12 × 10)

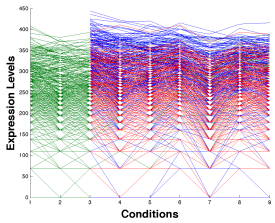


(g) *Dataset 1*, **ChengChurch** biclusters: 2470(130 × 19), 286(143 × 2)

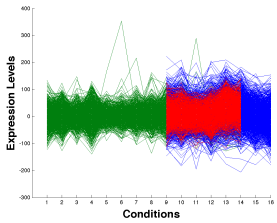


(h) *Dataset 2*, **ChengChurch** biclusters: 406(203 × 2), 398(199 × 2)

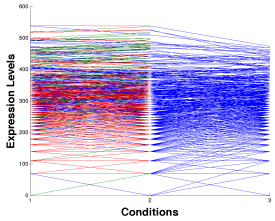
Figure 4.7: Biclusters found by four algorithms on two synthetic datasets. Each bicluster's information is presented in the format $Volume(numberOfRows \times numberOfColumns)$.



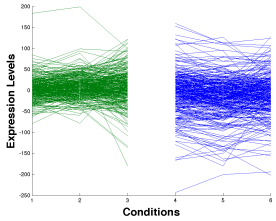
(a) *Yeast*, **FLEXBIC** biclusters: 4689(521 × 9), 5880(840 × 7)



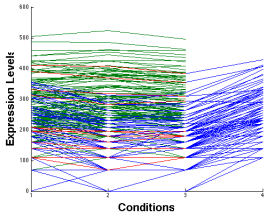
(b) *Lymphoma*, **FLEXBIC** biclusters: 13664(976 × 14), 10832(1354 × 8)



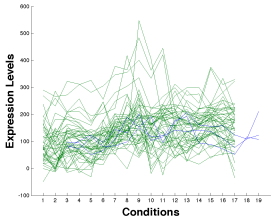
(c) *Yeast*, **FLOC** biclusters: 1500(750 × 2), 2709(903 × 3)



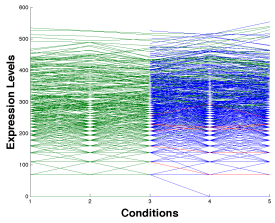
(d) *Lymphoma*, **FLOC** biclusters: 948(316 × 3), 885(295 × 3)



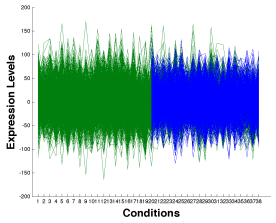
(e) *Yeast*, **ImpPlaid** biclusters: 555(185 × 3), 512(128 × 4)



(f) *Lymphoma*, **ImpPlaid** biclusters: 816(48 × 17), 51(3 × 17)



(g) *Yeast*, **ChengChurch** biclusters: 2540(508 × 5), 2058(686 × 3)



(h) *Lymphoma*, **ChengChurch** biclusters: 39026(1027 × 38), 10697(563 × 19)

Figure 4.8: Biclusters found by four algorithms on two real datasets. Each bicluster's information is presented in the format $Volume(numberOfRows \times numberOfColumns)$.

poor results in these two datasets with quite small biclusters. We also run **BICRELS** with the random masking process of the **ChengChurch** algorithm and observe that **BICRELS** can only discover the largest bicluster but cannot find the second largest one with the overlap part. For the sake of space, we do not plot the **BICRELS** biclusters.

Fig.4.8 shows the biclusters found by four algorithms on two real datasets *Yeast* and *Lymphoma*. The figures show that **FLEXBIC** can find much more interesting non-redundant overlapping biclusters compared to those of the other algorithms. **FLOC** can only discover the overlapping biclusters on the *Yeast* dataset. However, the small bicluster is mostly included in the larger bicluster. In other words, the small bicluster is redundant when we already had the larger bicluster. As for **ImpPlaid**, on the *Yeast* dataset, **ImpPlaid** can produce a pair of small biclusters with a few overlapping rows and columns. However, it cannot find the overlapping ones on the *Lymphoma* dataset. **ChengChurch** can return biclusters with small overlap on the *Yeast* dataset and no overlap on the *Lymphoma* dataset. It is also noticeable that although the biclusters of **ChengChurch** share some common columns, **ChengChurch** cannot find at the same time common rows due to certain effects of the random masking process. Similarly, **BICRELS** also produces no overlapping biclusters on these datasets. In summary, on four datasets, **FLEXBIC** can produce non-redundant overlapping biclusters whereas the other algorithms either return disjoint or redundant biclusters.

Table 4.3 shows the run-time comparison between five algorithms on four datasets. **FLEXBIC** is slower than **BICRELS**, **ChengChurch**. This comes from the fact that **FLEXBIC** aims at solving the overlapping biclustering problem which is much more complex the disjoint biclustering problem considered by **ChengChurch** and **BICRELS**. However, comparing with two other algorithms allowing the overlapping, **FLEXBIC** is

Dataset / Algorithm	*Dataset 1*	*Dataset 2*	*Yeast*	*Lymphoma*
FLEXBIC	22.88	25.19	112.88	718.76
FLOC	2833.59	3462.77	16428.79	5891.46
BICRELS	13.01	13.11	33.04	108.34
ChengChurch	7.44	9.31	0.40	1.56
ImpPlaid	36.89	41.96	44.49	62.93

Table 4.3: Run-time Comparison (in seconds) of five algorithms on four datasets.

much faster than **FLOC** and comparable to **ImpPlaid**.

Analyzing the overlapping biclusters

To test whether the overlapping genes actually belong to meaningful gene ontology terms, we perform gene ontology analysis on the two first overlapping biclusters discovered by **FLEXBIC** (in Fig.4.8a) on the *Yeast* dataset. We use the *GO::Term Finder* software [3] [16] to find significant gene clusters on the gene sets of two biclusters. The statistical significance for functional category enrichment called *p-value* is measured by using a cumulative hypergeometric distribution to compute the chance probability of observing the number of genes from a particular gene ontology category within each cluster. In detail, the chance probability of obtaining at least k genes from a specific functional category within a cluster of size n is:

$$p = 1 - \sum_{i=0}^{k-1} \frac{\binom{f}{i}\binom{g-f}{n-i}}{\binom{g}{n}} \tag{4.18}$$

where f is the total number of genes within that functional category and g is the total number of genes within the genome [84]. Often, a test with *p-value* ≤ 0.05 is considered as having statistical significance.

We search for pairs of gene clusters with largest overlap where one cluster

[3] `http://go.princeton.edu/cgi-bin/GOTermFinder`

in the pair belonging to the first bicluster and the other in the second bicluster. We also remove pairs with identical functions or having parent-child relation. Table 4.4 shows 10 such pairs of gene clusters. The common genes can be classified in the biological process having the properties of both clusters or in another different process. For example, searching for gene ontology terms of the common genes in the first pair (the *cellular process* and the *metabolic process*), we find a group of 167/169 overlapping genes belonging to the *cellular metabolic process* with *p-value*$= 3.64 \times 10^{-37}$, i.e., the chance probability of obtaining at least 167 genes from the *cellular metabolic process* within a cluster of size 169 is 3.64×10^{-37}. Similarly, 149/150 common genes of the second pair (the *single organism process* and the *cellular process*) are clustered in the *single-organism cellular process* with *p-value* $= 2.84 \times 10^{-50}$. Testing on the common genes of the other pairs, we also see that most common genes are grouped into significant gene ontology terms.

Comparison on the largest bicluster

In this section, we compare the largest biclusters produced by five algorithms. In some cases, some algorithms like **FLOC** can be stuck in local minima and return biclusters with very small volumes whereas their residues are much lower than the threshold. Therefore, for a clearer comparison, we set different residue thresholds in our algorithm to produce biclusters with similar residues as in those of the other algorithms. The overlap constraint is not necessary in comparison as the largest bicluster with residue below a threshold is found with no overlap constraint, i.e., when the given bicluster set is empty ($\boldsymbol{B} = \emptyset$). Besides, we cannot set the overlap constraint for the other algorithms either.

Table 4.5 shows the performance comparison of all algorithms on the largest biclusters. It can be seen that **FLEXBIC** produces much larger

Pair	Gene Clusters in Bicluster 1 Function (Size), p-value	Gene Clusters in Bicluster 2 Function (Size), p-value	Overlapping Cluster Function (Size / Total), p-value
1	cellular process (415), p-value = 4.02×10^{-10}	metabolic process (536), p-value = 2.87×10^{-08}	cellular metabolic process (167/169), p-value = 3.64×10^{-37}
2	single organism process (283), p-value = 4.25×10^{-09}	cellular process (664), p-value = 2.45×10^{-15}	single-organism cellular process (149/150), p-value = 2.84×10^{-50}
3	localization (149), p-value = 1.88×10^{-09}	single organism cellular process (414), p-value = 1.29×10^{-07}	cellular localization (59/75), p-value = 1.87×10^{-38}
4	establishment of localization (139), p-value = 4.14×10^{-09}	cellular localization (135), p-value = 2.05×10^{-06}	establishment of localization in cell (51/55), p-value = 6.08×10^{-44}
5	transport (135), p-value = 7.77×10^{-09}	macromolecule localization (128), p-value = 3.05×10^{-06}	organic substance transport (49/51), p-value = 1.38×10^{-41}
6	cellular protein modification process (82), p-value = 1.56×10^{-07}	cellular metabolic process (517), p-value = 4.80×10^{-09}	cellular protein metabolic process (51/51), p-value = 2.58×10^{-31}
7	protein modification process (82), p-value = 1.56×10^{-07}	macromolecule metabolic process (417), p-value = 5.70×10^{-09}	cellular protein modification process (51/51), p-value = 4.28×10^{-51}
8	phosphorus metabolic process (90), p-value = 2.61×10^{-06}	organic substance metabolic process (512), p-value = 7.05×10^{-09}	organophosphate metabolic process (27/46), p-value = 4.39×10^{-18}
9	cell communication (58), p-value = 2.33×10^{-07}	biological regulation (242), p-value = 2.88×10^{-07}	regulation of cellular process 26/26, p-value = 1.43×10^{-16}
10	organic substance transport (97), p-value = 3.76×10^{-08}	primary metabolic process (498), p-value = 1.43×10^{-08}	macromolecule localization (20/22), p-value = 3.63×10^{-15}

Table 4.4: Gene Clusters of two biclusters discovered by **FLEXBIC** on the *Yeast* dataset. Each line corresponding to a pair of two gene clusters in two biclusters.

biclusters with smaller residues compared to those of the other algorithms. The biclusters of the other algorithms have residues $\text{Err}_{\mathbb{M}}$ much lower than the defined threshold, e.g., on the *Yeast* dataset, with the residue threshold 300, **FLOC** and **ChengChurch** can only find largest solutions with $\text{Err}_{\mathbb{M}}$ of 183.17 and 237.33, respectively. These algorithms may converge prematurely by grouping heterogeneous nodes in the early phases, and have no mechanism to resolve this premature convergence problem later. **BICRELS** and **FLEXBIC** instead can overcome this issue by replacing unsuitable nodes in a bicluster with more suitable external nodes during the **replaceNodes** procedure.

Comparison on the coverage of K biclusters

We compare the coverage percentage of 10 first biclusters generated by the five algorithms in Table 4.6. The residue thresholds of all algorithms are set to the same values (300 and 1200 on the *Yeast* and *Lymphoma* dataset, respectively). When computing the coverage, we only count the overlap part once, so the overlap constraint does not affect the coverage. The coverage of **FLOC** is computed as the mean coverage of 10 runs. It can be seen that **FLEXBIC** covers significantly larger areas compared to those of the other algorithms. For example, on the *Yeast* dataset, **FLEXBIC** explores more than 60%, 15%, 20% and 70% of the data matrix compared to **FLOC**, **BICRELS** and **ChengChurch**, **ImpPlaid** respectively. In other words, not only can **FLEXBIC** produce large and overlapping biclusters, but it can also discover the regions that are omitted by the other algorithms.

Balancing rows and columns

In some cases, users want to increase the number of columns in biclusters to have more evidence on the coherence of rows. In other cases, the number

Algorithm	Max Volume ($\|I\| \times \|J\|$)	$\text{Err}_{\mathbb{M}}$
FLEXBIC ($\delta = 300$)	**16968** (1414×12)	299.99
FLOC ($\delta = 300$)	2709 (903×3)	183.17
FLEXBIC ($\delta = 183.10$)	8910 (1485×6)	182.97
BICRELS ($\delta = 300$)	16577 (1507×11)	299.93
FLEXBIC ($\delta = 299.90$)	16956 (1413×12)	299.83
ChengChurch ($\delta = 300$)	12012 (1001×12)	237.33
FLEXBIC ($\delta = 237.30$)	12490 (1249×10)	237.26
ImpPlaid	564 (141×4)	270.37
FLEXBIC ($\delta = 270.30$)	14772 (1231×12)	270.15

(a) On the *Yeast* dataset

Algorithm	Max Volume ($\|I\| \times \|J\|$)	$\text{Err}_{\mathbb{M}}$
FLEXBIC ($\delta = 1200$)	**45878** (1582×29)	1199.65
FLOC ($\delta = 1200$)	948 (316×3)	460.31
FLEXBIC ($\delta = 460.30$)	13796 (3449×4)	460.03
BICRELS ($\delta = 1200$)	43907 (1909×23)	1199.50
FLEXBIC ($\delta = 1199.50$)	45663 (1473×31)	1199.41
ChengChurch ($\delta = 1200$)	39026 (1027×38)	1101.52
FLEXBIC ($\delta = 1101.50$)	40185 (2115×19)	1101.08
ImpPlaid	816 (48×17)	2323.56
FLEXBIC ($\delta = 2323.50$)	113435 (2315×49)	2323.02

(b) On the *Lymphoma* dataset

Table 4.5: Performance comparison of five algorithms on the largest biclusters.

Algorithm	Coverage Percentage
FLEXBIC	75.43%
FLOC	17.06%
BICRELS	61.44%
ChengChurch	56.59%
ImpPlaid	5.63%

(a) On the *Yeast* dataset

Algorithm	Coverage Percentage
FLEXBIC	37.56%
FLOC	0.77%
BICRELS	33.29%
ChengChurch	17.96%
ImpPlaid	0.22%

(b) On the *Lymphoma* dataset

Table 4.6: The coverage percentage of 10 biclusters of fives algorithms over the full matrix.

of columns can be decreased to find more similar rows. To this end, our algorithm allows users to control the role of rows and columns by adjusting the balance factor θ of **BICRELS** and the search range parameter (α_1, α_2) of **FLEXBIC**.

To illustrate this, we set different values for the balance factor θ of **BICRELS** and assign the search range parameter (α_1, α_2) of **FLEXBIC** to $(0, 0.5)$. Other settings are the same as in previous sections. Table 4.7 and Table 4.8 show the largest biclusters obtained by two algorithms on the two datasets. By increasing the balance factor **BICRELS** produces biclusters with a larger numbers of columns. A similar behavior can be observed for **FLEXBIC**. However, there exists in some special situations a larger balance factor for **FLEXBIC** does not increase the number of columns. For example, on the *Lymphoma* dataset, the bicluster obtained with $\theta = 1.0$ contains 29 columns whereas the bicluster returned by $\theta = 2.5$ contains 28 columns. The main reason is that if two balance factors produce core biclusters with similar numbers of columns, then the expansion step of **FLEXBIC** will search in similar ranges of numbers. In addition, as the core biclusters of two balance factors are different, **FLEXBIC** starts the expansion step from two different initial points. From these facts, if two balance factors are very close, then a bicluster with a larger number of columns can be obtained with a smaller balance factor.

4.6 Conclusion

In this chapter, we proposed a novel algorithm for subspace clustering. Our algorithm is flexible and can be applied in different domains as different analyzing models can be used without modifying its overall structure. Besides, **FLEXBIC** can generate K overlapping biclusters where the maximum overlap between them is below a predefined threshold. Our algorithm

Balance Factor (θ)	Max Volume ($\|I\| \times \|J\|$)
1.0	16577 (1507 × **11**)
1.5	16968 (1414 × **12**)
2.0	16458 (1266 × **13**)
2.5	15148 (1082 × **14**)
3.0	13056 (816 × **16**)
3.5	11832 (696 × **17**)

(a) On the *Yeast* dataset

Balance Factor (θ)	Max Volume ($\|I\| \times \|J\|$)
1.0	43907 (1909 × **23**)
2.5	40575 (1623 × **25**)
4.0	35048 (674 × **52**)
5.5	16492 (217 × **76**)
7.0	10323 (111 × **93**)
8.5	10323 (111 × **93**)

(b) On the *Lymphoma* dataset

Table 4.7: Largest biclusters produced by **BICRELS** with different balance factors on two real datasets.

Balance Factor (θ)	Max Volume ($\|I\| \times \|J\|$)
1.0	16968(1414 × **12**)
1.5	16968(1414 × **12**)
2.0	16458(1266 × **13**)
2.5	15148(1082 × **14**)
3.0	13056(816 × **16**)
3.5	11832(696 × **17**)

(a) On the *Yeast* dataset

Balance Factor (θ)	Max Volume ($\|I\| \times \|J\|$)
1.0	45878(1582 × **29**)
2.5	42700(1525 × **28**)
4.0	35048(674 × **52**)
5.5	16492(217 × **76**)
7.0	10323(111 × **93**)
8.5	10323(111 × **93**)

(b) On the *Lymphoma* dataset

Table 4.8: Largest biclusters produced by **FLEXBIC** with different balance factors on two real datasets.

also allows users to control the bicluster shape by adjusting the relative row and column weight. When implemented with different and specific metrics and various ways to measure the bicluster quality, the experimental results confirm that our algorithm can discover different overlapping biclusters on gene expression data and disjoint biclusters on UCI datasets. Besides, under the same constraints, our algorithm produces much larger and higher-quality biclusters when compared to those of the other state-of-the-art algorithms.

Chapter 5

Conclusion and Future Work

In this thesis, we discussed the main issues of alternative clustering and non-redundant overlapping subspace clustering, and proposed novel algorithms for solving them effectively.

In Chapter 3, after observing that alternative clustering is a multi-objective optimization problem, and most state-of-the-art approaches solve it sequentially or indirectly, we proposed a direct multi-objective framework for approximating the Pareto front of solutions. Our method not only produces better solutions than those of the competition algorithms, but it also is more flexible because it allows the use of different clustering objectives instead of fixing them as in the previous approaches.

In Chapter 4, to deal with the redundancy and overlap problem in subspace clustering, we introduced a biclustering algorithm which can produce overlapping biclusters where the maximum overlap between them is below a predefined threshold. The bicluster shape can also be controlled by setting the relative row and column weight. In addition, different analyzing models can be used under our algorithm. Hence, it can be applied to several domains without modifying its main structure.

In the future, we plain to develop parallel versions of our algorithms and to test them on very large datasets. As our algorithms do not depend on

specific domains, extending them to new applications will be an interesting topic.

Bibliography

[1] Charu C Aggarwal, Joel L Wolf, Philip S Yu, Cecilia Procopiuc, and Jong Soo Park. Fast algorithms for projected clustering. In *ACM SIGMOD Record*, volume 28, pages 61–72. ACM, 1999.

[2] A.A. Alizadeh, M.B. Eisen, R.E. Davis, C. Ma, I.S. Lossos, A. Rosenwald, J.C. Boldrick, H. Sabet, T. Tran, X. Yu, et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–511, 2000.

[3] E. Bae and J. Bailey. Coala: A novel approach for the extraction of an alternate clustering of high quality and high dissimilarity. In *Proceeding of the Sixth IEEE International Conference on Data Mining*, pages 53–62. IEEE, 2006.

[4] P. Baldi and G.W. Hatfield. *DNA Microarrays and Gene Expression: From Experiments to Data Analysis and Modelling*. Cambridge University Press, 2002.

[5] G.H. Ball. Isodata, a novel method of data analysis and pattern classification. Technical report, DTIC Document, 1965.

[6] A. Banerjee and J. Ghosh. Clustering with balancing constraints. *Constrained clustering: advances in algorithms, theory, and applications*, page 171, 2008.

[7] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 19–26, 2002.

[8] S. Basu, A. Banerjee, and R.J. Mooney. Active semi-supervision for pairwise constrained clustering. In *Proceedings of the SIAM international conference on data mining*, pages 333–344, 2004.

[9] S. Basu, M. Bilenko, and R.J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 59–68. ACM, 2004.

[10] R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*. Operations research/Computer Science Interfaces. Springer Verlag, 2008. ISBN: 978-0-387-09623-0.

[11] R. Battiti and A. Passerini. Brain–computer evolutionary multiobjective optimization: A genetic algorithm adapting to the decision maker. *IEEE Transactions on Evolutionary Computation*, 14(5):671–687, 2010.

[12] P. Berkhin. Survey of clustering data mining techniques. *Grouping Multidimensional Data: Recent Advances in Clustering*, pages 25–71, 2006.

[13] T. De Bie. Subjectively interesting alternative clusters. In *Proceedings of the 2nd MultiClust Workshop: Discovering, Summarizing, and Using Multiple Clusterings. Athens, Greece: CEUR Workshop Proceedings (CEUR-WS. org)(online)*, pages 43–54, 2011.

[14] C.M. Bishop and SpringerLink (Online service). *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.

[15] D. Boley. Principal direction divisive partitioning. *Data mining and knowledge discovery*, 2(4):325–344, 1998.

[16] E. I. Boyle, S. Weng, J. Gollub, H. Jin, D. Botstein, J M. Cherry, and G. Sherlock. Go:: Termfinder? open source software for accessing gene ontology information and finding significantly enriched

gene ontology terms associated with a list of genes. *Bioinformatics*, 20(18):3710–3715, 2004.

[17] J. Branke, K. Deb, and K. Miettinen. *Multiobjective optimization: Interactive and evolutionary approaches*, volume 5252. Springer-Verlag New York Inc, 2008.

[18] D.E. Brown and C.L. Huntley. A practical application of simulated annealing to clustering. *Pattern Recognition*, 25(4):401–412, 1992.

[19] Y. Cheng and G.M. Church. Biclustering of expression data. In *Proceedings of the eighth international conference on intelligent systems for molecular biology*, volume 8, pages 93–103, 2000.

[20] D. Cohn, R. Caruana, and A. McCallum. Semi-supervised clustering with user feedback. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*, pages 17–31, 2008.

[21] David Cohn, Rich Caruana, and Andrew Mccallum. Semi-supervised clustering with user feedback. Technical report, Cornell University, 2003.

[22] Y. Cui, X.Z. Fern, and J.G. Dy. Non-redundant multi-view clustering via orthogonalization. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 133–142. IEEE, 2007.

[23] X.H. Dang and J. Bailey. Generation of alternative clusterings using the cami approach. In *the SIAM International Conference on Data Mining*, pages 118–129, 2010.

[24] S. Dasgupta and V. Ng. Mining clustering dimensions. In *Proceedings of the 27th International Conference on Machine Learning*, pages 263–270, 2010.

[25] I. Davidson and Z. Qi. Finding alternative clusterings using constraints. In *The eighth IEEE International Conference on Data Mining*, pages 773–778. IEEE, 2008.

[26] I. Davidson and SS Ravi. Clustering with constraints: Feasibility issues and the k-means algorithm. In *Proceedings of the Fifth SIAM International Conference on Data Mining*, volume 119, page 138. Society for Industrial Mathematics, 2005.

[27] Ian Davidson and Sugato Basu. A survey of clustering with instance level constraints. *ACM Transactions on Knowledge Discovery from Data*, 2007.

[28] W.H.E. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of classification*, 1(1):7–24, 1984.

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[30] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364, 1977.

[31] A. Demiriz, K.P. Bennett, and P.S. Bradley. Using assignment constraints to avoid empty clusters in k-means clustering. *Constrained clustering: advances in algorithms, theory, and applications*, page 201, 2008.

[32] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[33] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern classification*, volume 2. wiley New York:, 2001.

[34] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

[35] V. Estivill-Castro and J. Yang. Fast and robust general purpose clustering algorithms. *PRICAI 2000 Topics in Artificial Intelligence*, pages 208–218, 2000.

[36] E. Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.

[37] E.W. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[38] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[39] G. Getz, H. Gal, I. Kela, D.A. Notterman, and E. Domany. Coupled two-way clustering analysis of breast cancer and colon cancer gene expression data. *Bioinformatics*, 19(9):1079–1089, 2003.

[40] D. Gondek and T. Hofmann. Conditional information bottleneck clustering. In *3rd ieee international conference on data mining, workshop on clustering large data sets*, pages 36–42. Citeseer, 2003.

[41] D. Gondek and T. Hofmann. Non-redundant data clustering. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 75–82. IEEE, 2004.

[42] D. Gondek and T. Hofmann. Non-redundant clustering with conditional ensembles. In *Proceedings of the eleventh ACM SIGKDD in-*

ternational conference on Knowledge discovery in data mining, pages 70–77. ACM, 2005.

[43] D. Greene and P. Cunningham. Constraint selection by committee: An ensemble approach to identifying informative constraints for semi-supervised clustering. *Machine Learning: ECML 2007*, pages 140–151, 2007.

[44] N. Grira, M. Crucianu, and N. Boujemaa. Active semi-supervised fuzzy clustering for image database categorization. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 9–16. ACM, 2005.

[45] S. Günnemann, I. Färber, and T. Seidl. Multi-view clustering using mixture models in subspace projections. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 132–140. ACM, 2012.

[46] S. Günnemann, E. Müller, I. Färber, and T. Seidl. Detection of orthogonal concepts in subspaces of high dimensional data. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1317–1326. ACM, 2009.

[47] J. Handl and J. Knowles. An evolutionary approach to multiobjective clustering. *Evolutionary Computation, IEEE Transactions on*, 11(1):56–76, 2007.

[48] J.A. Hartigan. Direct clustering of a data matrix. *Journal of the american statistical association*, 67(337):123–129, 1972.

[49] E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information processing letters*, 76(4-6):175–181, 2000.

[50] T. Hertz, A. Bar-Hillel, and D. Weinshall. Boosting margin based distance functions for clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 50. ACM, 2004.

[51] E.R. Hruschka, R.J.G.B. Campello, A.A. Freitas, and A.C.P.L.F. de Carvalho. A survey of evolutionary algorithms for clustering. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(2):133–155, 2009.

[52] R. Huang, W. Lam, and Z. Zhang. Active learning of constraints for semi-supervised text clustering. In *Proceedings of the SIAM International Conference on Machine Learning*, pages 113–124, 2007.

[53] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[54] P. Jain, R. Meka, and I.S. Dhillon. Simultaneous unsupervised learning of disparate clusterings. *Statistical Analysis and Data Mining*, 1(3):195–210, 2008.

[55] Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. Density-connected subspace clustering for high-dimensional data. In *Proceeding of the 4th SIAM International Conference on Data Mining*, volume 4, pages 246–257, 2004.

[56] L. Kaufman, P.J. Rousseeuw, and Ebooks Corporation. *Finding groups in data: an introduction to cluster analysis*, volume 39. Wiley Online Library, 1990.

[57] S. Kirkpatrick, C.D. Gelatt Jr, and M.P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[58] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based

clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1, 2009.

[59] K. Krishna, N. Murty, et al. Genetic k-means algorithm. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(3):433–439, 1999.

[60] G.N. Lance and W.T. Williams. A general theory of classificatory sorting strategies. *The computer journal*, 9(4):373, 1967.

[61] L. Lazzeroni, A. Owen, et al. Plaid models for gene expression data. *Statistica Sinica*, 12(1):61–86, 2002.

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

[63] Z. Lu and T.K. Leen. Pairwise constraints as priors in probabilistic clustering. *Constrained clustering: advances in algorithms, theory, and applications*, page 59, 2008.

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

[65] S.C. Madeira and A.L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 1(1):24–45, 2004.

[66] G.J. McLachlan and T. Krishnan. *The EM algorithm and extensions*. LibreDigital, 2008.

[67] A. Moore. Very fast em-based mixture model clustering using multiresolution kd-trees. In *In Advances in Neural Information Processing Systems 11*. Citeseer, 1999.

[68] Emmanuel Müller, Stephan Günnemann, Ira Assent, and Thomas Seidl. Evaluating clustering in subspace projections of high dimensional data. *Proceedings of the VLDB Endowment*, 2(1):1270–1281, 2009.

[69] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[70] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354, 1983.

[71] D. Niu, J.G. Dy, and M.I. Jordan. Multiple non-redundant spectral clustering views. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 831–838, 2010.

[72] D. Pelleg and D. Baras. K-means with large and noisy constraint sets. *Machine Learning: ECML 2007*, pages 674–682, 2007.

[73] J.M. Pena, J.A. Lozano, and P. Larranaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern recognition letters*, 20(10):1027–1040, 1999.

[74] Z.J. Qi and I. Davidson. A principled and flexible framework for finding alternative clusterings. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 717–726. ACM, 2009.

[75] W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, pages 846–850, 1971.

[76] S.M. Savaresi and D.L. Boley. On the performance of bisecting k-means and pddp. *Proceedings of the 1 st SIAM ICDM, Chicago, IL*, 2001.

[77] S.M. Savaresi, D.L. Boley, S. Bittanti, and G. Gazzaniga. Cluster selection in divisive clustering algorithms. *Proceedings of the 2nd SIAM*, 1401:299–314, 2002.

[78] R. Sharan and R. Shamir. Click: A clustering algorithm with applications to gene expression analysis. In *Proc Int Conf Intell Syst Mol Biol*, volume 8, page 16, 2000.

[79] N. Shental, A. Bar-Hillel, T. Hertz, and D. Weinshall. Gaussian mixture models with equivalence constraints. *Constrained clustering: advances in algorithms, theory, and applications*, page 33, 2008.

[80] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30, 1973.

[81] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Citeseer, 2000.

[82] K. Stoffel and A. Belkoniene. Parallel k/h-means clustering for large data sets. *Euro-Par'99 Parallel Processing*, pages 1451–1454, 1999.

[83] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18(suppl 1):S136–S144, 2002.

[84] S. Tavazoie, J.D. Hughes, M.J. Campbell, R.J. Cho, and G.M. Church. Systematic determination of genetic network architecture. *Nature genetics*, 22:281–285, 1999.

[85] N. Tishby, F.C. Pereira, and W. Bialek. The information bottleneck method. *Arxiv preprint physics/0004057*, 2000.

[86] D.T. Truong and R. Battiti. A cluster-oriented genetic algorithm for alternative clustering. In *Proceedings of the 12th International Conference on Data Mining (ICDM12)*, pages 1122–1127. IEEE, 2012.

[87] D.T. Truong and R. Battiti. A flexible cluster-oriented alternative clustering algorithm for choosing from the Pareto front of solutions. *Machine Learning*, pages 1–35, 2013.

[88] D.T Truong, R. Battiti, and M. Brunato. Discovering non-redundant overlapping biclusters on gene expression data. In *Proceeding of the 13th IEEE International Conference on Data Mining (ICDM 2013)*. IEEE, 2013.

[89] D.T. Truong, R. Battiti, and M. Brunato. A repeated local search algorithm for biclustering of gene expression data. In E. Hancock and M. Pelillo, editors, *Proceedings of the 2nd International Workshop on Similarity-Based Pattern Analysis and Recognition (SIMBAD 2013)*, number 7953 in LNCS, pages 281–296. Springer Verlag, 2013.

[90] H. Turner, T. Bailey, and W. Krzanowski. Improved biclustering of microarray data demonstrated through systematic performance tests. *Computational statistics & data analysis*, 48(2):235–254, 2005.

[91] M. Van der Laan, K. Pollard, and J. Bryan. A new partitioning around medoids algorithm. *Journal of Statistical Computation and Simulation*, 73(8):575–584, 2003.

[92] N.X. Vinh and J. Epps. mincentropy: A novel information theoretic approach for the generation of alternative clusterings. In *Proceed-*

ings of the 10th International Conference on Data Mining (ICDM10), pages 521–530. IEEE, 2010.

[93] E.M. Voorhees. Implementing agglomerative hierarchic clustering algorithms for use in document retrieval. *Information Processing & Management*, 22(6):465–476, 1986.

[94] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrodl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584. Citeseer, 2001.

[95] E.P. Xing, A.Y. Ng, M.I. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. *Advances in neural information processing systems*, pages 521–528, 2003.

[96] R. Xu, D. Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.

[97] J. Yang, H. Wang, W. Wang, and P. Yu. Enhanced biclustering on expression data. In *Bioinformatics and Bioengineering, 2003. Proceedings. Third IEEE Symposium on*, pages 321–327. IEEE, 2003.

[98] M.L Yiu and N. Mamoulis. Frequent-pattern based iterative projected clustering. In *Proceeding of the Third IEEE International Conference on Data Mining*, pages 689–692. IEEE, 2003.

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

[100] W. Zhao, Q. He, H. Ma, and Z. Shi. Effective semi-supervised document clustering via active learning with instance-level constraints. *Knowledge and Information Systems*, pages 1–19, 2011.

Appendix A

Appendix

A.1 Analysis of the Pareto front

In this section, we plot all border solutions of **COGNAC** on four datasets (Section 3.4.2) by applying the analysis procedure in Section 3.3.4. We arrange the solutions in the ascending order of dissimilarity.

Table A.1a and A.1b show the 10 border solutions of the first partitioning and 20 border solutions of the second partitioning (on the last group) on the *CMUFaces* dataset, respectively. Table A.2 presents the 10 border solutions of the first partitioning on the *WebKB* dataset. Fig.A.1 and Fig.A.2 depict the 10 border solutions of the first partitioning on the *Birds* and *Flowers* dataset. As it can be seen, the first alternative solution is very similar to the negative solution, but then the dissimilarity is increased gradually, and at some point, the solution is transformed into a completely different one. Besides, as the border solution with the highest dissimilarity of a group and the border solution with the highest quality of the next group are very close to each other, therefore their results are very similar.

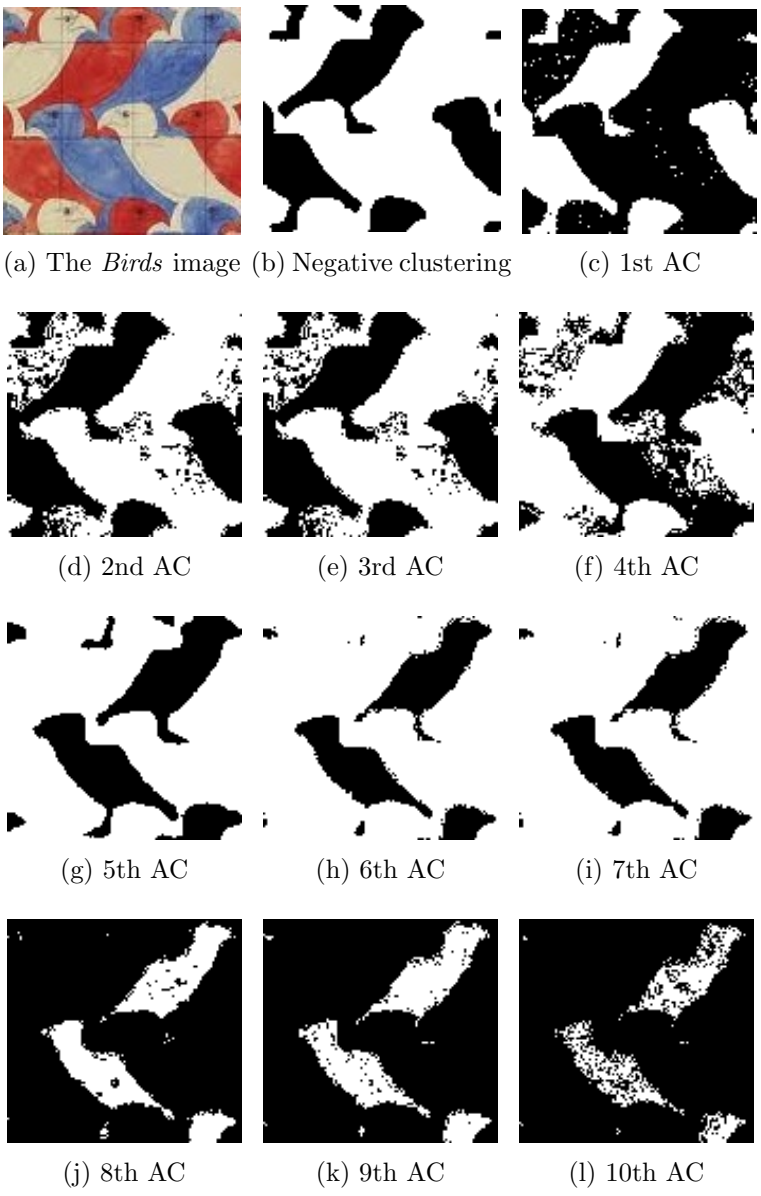
Cluster 1	Cluster 2	Cluster 3	Cluster 4
up(0.26)	straight(0.39)	left(0.77)	right(0.93)
up(0.35)	straight(0.31)	left(0.88)	right(0.86)
up(0.35)	straight(0.32)	left(0.80)	right(0.87)
straight(0.34)	left(0.77)	left(0.41)	right(0.84)
straight(0.34)	left(0.77)	left(0.44)	right(0.83)
straight(0.33)	left(0.63)	right(0.89)	right(0.63)
straight(0.33)	left(0.65)	right(0.89)	right(0.59)
straight(0.41)	left(0.62)	left(0.70)	right(0.84)
straight(0.41)	left(0.67)	left(0.71)	right(0.85)
straight(0.41)	left(1.00)	left(0.70)	right(0.86)

(a) 10 alternative clusterings in the first partitioning

Cluster 1	Cluster 2	Cluster 3	Cluster 4
straight(0.41)	left(0.67)	left(0.71)	right(0.85)
straight(0.41)	left(0.80)	left(0.70)	right(0.85)
straight(0.41)	left(0.70)	left(0.77)	right(0.86)
straight(0.41)	left(0.89)	left(0.71)	right(0.86)
straight(0.41)	left(0.92)	left(0.70)	right(0.85)
straight(0.40)	left(0.94)	left(0.70)	right(0.86)
straight(0.41)	left(0.94)	left(0.70)	right(0.86)
straight(0.41)	left(0.70)	left(0.97)	right(0.86)
straight(0.40)	left(0.97)	left(0.70)	right(0.86)
straight(0.41)	left(0.64)	left(1.00)	right(0.86)
straight(0.38)	left(0.91)	left(0.72)	right(0.88)
up(0.63)	**straight(0.44)**	**left(0.70)**	**right(0.86)**
up(0.63)	straight(0.44)	left(0.70)	right(0.86)
up(0.36)	straight(0.42)	left(0.70)	right(0.86)
straight(0.42)	straight(0.35)	left(0.70)	right(0.87)
up(0.39)	straight(0.42)	left(0.70)	right(0.86)
up(0.41)	straight(0.42)	left(0.70)	right(0.86)
straight(0.42)	left(0.43)	left(0.69)	right(0.86)
up(0.42)	straight(0.41)	left(0.70)	right(0.86)
straight(0.41)	left(1.00)	left(0.70)	right(0.86)

(b) 20 alternative clusterings in the second partitioning on the last group

Table A.1: Alternative clusterings (in two partitionings) of **COGNAC** on the *CMUFaces* dataset



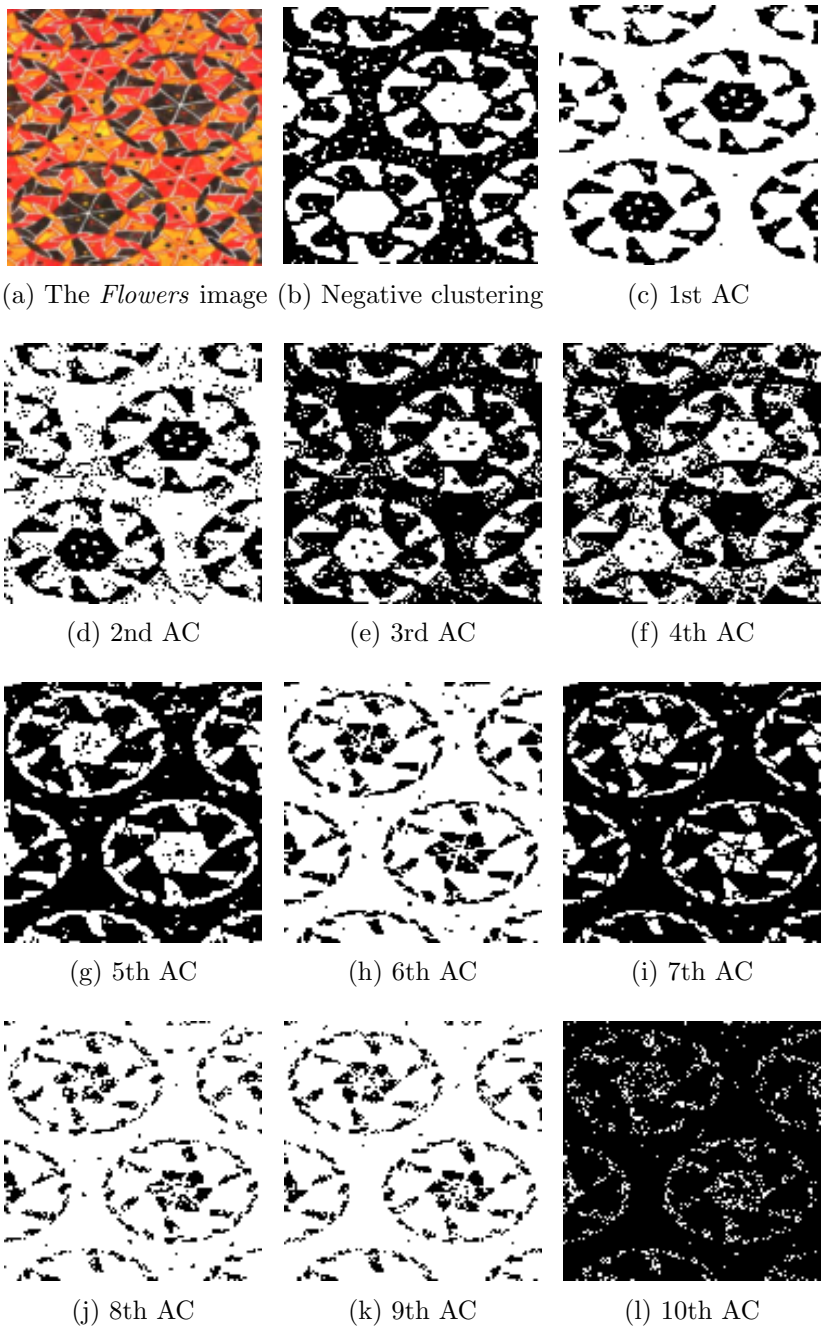
(a) The *Birds* image (b) Negative clustering (c) 1st AC

(d) 2nd AC (e) 3rd AC (f) 4th AC

(g) 5th AC (h) 6th AC (i) 7th AC

(j) 8th AC (k) 9th AC (l) 10th AC

Figure A.1: 10 alternative clusterings (AC) of **COGNAC** in the first partitioning on the *Birds* dataset



(a) The *Flowers* image (b) Negative clustering (c) 1st AC

(d) 2nd AC (e) 3rd AC (f) 4th AC

(g) 5th AC (h) 6th AC (i) 7th AC

(j) 8th AC (k) 9th AC (l) 10th AC

Figure A.2: 10 alternative clusterings (AC) of **COGNAC** in the first partitioning on the *Flowers* dataset

Cluster 1	Cluster 2	Cluster 3	Cluster 4
texas(0.98)	**cornell(0.99)**	**wisconsin(0.82)**	**washington(0.97)**
texas(1.00)	cornell(1.00)	wisconsin(0.65)	washington(1.00)
texas(1.00)	cornell(1.00)	wisconsin(0.64)	washington(1.00)
texas(1.00)	cornell(1.00)	wisconsin(0.57)	washington(1.00)
texas(1.00)	cornell(1.00)	wisconsin(0.56)	washington(1.00)
texas(1.00)	cornell(1.00)	wisconsin(0.51)	washington(0.99)
texas(1.00)	cornell(1.00)	wisconsin(0.50)	washington(1.00)
texas(1.00)	cornell(1.00)	wisconsin(0.45)	washington(1.00)
texas(1.00)	cornell(1.00)	wisconsin(0.44)	washington(1.00)
texas(1.00)	cornell(1.00)	wisconsin(0.37)	washington(1.00)

Table A.2: 10 alternative clusterings (in the first partitioning) of **COGNAC** on the *WebKB* dataset