UCLouvain

École polytechnique de Louvain

Pattern Visualisation in Heat Maps

Author: **Gauthier Van Vracem**
Supervisor: **Siegfried Nijssen**
Readers: **Pierre Schaus, Vincent Branders**
Academic year 2020-2021
Master [120] in Computer Science

Abstract

Data visualization is an important tool in data science to quickly capture the main patterns and trends in large datasets. One of these techniques concerns *Heatmaps* and allows the visualization of matrices or tables in their entirety. This approach assumes that the order of the rows and columns can be modified in order to highlight patterns of interest in the analysis. Unfortunately, existing techniques that aim at finding ideal permutations are often inefficient to obtain a meaningful order when the data are characterized by a high degree of noise. To address this issue, this thesis introduces a new framework that boosts these inefficient methods by integrating them into an iterative process using convolution. The key idea being to temporarily transform the noisy matrix into a simpler model to reorder. The approach is intended to be as generic as possible by being able to process both numerical and binary data while offering satisfactory execution times. We show that this approach is efficient to improve the quality of the output of all tested basic methods, applicable on both synthetic datasets where the pattern is known and on real-world data.

Keywords: Data visualization, Matrix reordering, Matrix seriation, Convolution, Local-Search, Pattern mining, Framework

Acknowledgments

I would like to thank Prof. Siegfried Nijssen for his availability and his time throughout the year. As well as assisting me during the writing of the paper.

I would also like to thank my family for supporting me during my research and writing of this thesis.

Contents

Symbols

$\mathbf{M}$	Matrix or data table
$\mathbf{M}_i$	i-th row of the matrix
$\mathbf{M}_{i,j}$	Value at row i and column j of the matrix
$\mathbf{K}$	Kernel
$\mathbf{D}$	Distance matrix
$\mathbf{T}$	Template matrix
$\mathbf{M} * \mathbf{K}$	Convolution
$\mathbf{K}_c$	Kernel criterion
κ	Set of kernels
φ	Ordering method
$\mathcal{D}$	Matrix domain
π	Permutation
$\mathcal{P}_n$	Set of all permutations of n elements
d	Distance metric
GW	Gruvaeus and Wainer
HC	Hierachical clustering
MDS	Multidimensional scaling
OLO	Optimal leaf ordering
QAP	Quadratic assignment problem
TSP	Traveling salesman problem

CHAPTER 1

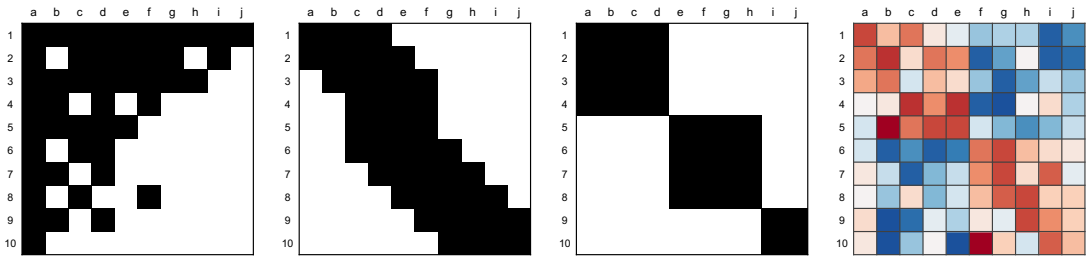
Introduction

1.1 Context presentation

Nowadays, more and more information is gathered and collected from various sources, whether from the internet or by means of research experiments. Data scientists or researchers who analyze these data need to develop powerful tools that allow them to visualize this huge amount of information in a graphical and understandable way. These visualization methods can be very diverse and generally depend on the type of data that is analyzed (time series, scatter plot, histogram, etc). In data science, data visualization tools are generally used as a preliminary tool to perform more detailed studies later.

One of these approaches concerns the *Heatmaps* or more generally *matrix visualization*, an exploratory data analysis method that allows to visualize a matrix or double entry table in its entirety. These two-dimensional data can be binary as well as numerical. They generally describe relationships between different sets of objects or features. In many cases, the order in which the rows and columns are displayed is arbitrary and thus can be reordered allowing different views of the data set in order to highlight a specific pattern. The challenge is that this order is not necessarily known and it is necessary to implement different *matrix reordering* algorithms that aim to find a permutation that displays the data the best.

A properly ordered matrix tends to put similar rows and columns close together and reveals different patterns that are subject of interest to understand the data.



(a) Nested (b) Banded (c) Blocks (d) Heatmap

Figure 1.1: Example of patterns

Some toy examples of these patterns are shown in Figure 1.1. The first three matrices show binary data. The data can display a *nested structure*, i.e. there is a hierarchy/subgroup relationship in the data. This pattern is strongly studied in ecology. It can also describe a linear or *banded* structure commonly used in archaeology. And finally, one of the most frequent patterns are the *blocks* patterns, these models generally describe data that are strongly partitioned in the form of clusters i.e. cohesive groups. The last matrix shows the type of pattern that is usually obtained with numerical data. This method where a color spectrum is assigned is generally called *Heatmap* and is commonly used in bioinformatics.

1.2 Research Questions

In this thesis, we study the problem of correctly ordering matrices in order to display the underlying patterns the best. The objective is to improve the existing methods while offering high levels of computational performance as well. The work will be guided and challenged by the following research questions:

Noise: What are the different weaknesses of existing methods?

We will first analyze and classify the existing reordering methods in order to clearly identify their weaknesses. We will see that these methods work very well for artificially generated data where the pattern is known. However, these same methods often fail to reorder correctly when *noise* is introduced in the data. This problem is particularly important because real data are often described by very complex and noisy patterns.

Framework: How to improve existing methods?

Instead of creating a brand new algorithm built from scratch, we will present a framework that allows to integrate any ordering matrix method (as a black box component) by improving it through *convolution*. Convolution is a method of image processing that allows to blur an image for example. The idea is to use an inefficient method and to boost it by temporarily transforming the matrix via different techniques (blurring, thresholding) in order to remove the noise. The proposed approach will work in an iterative way. We will see that this algorithm initially designed for binary data can be easily adapted for other types of data (numerical, categorical) and is particularly resilient to noise and sparsity no matter the size of the matrix.

Past work and criterion: How does the framework compares to ConvoMap?

This thesis is built on the previous work of T. Bollen and G. Leurquin [2]. They tackled a similar problem and introduced the notion of *evaluation criterion*. An evaluation criterion is a metric that assigns a score to the order of a matrix with the aim of evaluating its quality. This task is not trivial because it requires to find a universal and generic method whereas the quality of the order is generally evaluated in a subjective way. For this purpose, they introduced a criterion also based on *convolution* and demonstrated that it could be used as a relevant universal optimization criterion. They built an innovative reordering algorithm (ConvoMap) based on this score.

In this work we will reuse this criterion as a way to evaluate the quality of a matrix. But the proposed framework is radically different since we use other image processing techniques. We will also try to improve the performances compared to ConvoMap, both in terms of execution time and quality.

1.3 Thesis structure

In Chapter 2, we formalize the matrix reordering problem and describe the existing methods in the literature and their main limitations. Chapter 3 presents the concept of evaluation criterion and introduces the theoretical basis of convolution. We then present the ConvoMap algorithm and how it is based on convolution. Chapter 4 presents the main contribution of the dissertation: the iterative reordering framework and its different components which are explained in detail. In Chapter 5, we evaluate the performance of the framework by conducting different experiments on several datasets. Finally, in Chapter 6, we step back from what has been done and lay the foundation for future improvements.

CHAPTER 2

Related work

This chapter introduces the theoretical basis of matrix reordering. We will first specify the different types of data that can be reordered. Then we will formally define the reordering problem. Finally, we will briefly present and classify the most common existing methods present in the literature and their limitations.

2.1 Types of data

We will use the term matrix denoted by $\mathbf{M}$ to define any "data table" that can be reordered by swapping rows and columns. It is however necessary to classify the different types of matrices as some methods do not apply universally to all types. It is also important to define such a classification in order to clearly understand what data the matrix represents. There are three ways to classify matrices: the attributes, the mode and the density.

Attributes

The first classification is based on the content of the matrix and more specifically the entries or attributes. Each entry is mapped to a specific value, $\mathbf{M}_{i,j} \in \mathcal{D}$.

- **Binary**: The simplest version are the boolean or binary matrices. They are composed of only two values $\mathcal{D} = \{0, 1\}$. These matrices tend to model relations between different sets of objects. The value 1 means that there is a relationship (and conversely 0 the absence of relationship).

- **Categorical**: In some data the attributes can take more than two possible values. These data remain however discrete (finite set of countable objects) and are referred to as categorical $\mathcal{D} = \{c_1, c_2, ..., c_k\}$. These categories can be the same for the whole matrix or specific to columns or rows.
- **Numerical**: Some matrices are characterized by numerical values, either integers or real numbers. These data sometimes require a pre-processing which consists in normalizing the set of values so that it fits a certain interval e.g. $\mathcal{D} = [0, 1]$. The term heatmap is generally used for this kind of data where a color code is assigned to each value.

Within this thesis, we will focus mainly on binary data, but we will see that the algorithms are also suitable for the other types as well.

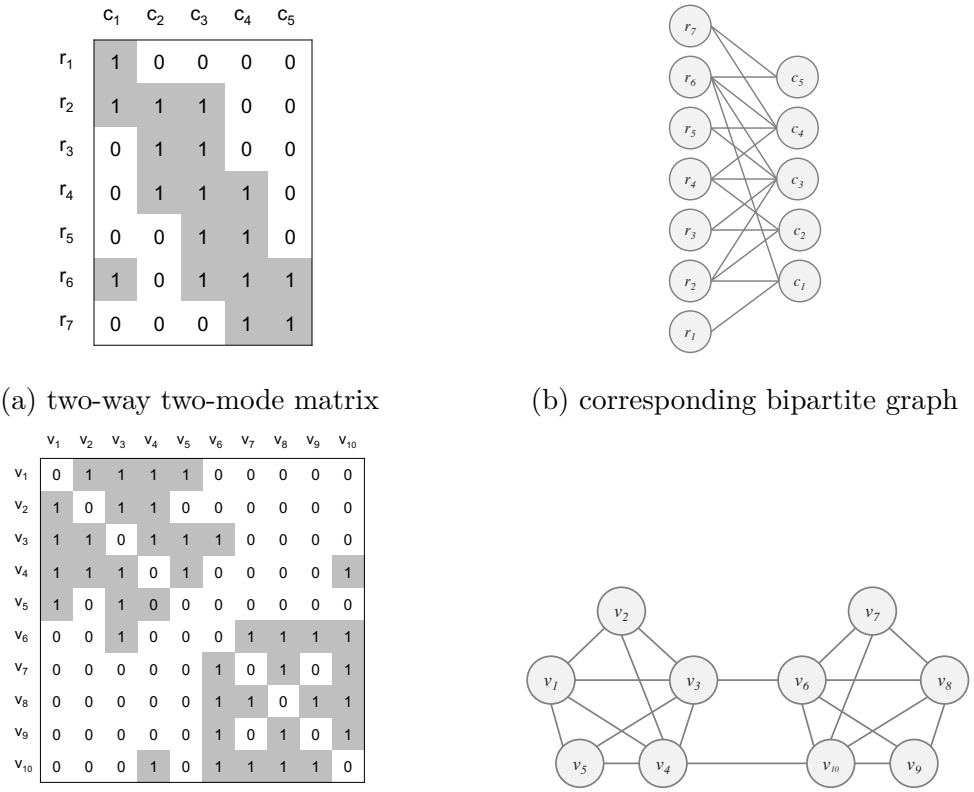
Mode

We have seen that matrices are in fact double-entry tables, meaning that the columns and rows are labeled and represent a set of "objects" or "features". This is why matrix visualization is called *two-way* visualization. But an important distinction is the "*mode*", which sets do the columns and rows represent? We distinguish two types of mode [6].

- **two-way two-mode**: These data represent the relationship between two sets of objects (two-mode). For example `<sample, feature>`, `<transaction, item>`. The matrix is of size $m \times n$ where m and n are not necessarily equal. We will use the term *tabular* data. The matrix can be seen as an SQL table or an excel table. The two-way two-mode binary matrices represent a bipartite graph as shown in Figure 2.1b. The rows and columns of such matrices can be ordered independently from each other.
- **two-way one-mode**: A more specific type of matrix is considered as one-mode. They are of size $n \times n$ and symmetric. These data represent a single set of n objects present in both columns and rows; the same order is used. These matrices model a relation between each object of the set. Binary matrices generally represent a graph as shown in Figure 2.1d. Each row of the matrix can thus be seen as a node and $\mathbf{M}_{i,j} = 1$ means that node i and j are connected.

Density

A last possible distinction concerns binary matrices and their "density". Some matrices are considered *sparse*, meaning they contain very few 1 s (or non-zero



(a) two-way two-mode matrix

(b) corresponding bipartite graph

(c) two-way one-mode matrix

(d) corresponding graph

Figure 2.1: Binary matrices and corresponding graphs

elements for numerical). Many datasets from the real world are sparse and of large size (generally one-mode networks). This kind of data has to be treated in a different way because it is quite difficult to visualize all the information without loss. On the contrary, non-sparse matrices are considered as *dense*.

2.2 The reordering problem

Ordering or permuting a matrix consists in applying a permutation [3, 4, 5] to the rows and columns. We define the permutation π on $[n]$ or $\{1, .., n\}$ as a bijection:

$$\pi : [n] \rightarrow [n] \tag{2.1}$$

For each element of a set of n elements we associate a permutation function $\pi(i) = j$ which moves the element from position j to i. $\mathcal{P}_n$ denotes the set of all possible permutations π on $[n]$. The total number of permutations $|\mathcal{P}_n|$ is equal to $n!$.

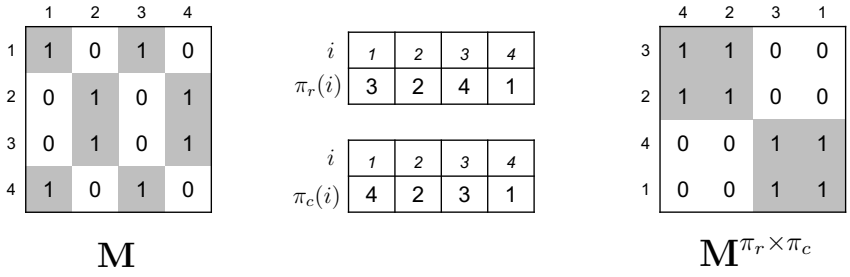


Figure 2.2: Example of permutations on tabular data

Permuting or ordering a matrix consists in assigning a permutation function to the columns and rows. In the case of *tabular* or 2-mode data, two permutations must be assigned (Figure 2.2). On the other hand, only one permutation is assigned on one-mode data for both columns and rows.

Given a matrix $\mathbf{M}$ of size $m \times n$ (*two-mode*), a row permutation $\pi_r \in \mathcal{P}_m$ and column permutation $\pi_c \in \mathcal{P}_n$, we define the permuted matrix $\mathbf{M}^{\pi_r \times \pi_c}$ as follows:

$$\mathbf{M}_{i,j}^{\pi_r \times \pi_c} = \mathbf{M}_{\pi_r(i),\pi_c(j)} \tag{2.2}$$

The *matrix reordering problem* [7] consists in finding a permutation (or an order) for the columns and rows in order to display a specific pattern or structure. As shown in Chapter 1, this task is not trivial since many patterns are possible and it depends on the analysis context. Finding a good permutation involves minimizing or maximizing a specific objective function also called criterion. The reordering of a matrix $\mathbf{M}$ of size $m \times n$ with a given loss function L could be thus seen as the following combinatorial optimization problem:

$$\underset{\pi_r \in \mathcal{P}_m,\, \pi_c \in \mathcal{P}_n}{\operatorname{argmin}} \; L(\mathbf{M}^{\pi_r \times \pi_c}) \tag{2.3}$$

One of the main challenges is the plurality of criterion functions. Indeed, there is no consensus or universal method since the quality of the order is generally subjective. For the case of binary matrices for example, there is a large variety of possible functions to optimize. Some of them are structural and involve minimizing the distance of all the '1's from the main diagonal. Others involve compressing the matrix [7] and are related to the minimum description length (MDL) principle. Such methods for example try to minimize the number of consecutive 1's for each row [12].

Another challenge is related to the search space. A bruteforce approach (trying all possible permutations) applied to a matrix of size $m \times n$ would involve $m!n!$ possibilities. Some matrix reordering methods are $\mathcal{NP}$-hard problems like TSP. It is therefore impossible to make an exhaustive search. The use of approximation methods (partial enumeration, heuristics) is therefore necessary.

2.3 Existing methods

We will describe the most commonly used existing methods to reorder a matrix or heatmap. We have seen in the previous section that the task can be formally written as an optimisation problem; we will go further and describe some of these methods. Many of these methods generally involve the transformation of the reordering problem into another known problem. We will distinguish two families of methods: *structure-based* and *distance-based* [1].

2.3.1 Structure-based

The so-called *structure-based* methods [11] aim at swapping rows and columns so that the image looks like a specific *structure*. The following methods are exclusively applicable on binary data (either simple or bi-partite networks). They are focused on two types of patterns: *nested* and *banded*.

Nested ordering

This method is one of the simplest and fastest; it assumes that the matrix has a so-called *nested* structure. A nested structure usually exhibits a triangular shape and shows the existence of a hierarchical relationship between the different columns

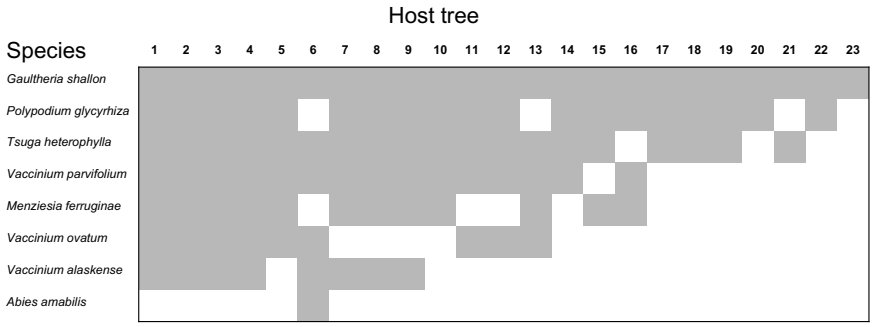
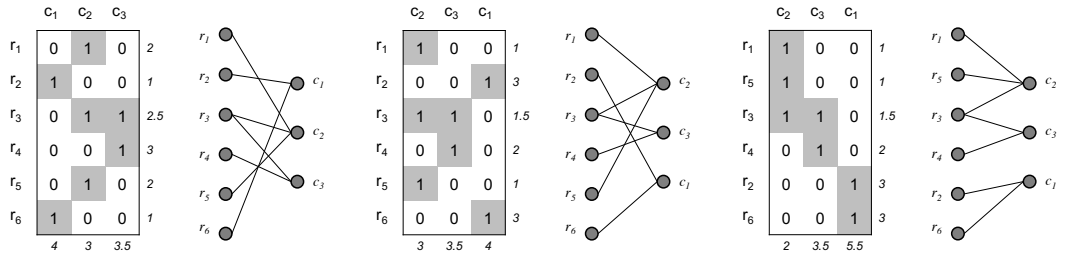


Figure 2.3: Nested ordering real example [10]



(a) (b) (c)

Figure 2.4: Barycenter heuristic example [14]

and rows. It is commonly used in ecology [8, 9] to model species systems through *species-sites matrix*. The idea is to count for each row the total number of 1's and then reorder the rows according to these counts; the same process is applied for the columns. A concrete example is illustrated in Figure 2.3 which shows real data [10]. It features 8 epiphyte species growing on a sample of 23 host trees. After applying a nested ordering, we see that there is a nested pattern involving a hierarchical relationship among the species.

Barycenter heuristic

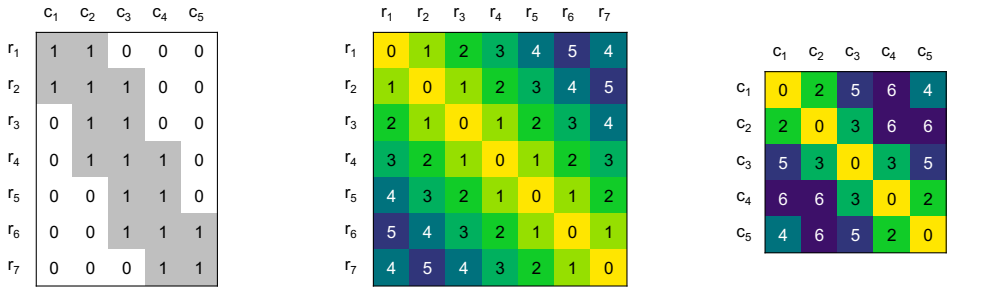
This method presented in [13] aims to discover banded structures. For each row, a barycentric score is calculated. It corresponds to the gravitational centre or the *average position* of all the 1's on the row. It is defined in the following way for a $m \times n$ matrix $\mathbf{M}$:

$$\text{Barycenter}\,(r) = \frac{\sum_{c=1}^{n} c \,.\, \mathbf{M}_{r,c}}{\sum_{c=1}^{n} \mathbf{M}_{r,c}} \tag{2.4}$$

Once the barycenter has been calculated for each row, they are ordered from lowest to highest value. After this, the matrix is transposed so that the same process is applied to the columns. The process is then iterated until convergence. Another way of visualising the problem is to draw the bipartite graph corresponding to the matrix. The barycentric method indirectly permutes the nodes in order to minimise the number of link crossings. A concrete example is shown in Figure 2.4a. Figure 2.4b shows the matrix after iterating over the columns, Figure 2.4c one step later over the rows.

2.3.2 Distance-based

The second family concerns the so-called *distance-based* methods. They aim at putting similar rows and columns close to each other. By maximizing the similarity



(a) Matrix table (b) Rows distance (c) Columns distance

Figure 2.5: Distance matrix (Hamming distance)

between each row (and column) we tend to obtain a visually ordered image with well defined and easily interpretable patterns. These methods do not seek to find a specific type of pattern and have the advantage of being adaptable to all types of data (numerical, categorical).

Since these methods are based on the *distance* or *dissimilarity* between rows and columns it is necessary to define this notion. These methods require to transform the initial matrix into a new one, called the *dissimilarity matrix*. Such matrix denoted by $\mathbf{D}$ of size $n \times n$ represents the pairwise distance between a set of n elements (one-mode) and is characterized by the following properties [15] :

- Symmetry: $\mathbf{D}_{i,j} = \mathbf{D}_{j,i}$ for all i, j.
- Non-negativity: $\mathbf{D}_{i,j} \geq 0$ for all i, j.
- Reflexivity: $\mathbf{D}_{i,j} = 0$ if i and j are equal.
- Triangle inequality: $\mathbf{D}_{i,k} \leq \mathbf{D}_{i,j} + \mathbf{D}_{j,k}$ for all i, j, k.

We use the notation $d(\mathbf{x}, \mathbf{x}')$ to model the distance between two vectors (two rows or columns for example). If we want to reorder a one-mode matrix, only one dissimilarity matrix is needed. For tabular data (2-mode), two matrices must be constructed. One for the rows: $\mathbf{D^r}_{i,j} = d(\mathbf{M}_i, \mathbf{M}_j)$ and one for the columns: $\mathbf{D^c}_{i,j} = d(\mathbf{M}_i^T, \mathbf{M}_j^T)$ as depicted in Figure 2.5, which results in two independent ordering problems.

To measure the distance $d(\mathbf{x}, \mathbf{x}')$ between two vectors of size n, we generally use the following *p-norm* metrics:

- Manhattan distance ($L1$-norm): $d_{L1}(\mathbf{x}, \mathbf{x}') = \sum_i^n |\mathbf{x}_i - \mathbf{x}'_i|$
- Euclidean distance ($L2$-norm): $d_{L2}(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_i^n (\mathbf{x}_i - \mathbf{x}'_i)^2}$

For binary matrices we generally use the Hamming distance, which models the difference between two sequences of 0 and 1 (xor operation); this measure corresponds to the $L1$-norm.

Seriation (R package)

The task of reordering a set of elements on the basis of their similarity or dissimilarity matrix is known as the *seriation problem* [16] and can be seen as a very specific case of matrix reordering. Given a set of n objects where $\mathbf{D}_{i,j}$ represents the dissimilarity between i and j, the Seriation problem consists in finding a permutation $\pi \in \mathcal{P}_n$ that minimises or maximizes a loss/merit function L. Based on the equations 2.2 and 2.3 we can write:

$$\underset{\pi \in \mathcal{P}_n}{\operatorname{argmin}}\, L(\mathbf{D}^{\pi \times \pi}) \tag{2.5}$$

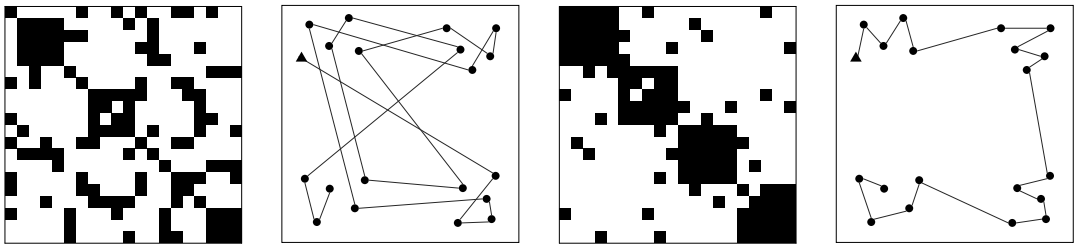
The seriation library in R offers a wide range of criteria, methods and heuristics. [17] provides a survey that details and compares these different methods.

Travelling salesman problem

The advantage of dissimilarity matrices is that they allow the reordering problem to be viewed from different perspectives. One way of solving it is to consider it as a TSP (*traveling salesman problems*) instance [18]. The TSP is an optimization problem that given a list of cities, and distances between all pairs of cities (distance matrix), seeks to determine a shortest route that visits each city once and only once. Since we already have a distance matrix for the rows/columns one simple way to reorder the matrix is to enter $\mathbf{D}$ as input into a TSP solver (e.g. 2-opt algorithm) and use the output route as new order. More formally, the criterion to minimize is the following:

$$\sum_{i=1}^{n-1} \mathbf{D}_{\pi(i),\pi(i+1)} \tag{2.6}$$

A difference with the original problem is that we visit all the "cities" without forming a cycle. For that we usually introduce an additional dummy null column/row to the distance matrix. Figure 2.6 shows an example of TSP (2-opt) ordering applied



(a) (b) (c) (d)

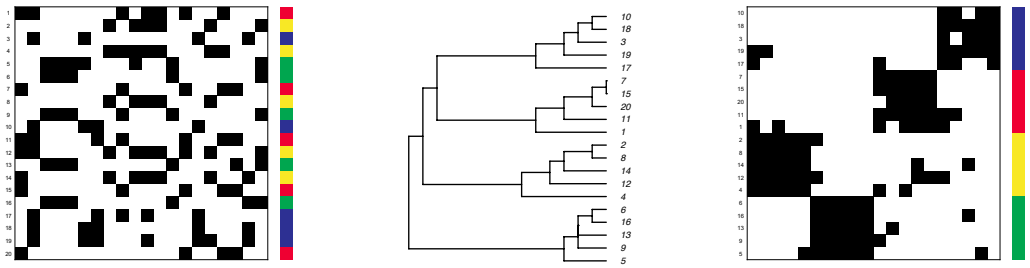
Figure 2.6: TSP ordering example

on a one-mode matrix. (a-c) show the matrix before and after the ordering; (b-d) show a 2D analogical representation of the TSP problem by representing in the plane what the path would look like.

Hierarchical clustering

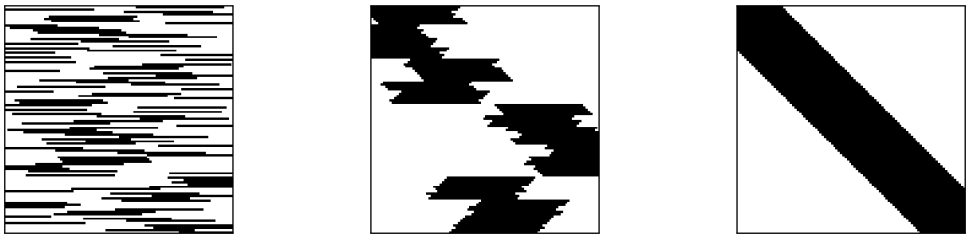
One of the most used methods in data science and bioinformatics for gene expression visualization is arguably *hierarchical clustering* (HC) [19]. It is a clustering algorithm (the task of grouping a set of objects in groups based on their similarity). The particularity of this technique is that it produces and orders the clusters in a hierarchical way meaning that each cluster contains sub-clusters or is part of a larger one. The output of this kind of method is usually a dendrogram (Figure 2.7b), i.e. a tree showing the division into sub-clusters.

This kind of method is easily usable to reorder a matrix because it only needs a distance matrix. To reorder the data it only requires to apply the order of the dendrogram on the matrix. When a dendrogram is constructed an order is also generated. An application of this algorithm is shown in Figure 2.7; the columns



(a) Unordered table (b) Dendrogram of rows (c) Ordered table

Figure 2.7: Example of a dendrogram used for hierarchical clustering



(a) Random rows (b) Hierarchical clustering (c) Optimal leaf ordering

Figure 2.8: Example of optimal leaf ordering

and rows (two independent problems since 2-mode data) are reordered using the dendrogram.

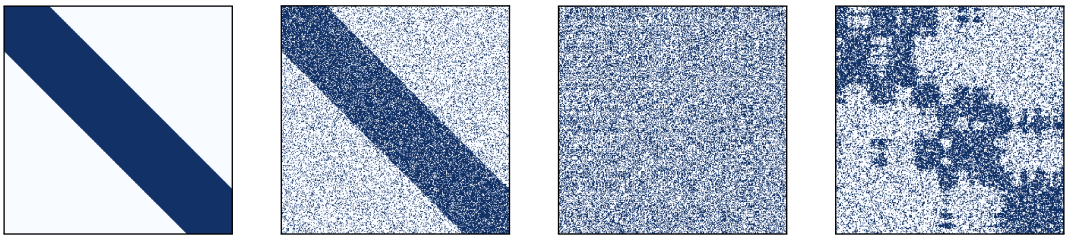
An interesting observation in this example is that the method succeeds in correctly finding the clusters. However, they are not aligned on the diagonal. Distance-based methods would have no reason to select one order for the clusters over another.

Dendrogram ordering

Hierarchical clustering, although very efficient, has several limitations. The method is efficient on data where the pattern forms clear blocks but does not work on matrices with a non-rectangular pattern (banded for example). It is nevertheless possible to improve hierachical clustering through methods called *dendrogram seriation* or *leaf ordering*.

A dendrogram composed of n objects has $n - 1$ internal nodes. Knowing that for each node we can swap the left and right sub-trees, there exists 2^{n-1} ways to reorder the leaves and nodes of a dendrogram. The so-called leaf ordering methods thus aim at starting from an initial solution produced by HC and improving it by rearranging the structure of the dendrogram.

We distinguish two methods. Each of them starts from the dendrogram produced by a HC and modifies its structure. Gruvaeus and Wainer [20] propose a method (GW) that traverses all internal nodes of the tree and permutes the branches so that the adjacent sub-trees are the most similar. Bar-Joseph et al. [21] propose an Optimal Leaf Ordering (OLO) that tries to maximize the similarity between each adjacent element in the order. In other words, it minimizes the *Hamiltonian path* length criterion already described in equation 2.6. Figure 2.8 shows an example of a banded dataset where only the rows are shuffled (a). (b) shows the order produced by HC and (c) shows the result after applying OLO.



(a) original (b) 25% noise (c) Random order (d) Reconstructed

Figure 2.9: Effect of noise on the ordering

2.3.3 Convolution-based

In recent works, a new type of reordering (ConvoMap) was proposed for binary matrices. It is based on convolution [1, 2]. This method relies on an *evaluation criterion* that measures the quality of an order by computing the difference between the image in this order and a blurred version of the same image.

In Chapter 3, we will detail the notion of *evaluation criterion* and state the theoretical basis related to *convolution*. We will then briefly present the convolution-based algorithm and state its weaknesses.

2.4 Limitations

The methods presented above are particularly effective for reordering matrices where the pattern (if such a one exists) is clearly identifiable; these data are generally "noise-free".

However, assume we generate (Figure 2.9) for example (a) an artificial binary pattern (a band for example); we (b) introduce noise with a probability p (for $p\%$ of the entries we decide to reverse the value); and we (c) randomize the order of columns and rows. It turns out that the traditional methods, both structure-based and distance-based are unable to (d) reconstruct the basic pattern. The *TSP* method is used for this particular case. The new order produces a pattern with some areas denser than others but the shape is too distorted for any analysis to be made. A similar problem is encountered for sparse data. We will evaluate in Chapter 5 (Experiments) all presented methods on different artificial and real data sets.

CHAPTER 3

Evaluation criterion

In this chapter, we will discuss how to assess the quality of a particular ordered matrix and more generally how to compare two different orders of the same matrix. We will show the challenge of formalizing such a *criterion* and what is its usefulness. We will then study how *convolution*, a popular image processing technique, can be used to define a generic evaluation criterion. Finally, we will sketch how this criterion could be used as an objective function to create a brand new reordering algorithm. A large part of the material of this chapter is broadly adapted from the previous work of T. Bollen, G. Leurquin and S. Nijssen [2].

3.1 Criterion

Let's assume a matrix $\mathbf{M}$ that we desire to reorder and a set of different reordering methods $\{\varphi_1, ..., \varphi_k\}$. We consider any reordering method φ as a kind of *blackbox* (TSP for example) which takes as input a matrix and returns the same matrix but with the columns and rows permuted $\mathbf{M}' \leftarrow \varphi(\mathbf{M})$. The implementation of this method and the criterion it optimizes is hidden and does not matter.

We reorder the matrix with each of the methods in the set, thus obtaining a set of different ordered matrices $\{\mathbf{M'}_1, ..., \mathbf{M'}_k\}$. The question we ask is: how to determine the best order among all these matrices? To do this, we need to define a score or "*evaluation criterion*" that would judge the quality of each matrix. The lower the score, the better the order is judged.

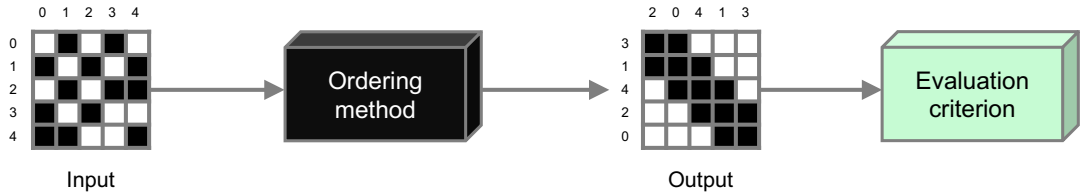


Figure 3.1: Reordering method and criterion as a black box

Formalizing such a criterion is complex because it should be as universal as possible and applicable on all possible data types. As example, [8] describes a "nestedness" measure for nested patterns, [12] proposes a criterion that measures how well a matrix is "fully banded". Such a criterion based on a particular pattern is proscribed. We have seen that distance-based methods and the seriation problem seek to minimize a specific function L. This kind of function could be used as a criterion but it is too specific to the method. Each method optimizes its own criterion, moreover the 2-mode data are generally reduced to 2 independent problems for rows and columns.

A good evaluation criterion should be able to measure the "quality" of the patterns, how well they are defined. A matrix with only noise should have a very high (bad) score. On the other hand, an image where denser areas can be clearly distinguished from others with a well defined shape should have a lower (better) score.

One solution is to treat and process the matrix as a whole picture and to use an image processing technique called *convolution*. We will describe in the next sections in detail the convolution process, as well as how it can be used as evaluation criterion.

3.2 Convolution

From now on, we will consider any matrix or data table $\mathbf{M}$ of size $m \times n$ as a picture with coordinate (r, c) that starts at the upper left corner $(1, 1)$. We will present convolution in the context of binary data but we will see in Chapter 4 how it can be adapted for numerical or categorical data as well. Each entry of the matrix is modeled by a pixel, white if 0 and another color if 1 (usually blue or black).

Convolution is a mathematical operation that is used in many fields, like signal processing, or deep learning (convolutional neural networks). We will focus on convolution in the context of image processing.

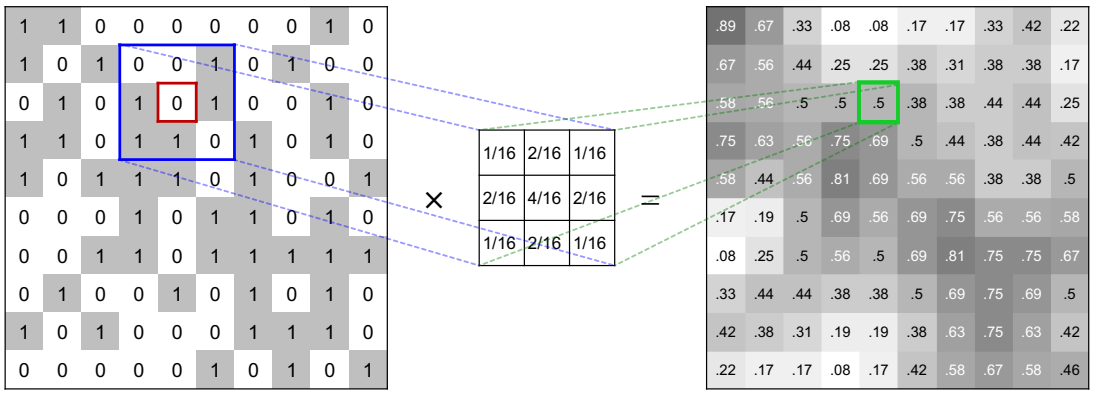


Figure 3.2: Example of convolution with an exponential kernel (3×3)

The principle consists in modifying or filtering a matrix $\mathbf{M}$ by using a second matrix $\mathbf{K}$ called Kernel which will act as a filter. The convolution is an operation between two functions (here two matrices) which produces a 3rd one.

For each element of the matrix $\mathbf{M}$, we sum the surrounding elements (neighbors) multiplied by a specific weight which is modeled by the Kernel. The whole matrix is processed by sliding the kernel over all the pixels of the matrix (row by row for example). By applying such a process, the matrix will be modified or filtered, each element being somehow averaged by the neighboring elements. An example of convolution is shown in Figure 3.2. It should be noted that the effect of convolution depends on which kernel is used (next section).

Given a matrix $\mathbf{M}$ of size $m \times n$ and a kernel $\mathbf{K}$ of size $k_m \times k_n$, we define the convolution as follows:

$$(\mathbf{M} * \mathbf{K})_{i,j} = \sum_{r=1}^{k_m} \sum_{c=1}^{k_n} \mathbf{K}_{r,c} \cdot \mathbf{M}_{i+r-\lceil k_m/2 \rceil, j+c-\lceil k_n/2 \rceil}. \tag{3.1}$$

3.2.1 Kernels

Depending on the kernel used, the effect on the matrix can vary greatly. Some have the effect of sharpening the matrix, others are used to detect the edges. In

identity $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ shift $\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ box blur $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

linear $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ exponential $\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$ cross $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

Figure 3.3: Example of 3×3 kernels

this work we will focus on the case of **blurring** kernels.

Some examples of kernels are shown in Figure 3.3. The *identity* kernel is one of the simplest consisting of all 0s except a 1 in the middle. Applying this kernel has no effect on the matrix. A variation would be to put the single 1 element at another location, it has the effect of *shifting* the matrix. A 3×3 kernel with only 1's is called *box blur* or *uniform* kernel, it computes the arithmetic mean of the entire surrounding for each pixels. This type of blurring is the simplest possible. We will define 3 types of blur that we will use in the continuation of this work: *exponential*, *linear* and *cross*.

We consider only kernels $\mathbf{K}$ $(k_m \times k_n)$ which have an odd size and are symmetric. We define: $c_m = \lfloor k_m/2 \rfloor$, $c_n = \lfloor k_n/2 \rfloor$ and $mid = max(c_m, c_n)$.

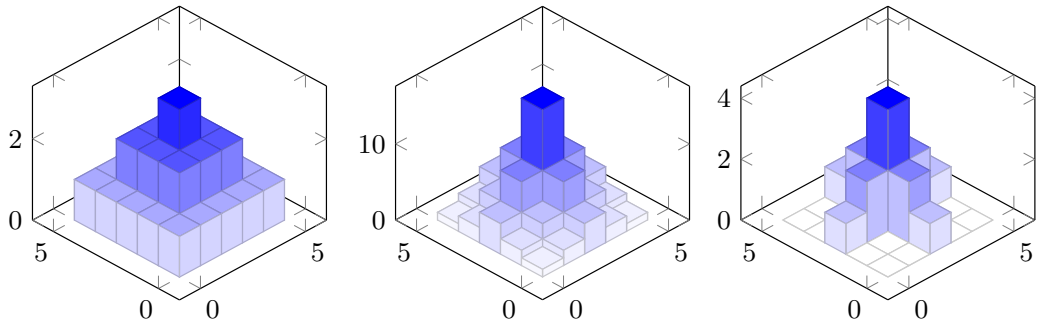
Linear kernel

This kernel produces a blur often described as smooth. The weights decrease linearly with respect to the distance from the center. It is very efficient to remove noise from an image. However it tends not to preserve the shape of the patterns. A rectangle has its corners rounded for example. It is defined as follows:

$$\mathbf{K}_{i,j} = mid - max\Big(|c_m - i - 1|,\, |c_n - j - 1|\Big) \tag{3.2}$$

Exponential kernel

Unlike the linear kernel, this one tends to blur the picture while preserving the shape and direction of the edges. The central pixel still receives the highest weight but the other pixels follow an exponential function. This kernel tends to approximate another type of blur called *Gaussian*, often used in image processing.



(a) Linear kernel (b) Exponential kernel (c) Cross kernel

Figure 3.4: 3D representation of kernels weights (5×5)

The exponential kernel is however easier to generate because it does not require to compute a discrete approximation of a 2D Gaussian function. It is defined as follows:

$$\mathbf{K}_{i,j} = \frac{2^{2.mid}}{2^{|i-1-c_m|+|j-1-c_n|}} \tag{3.3}$$

Cross kernel

In some specific cases, one only wants to take into account the pixels aligned on the column and the row. The diagonal neighboring pixels must be ignored, such matrices where every value that is not on the middle row or column is equal to 0 are called "cross" kernel.

Normalization

By applying a blurring convolution on binary data, we transform a discrete matrix (booleans) into a continuous one: $\mathbf{N} \leftarrow \mathbf{M} * \mathbf{K}$ with $\mathbf{N}_{i,j} \in [0, 1]$. Each value of the new matrix must fall in the interval between 0 and 1. To do this, the kernel is normalized, meaning that the sum of the values is equal to 1.

$$\mathbf{K}_{normalized} = \frac{\mathbf{K}}{\sum_{r=1}^{k_m} \sum_{c=1}^{k_n} \mathbf{K}_{r,c}} \tag{3.4}$$

Padding

An important aspect of convolution is the management of the borders when the kernel exceeds the edges. For some pixels, the kernel is superposed with slots

outside the matrix. We need to give a replacement value to these off pixels. There are many types of possible padding, for example *extension* i.e. assigning the last value before the edge or *cropping* which consists in replacing these missing values by 0. For the sake of efficiency, these pixels will simply be ignored, the Kernel will only be normalized by the values being overlapped with the initial matrix.

3.3 Criterion based on convolution

Now that the theoretical foundations of convolution are laid, we will show how this technique can be applied as a criterion to measure the quality of an ordered matrix.

As explained before, we want to formalize a criterion that analyzes an ordered matrix and gives us a score that tells us how well ordered the matrix is. Are the patterns clearly defined? Are the edges straight? Is the noise low? Is the number of 0-1 transitions minimized? All these questions are judged in a subjective way and left to the appreciation of the user but thanks to convolution and more particularly to blurring we can automate this process.

The key idea is to measure the difference between the ordered matrix and the same matrix after applying a blur. It is precisely this difference that is used as an evaluation criterion. We define the difference or error between two matrices in the following way:

$$error(\mathbf{M},\, \mathbf{M}') = \sum_{r=1}^{m} \sum_{c=1}^{n} \left| \mathbf{M}_{r,c} - \mathbf{M}'_{r,c} \right| \tag{3.5}$$

For each entry the absolute difference between the two matrices is measured, the error is the sum of these values. The evaluation criterion is therefore simply written as such:

$$error(\mathbf{M},\, \mathbf{M} * \mathbf{K}) \tag{3.6}$$

The lower the score, the better the order is. The intuition is that if applying blurring to an image does not change it significantly, it means that the original image already has a pattern with a clear structure with little transition $0 - 1$. A concrete example of this criterion is shown in Figure 3.5. The images on the left represent the same matrix (300×300) with different orders. The middle ones represent these matrices after applying a convolution with the linear blurring kernel (49×49). The right-hand images show the difference between the original and the blurred matrix. The reddish colour symbolises a higher error. The score for the first matrix (1^{st} row) is evaluated at 35.135, the second (2^{nd} row) at 37.728.

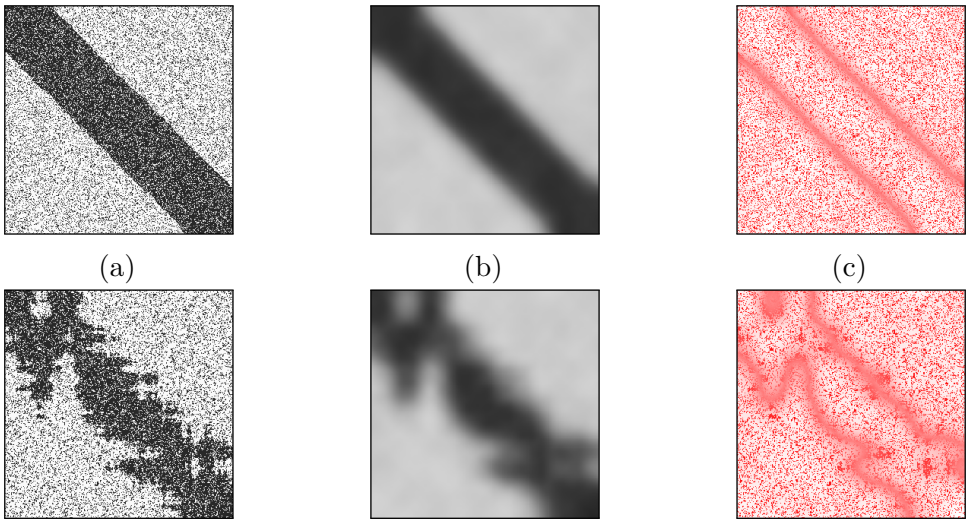


Figure 3.5: Convolution evaluation criterion example

Various findings can be drawn from Figure 3.5. By measuring the difference (red color) between a binary matrix and its convolution we notice that the criterion indirectly (*1*) detects the boundaries of the patterns and penalizes them (*2*) detects the background noise (random 0-1) (*3*) penalizes the imperfections (curved contours, dislocated pattern). The criterion is just a sum of all these anomalies.

3.4 Convolution ordering

We have seen in Chapter 2, that the existing methods can be divided into two main families (*distance* and *structure based* methods). We have also stated that these methods although efficient in simple cases with basic patterns are unable to correctly reorder when random noise is introduced.

Bollen and Leurquin [2] have proposed an algorithm called ConvoMap that addresses this weakness. They use the convolution based evaluation criterion that was presented in the previous section as an objective function through the implementation of a *local search* algorithm. We will give the main lines of this algorithm.

The method consists in starting from a partially ordered matrix called *candidate solution* (for example the output of another reordering algorithm, typically TSP). We define a set of operations called *moves* that can be applied on the initial solution in order to modify and improve it. After applying a move, the candi-

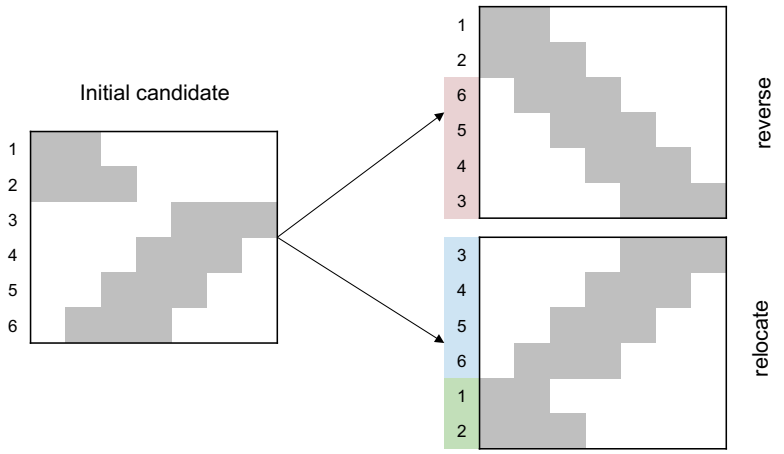


Figure 3.6: Initial candidate and neighborhood

date's score is updated. The set of candidates generated from an initial solution is defined as the *neighborhood*. The local search algorithm consists in generating this neighborhood and selecting a better solution that improves a scoring or objective function which in this specific case corresponds to the evaluation criterion.

In other words, the algorithm starts with a partially ordered matrix and tries to improve it with respect to the evaluation criterion by applying different transformations. 3 examples of "moves" are illustrated in figure 3.7. These operations target the order either of the rows or of the columns.

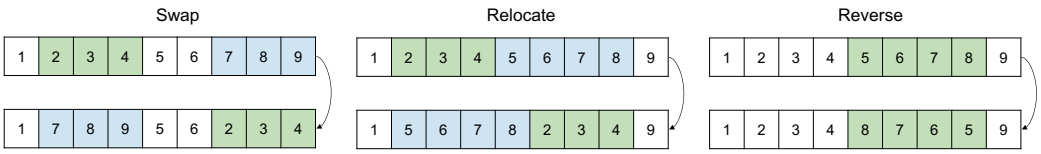


Figure 3.7: Local search moves applied on the order

We will see in the experimental part of the thesis that the ConvoMap approach is rather effective in reordering binary matrices with noise (up to 30%) while the other methods fail. A major issue of the algorithm is however the execution time (multiple hours have been reported for a 300×300 matrix). Indeed, a characteristic of ConvoMap is that the moves are randomly generated (random walk hill climbing) and no efficient algorithm is known to find an order that optimizes this evaluation criterion.

CHAPTER 4

Iterative ordering framework

In Chapter 2, we presented the matrix reordering problem and discussed the different methods existing in the literature. These methods, although very fast and commonly used for data visualization, suffer from several issues. Notably they are not robust enough to noise. Introducing even a small amount of noise seems to strongly alter the ordering and make it impossible to extract a coherent pattern. Another problem concerns the so-called sparse matrices, real world data (generally graphs) are often in this form and once again the existing methods fail to find a coherent order. We will study in the Experimental part of the thesis these problems in more details on both real and artificial data.

We have introduced in Chapter 3 the notion of evaluation criterion allowing to assess how well a matrix is ordered. In addition to that, we presented an algorithm which uses this criterion and allows to reorder the matrices despite the issues described previously. Unfortunately this randomized algorithm is not optimized and is not applicable in settings that require fast visualization.

In this chapter we introduce a new approach to reorder matrices regardless of the degree of noise, sparsity or size. Despite being based on convolution, we have taken a completely different approach. Instead of creating a new algorithm from scratch similar to ConvoMap, we propose a framework that allows to reuse any reordering method as a black box component and to augment it by taking advantage of convolution.

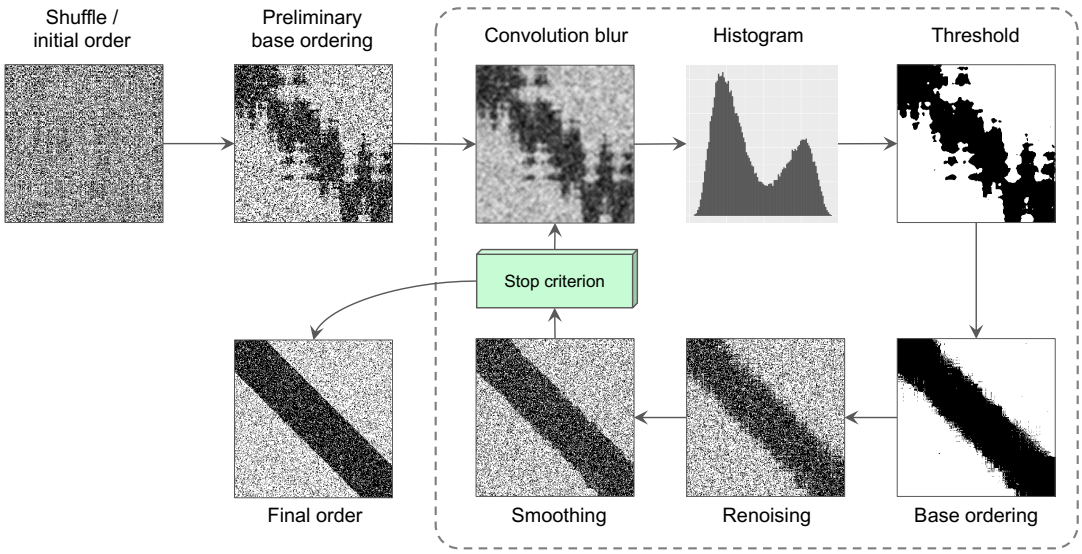


Figure 4.1: Overview of the iterative framework with thresholding enabled

4.1 Overview

The key idea of the *iterative framework* is that we embed a *base ordering method* φ_b in an iterative loop to make it more robust to noise through convolution in order to obtain a better output ordering. This is similar in spirit to how boosting methods used in machine learning improve the classification accuracy of an underlying base learning algorithm.

Convolution is used for two different purposes: on the one hand as an evaluation criterion to assess the quality of an order, on the other hand to temporally transform and simplify the matrix by reducing/removing the noise.

Before describing the approach in its global picture it is necessary to specify two assumptions about the underlying base ordering method φ_b:

1. on data with a perfect, noiseless structure (such as a banded structure, a nested structure, a block structure, and so on), the ordering method will recover this structure;

2. on noisy data the ordering method will recover some of the structure in the data, although in an imperfect manner.

We will refer to the imperfect, incomplete patterns identified by the base method as *embryonic pattern*. An example of an embryonic pattern is presented in Figure

4.1 (TSP used as base ordering). We will show in the experimental part (Chapter 5) that most existing methods in literature are able to generate these fragments of dense patterns.

Figure 4.1 shows an overview of the iterative approach. Initially, the base method φ_b is used to produce a preliminary order (embryonic pattern). Subsequently, a *convolution* and a *threshold* is applied to blur and remove noise from the image. The resulting data is once more put in the base ordering method, exploiting the assumption that for noise-free data the method will produce a better ordering. The resulting order is applied to the original data (*renoising*), after which the process is repeated until a *stopping criterion* indicates that the process has converged. The process is optimised by adding an additional *smoothing* refinement step which consists in a fast local search that locally improve the solution. We will now describe in detail the entire architecture of the iterative framework and its respective components.

4.2 Architecture

4.2.1 Preliminary ordering

The purpose of the framework is to improve a basic reordering algorithm φ_b by making it more robust to noise. In Chapter 2 we presented different methods and detailed their specificity. Here in the context of this framework these methods are seen as black boxes, components that rearrange the order of the matrix. Their internal implementation does not matter.

The first step is to start from the initial (raw, unordered) matrix and to reorder it with the chosen method that we want to enhance. It is from the output of this particular method that the framework will try to improve the order with respect to an evaluation criterion.

4.2.2 Blurring

In image processing, the blurring convolution is frequently used to remove noise from a picture. Indeed, by averaging the pixels with their neighbourhood, the image is more uniform. Let's assume a matrix particularly noisy with strong variation of $0-1$. One area of the picture is composed of 30 % of 1 and 70 % of 0. By applying a Gaussian convolution we obtain an average value of 0.3 for this area. The convolution allows to remove the noise from the image while keeping the shape

of the pattern intact.

It is for this very purpose that the convolution process is used in order to remove noise from the matrix and obtain a simpler model to reorder. A question that arises is: which kernel should be used for this process? In Chapter 3, we formalized 3 types of blurring kernels.

Since we do not know the data in advance and their degree of noise, we propose to use a set κ, a sequence of kernels used for the convolution process. This set is parameterized and given by the user and should be composed of kernels of different types and sizes in order to diversify and intensify the blur that we apply. We will see later how this set is reordered at each iteration.

$$\kappa = \left\{ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} , \dots , \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \right\}$$

Figure 4.2: Example of kernels sequence κ used for the blurring

4.2.3 Thresholding

After applying the blur convolution, the initial matrix is transformed from a binary one into a continuous one $\mathbf{N}$ with $\mathbf{N}_{i,j} \in [0, 1]$. Two strategies are therefore possible at this point. The first one consists in directly reordering the convoluted matrix with the base method. This is only possible for *distance-based* methods. The second option consists in reconverting the blurred matrix into a binary one by applying a *thresholding* operation. For this reason, the following step is **optional**. We will discuss in the experimentation part which cases require a thresholding and what is the impact of this step on the final result. The *threshold* of a Matrix $\mathbf{N}$ is defined by :

$$threshold(\mathbf{N}, t)_{i,j} = \begin{cases} 1 & \text{if } \mathbf{N}_{i,j} > t \\ 0 & \text{if } \mathbf{N}_{i,j} \leq t \end{cases} \tag{4.1}$$

This filter consists in choosing a threshold $t \in [0, 1]$, which is used to partition the values of the matrix into two classes. There exists different methods described in the literature.

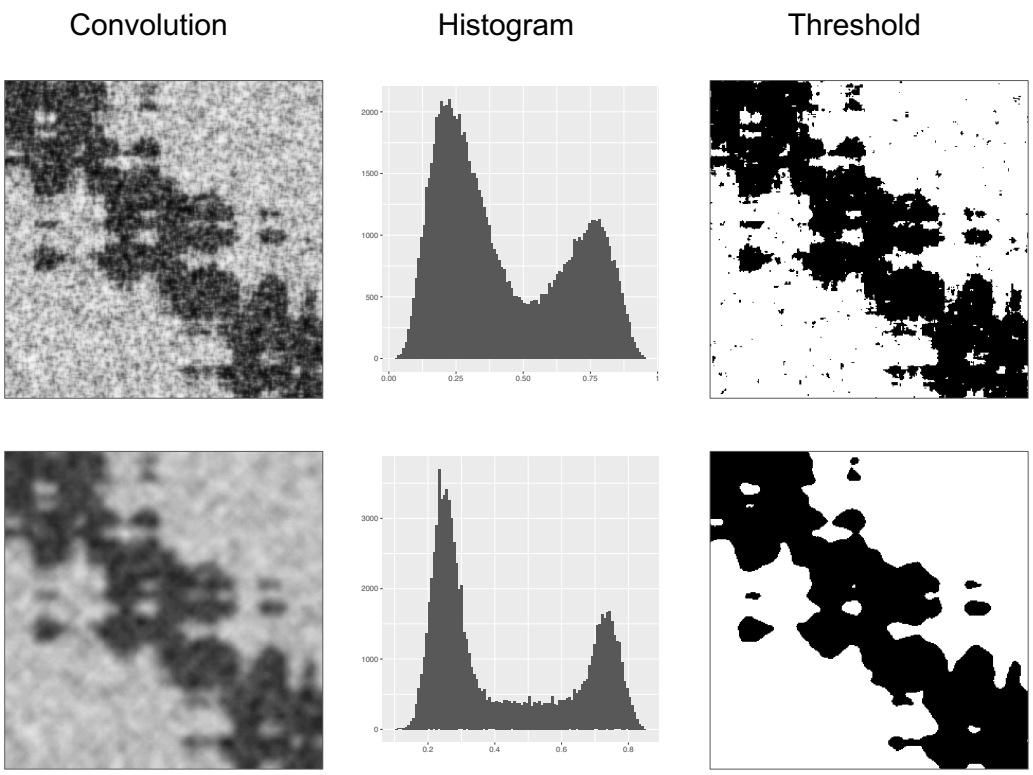


Figure 4.3: Histogram and thresholding

OTSU method

To achieve this, the Otsu method is used. This is a popular algorithm [22, 23] that constructs a histogram of the intensities of the matrix values and then partitions it into two classes in order to minimize their intra-class variance. This method is based on the shape of the histogram and makes the assumption that the distribution of the values is modeled by a bi-modal distribution. This assumption is partially satisfied if the original image contains a well defined pattern divided into different dense areas.

Figure 4.3 shows the same sub-optimal pattern (obtained with TSP) to which we applied two different kernels. Respectively an exponential kernel for the first line and a linear kernel for the second; both with a size of 15×15. We can easily notice that the distribution of the values follows a bi-modal trend. However, as the linear kernel removes the noise in a more intense way, the distribution is more polarized. Another finding of this analysis is that the exponential kernel also tends to preserve

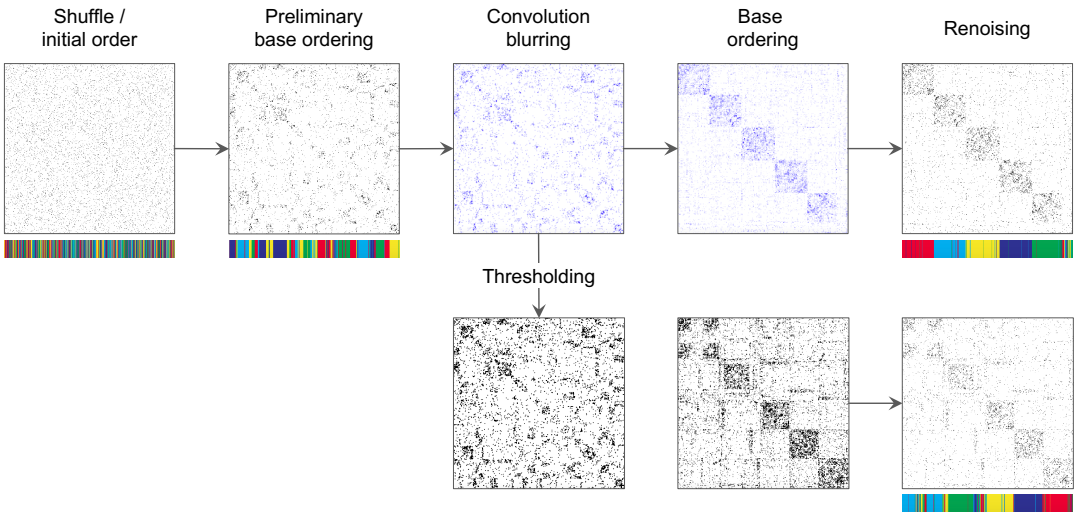


Figure 4.4: Iterated ordering with and without threshold

the shape of the contours better but does not totally remove the noise.

4.2.4 Iterated base ordering

Once the convolution and optionally the threshold are applied, a noiseless binary matrix is obtained, which contains only the shape of the *embryonic pattern*. It is then easier to reorder this new simpler matrix. The embedded base algorithm φ_b is then executed. We assume that we use the same and unique method all the time for both the preliminary and iterative order. We will discuss later that we can use 2 different methods for example.

It should be noted that there are two possible scenarios depending on the previous step. Either we reorder of a binary matrix. In this case we will use the Hamming distance if φ_b is a *distance-based* method. In the other case we reorder a numerical matrix $[0, 1]$. *Distance-based* (mandatory) will used the Euclidean distance. A concrete example is shown in Figure 4.4. We use φ_b = Hierachical clustering. Once the preliminary ordering is done we can either directly reorder the continuous matrix or re-binarize it with a thresholding.

The produced output order of rows and columns is then applied to the initial matrix. This operation is called *renoising*.

4.2.5 Smoothing refinement

Thanks to this successive process of convolution, ordering and denoising, we succeed in reordering a noisy or sparse matrix with a method that initially was ineffective to manage with such data. Figure 4.1 clearly shows that the preliminary (embryonic) pattern is improved in a much clearer banded structure. However, some defects may remain, notably for the quality of the edges. To address this issue, we apply a refinement operation that we refer to as *smoothing* (Figure 4.5).

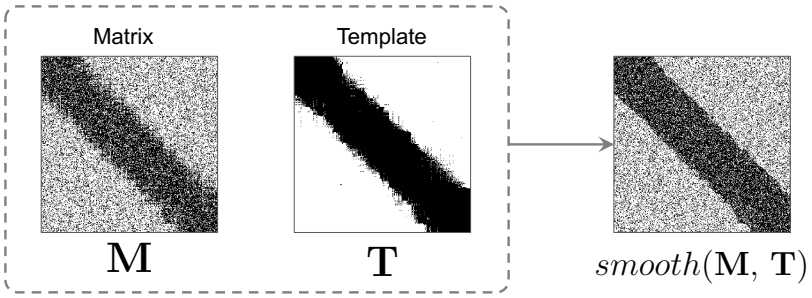


Figure 4.5: Example of smoothing

The key idea of this step is to slightly permute the denoised matrix so that it looks as similar as possible to the thresholded version. We define the algorithm $smooth(\mathbf{M}, \mathbf{T})$ which takes as input 2 matrices of the same size but can be of different types (one binary, the other numerical). The matrix $\mathbf{M}$ is the one we want to reorder and the matrix $\mathbf{T}$ called "*template*" is the model used to reorder $\mathbf{M}$. This method permutes the rows and columns of $\mathbf{M}$ so that the $error(\mathbf{M}, \mathbf{T})$ is minimized. $\mathbf{M}$ is the denoised matrix and $\mathbf{T}$ is the ordered thresholded or convulated matrix just before denoising. The pseudo-code of this procedure is given in Algorithm 1.

Algorithm 1: Smoothing

input : matrix $\mathbf{M}$ $(m \times n)$ to reorder, Template $\mathbf{T}$ $(m \times n)$

```
repeat
    changed ← false
    h ← height(M)
    for i ← 1 to h − 1 do
        for j ← 2 to h do
            if d(M_i, T_j) + d(M_j, T_i) < d(M_i, T_i) + d(M_j, T_j) then
                Swap(M_i, M_j)
                changed ← true
    M ← M^t, T ← T^t
until ¬changed;
return M
```

The algorithm can be seen as a local search in the same spirit as the 2-*opt* algorithm. We try to swap the columns/rows pair to pair by calculating the gain/delta that we would obtain by exchanging them. We define the function $d(x, x')$ as the Manhattan/Hamming distance. A toy example is shown in Figure 4.6. We observe that the algorithm is, at this specific instant of the execution, on rows 2 and 5 (i and j). It will swap these two rows because the gain is negative ($\Delta = -8$). After this move, the order of the matrix is improved with respect to the template.

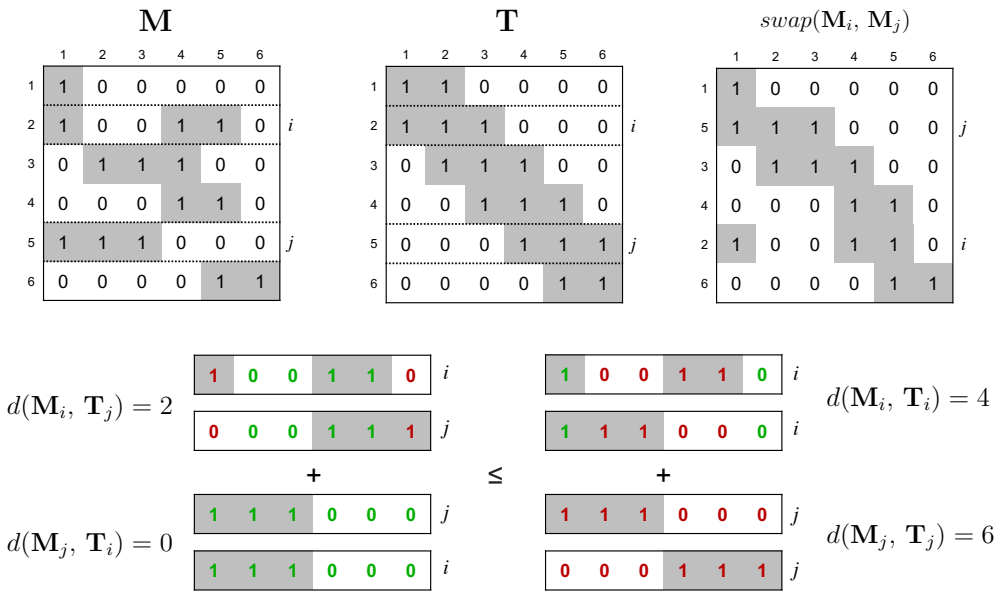


Figure 4.6: Smoothing: example of row permutations

4.2.6 Stopping criterion

The transformations described above are repeated iteratively. For each iteration, a new candidate ordered matrice is obtained. The challenge is to measure the quality of this candidate in order to verify if it improves the last one. For this we simply reuse the ConvoMap evaluation criterion which is based on convolution ($error(\mathbf{M}, \mathbf{M} * \mathbf{K}_c)$). The user only has to define a kernel $\mathbf{K}_c$ which will be used as stopping criterion. It is important to note that this unique kernel used for convolution is different from the set κ we defined to blur the image. In short, the convolution is used for 2 **different purposes**: the evaluation/stopping criterion and the blurring/thresholding process.

4.2.7 Parametrization

It remains now to determine how the framework is tuned. It takes as input a matrix $\mathbf{M}$ (binary but we will see how to easily adapt to numerical or categorical types). The framework integrates a unique ordering method (φ_b) and an evaluation criterion based on a kernel $\mathbf{K}_c$. The user also provides a kernel sequence κ which is used to blur the matrix. There are two additional boolean flags parameters to enable or disable *thresholding* and *smoothing*. Finally there is also a parameter to specify whether the matrix is *one-mode* or *two-mode*. A simplified pseudo-code is provided in Algorithm 2.

Algorithm 2: Framework

```
   input : M, K_c, κ, φ_b
 1 M ← φ_b(M)
 2 repeat
 3     e_0 ← error(M, M ∗ K_c)
 4     for ∀K_i ∈ κ do
 5         N ← M ∗ K_i
 6         T ← threshold(N)
 7         M' ← φ_b(T) or φ_b(N)
 8         S ← smooth(M, M')
 9         e_1 ← error(S, S ∗ K_c)
10         if e_1 < e_0 then
11             M ← S
12             break
13 until e0 ≤ e1;
14 return M
```

Kernel replacement

An important aspect that remains to be developed is which kernel is selected from the κ set and how is this sequence reordered? At each iteration, a new candidate matrix is produced. The evaluation criterion checks whether the score of this new order improves the previous one. If there is an improvement, the execution continues with it and starts a new cycle of convolution, thresholding, reordering and so on. If there is no improvement, the candidate is discarded and the algorithm starts again with a new kernel (the next one in κ) to produce a more pronounced blur for example. The execution ends eventually if all kernels fail to improve the last solution.

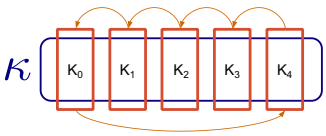


Figure 4.7: Kernel replacement policy

In this work and for the sake of simplicity, the κ sequence is not reordered to make the results more predictable (see Experimental Chapter). However, a strategy (which we refer to as *kernel replacement policy*) would be to continuously reorder the set by penalizing for example the inefficient kernels (with a system of points). We will discuss these possibilities in the future work section.

4.2.8 Summary

The figure below (4.8) shows a big picture of the framework. This architecture can be qualified as a framework because the user can integrate any reordering method like a *template-method* (design pattern), implementing various behaviors but following a strict specification which consists in reordering a matrix. The evaluation criterion can also be seen as an interchangeable component.

Another remark is that the proposed approach is in fact a local-search algorithm. We start from a sub-optimal solution (preliminary/embryonic ordering) which is improved by means of "moves" that are characterized by a combination of convolution, ordering and smoothing. The stopping criterion corresponds to the scoring function, making the approach not so different from ConvoMap. However, the framework is faster because it is not randomized. It takes advantage of existing methods which are very optimized and fast and run them a limited number of times. The algorithm converges in two minutes where ConvoMap takes several hours for the same instance.

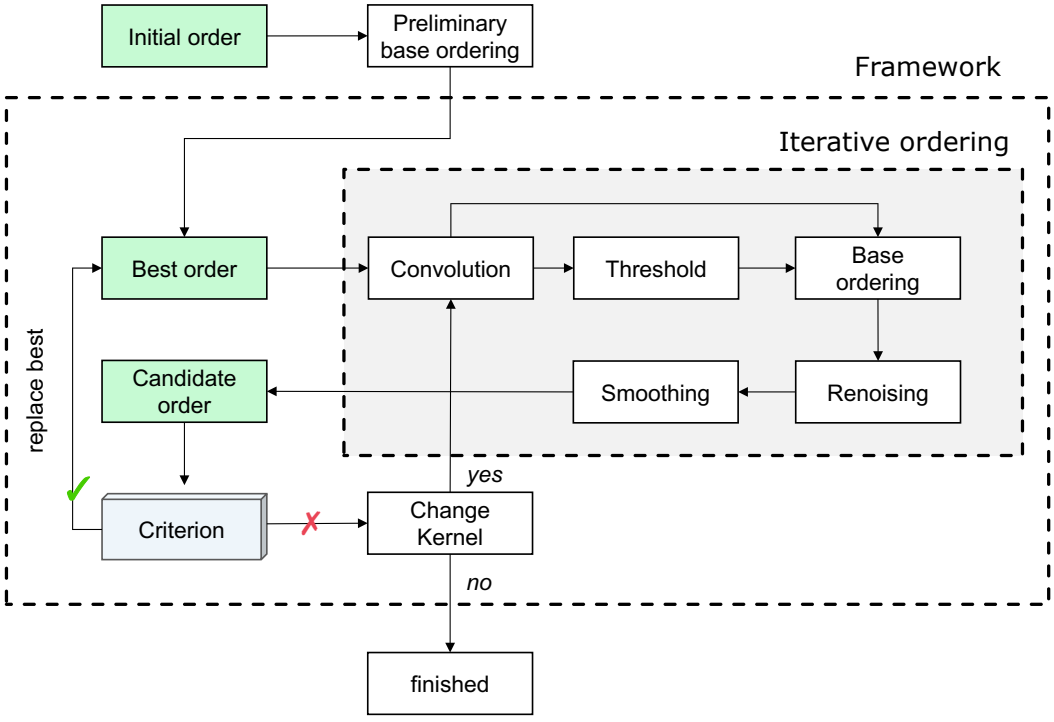


Figure 4.8: The framework in a nutshell.

4.3 Adaptation

We have presented in Chapter 3 and 4 the convolution process and the framework in the context of binary matrices. We will now briefly explain how convolution and the evaluation criterion can be adapted in order to deal with numerical and categorical matrices.

4.3.1 Numerical data

Many datasets are characterized by numerical attributes. It is necessary to ask how to adapt the convolution process. One of the main challenges with this kind of data is that the range of these attributes can vary greatly. Another question is: do we have to deal with *integers* or real (*floating-points*) numbers. Are these values negative?

A first important element to take into consideration is how to manage the so-called *extreme values*? Some datasets are composed of abnormally high (negatively low) values and can strongly disturb the convolution and the evaluation criterion. The display of the pixels according to the color scale also becomes unreadable. It is therefore necessary to impose a threshold to filter these values $[t_{min}, t_{max}]$. In other datasets, some values are missing, annotated as `NaN` (not a number). A strategy must be defined to know what to do with them: should we ignore them or replace them by the neighborhood?

Ideally, the values should be non-negative and included in a well-defined interval in order to make the convolution efficient, for example [0, 255] used to map RGB colors in image processing. It is for this reason that it is preferable to use a "*min-max scaling*" so that all values are between 0 and 1. The framework we have presented works on binary data. However by normalizing the values in this way we can handle any numerical matrix. The only difference is that $\mathbf{M}_{i,j} \in [0, 1]$ instead of $\{0, 1\}$.

$$\mathbf{M}'_{i,j} = \frac{\mathbf{M}_{i,j} - min(\mathbf{M})}{max(\mathbf{M}) - min(\mathbf{M})} \tag{4.2}$$

4.3.2 Categorical data

Defining the convolution and the evaluation criterion for categorical data is more complex. For binary data there are only two values 0 and 1 and for numerical data a continuum between these two. Categorical data do not represent any numerical quantity but labels or categories $\mathbf{M}_{i,j} \in \{c_1, c_2, ..., c_k\}$.

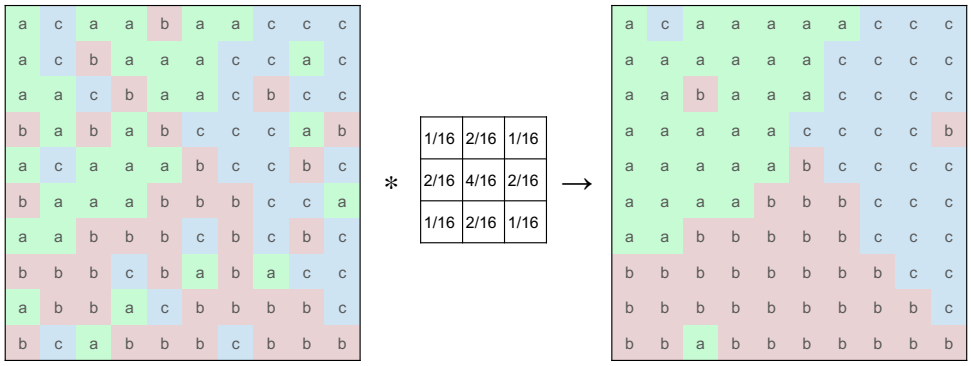


Figure 4.9: Categorical convolution and thresholding

For this reason, we will consider the convolution product of a categorical matrix as a set of several matrices, one for each category as depicted in Figure 4.10. For example if we have the following three labels: $\{a, b, c\}$, we obtain 3 convoluted matrices for each of them. For a given label x, we replace in the original matrix x by 1 and all other categories by 0. Then, we apply the kernel on this new binary matrix. The process is performed on each category.

In the framework, we will only work with thresholding activated. For this, each convoluted matrix is overlaid and the category with the highest intensity is the selected one. This process is illustrated in Figures 4.9 - 4.10. For instance, the pixel in the upper left corner (1,1) has the following convolution values: $a = 0.67$, $b = 0.33$ and $c = 0.0$. Its threshold is therefore a.

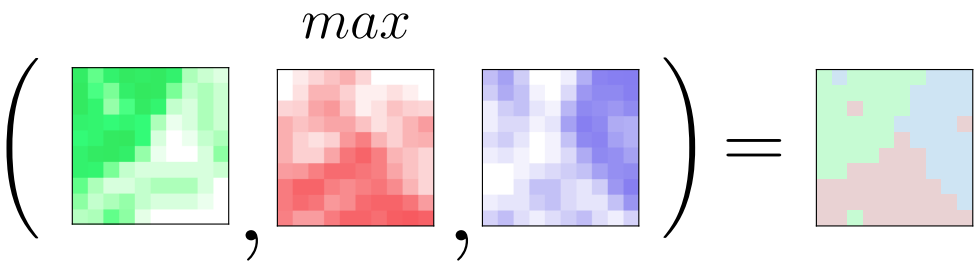
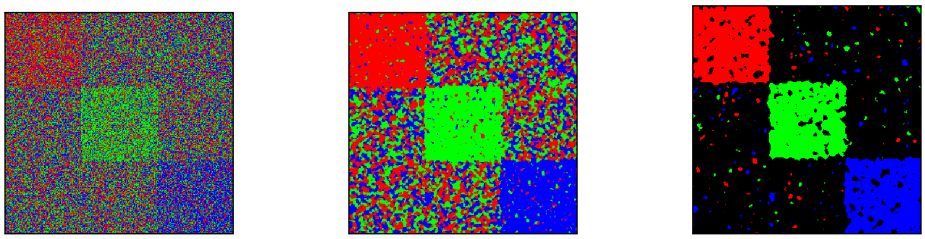


Figure 4.10: Thresholding majority

For the evaluation criterion, we just measure for each pixel $\mathbf{M}_{i,j}$ (sum) the difference $1 - c$ where c is the convolution value for the convoluted matrix associated with the category. The upper left pixel has thus an error of 0.33.



(a) original (b) naive threshold (c) advanced threshold

Figure 4.11: Thresholding comparison

Advanced thresholding: Figure 4.11(b) shows that this naive thresholding is not effective to remove the "background noise" on certain types of matrices. Instead of removing it, it amplifies it. To overcome this issue, we propose a more advanced technique (advanced thresholding) which consists in adding a new category (black in the figure) that models the background noise. This category is assigned to the pixels where the max value is too low. To implement it we reuse the Otsu method.

CHAPTER 5

Experiments

In the previous chapters, we presented a new framework for reordering matrices. In this section, we will conduct different experiments in order to evaluate the performance of the framework and the quality of the results produced for different types of datasets. The objective is to show how the results compared to traditional methods are improved. We also evaluate the influence of the different parameters. The code of our framework is available at: `https://github.com/MasterGauthe/Iterated-matrix-reordering`

5.1 Experimental setup

We configured the framework as follows:

Kernel set (κ): We defined a kernel set containing kernels of size $n \times n$ with $n \in \{3, 5, 7, 9, 15, 25\}$. For each size, we use 2 different types: a linear and an exponential kernel (defined in Chapter 3). They are ordered by decreasing order for tabular data (*two-mode*) and increasing order for networks data (*one-mode*).

Evaluation criterion ($\mathbf{K_c}$): It is important to carefully choose the kernel used as evaluation criterion. As a reminder, this score corresponds to the difference between the matrix and its blurred version. With a too small size, the blur is too local and the error not very informative. A larger kernel is more efficient to measure the noise and give an idea on the quality of the order. Bollen et al. [2] have already shown in their work that the **linear kernel** of size `49` $\times$ `49` is an excellent

compromise as criterion for matrices of size 200 up to 600. We will therefore use this kernel as a criterion unless otherwise specified.

Iterations limit: Instead of setting a timeout on the execution we set the maximum number of iterations to `50`.

Thresholding and smoothing: Except if specified otherwise, *smoothing* is enabled for all our experiments. The *thresholding* is active for tabular data and inactive for network data. We will discuss the reasons for this choice.

Embedded base algorithm (φ_b): Since our implementation can be considered as a white-box framework, any matrix ordering method can be embedded into it. We have chosen to use and benchmark the methods described in Chapter 2. We will use functions from the Seriation [16] and TSP [24] library in R: `TSP` (travelling salesman problem), `HC` (hierachical clustering), `OLO` (optimal leaf ordering), `GW` (Gruvaeus and Wainer) and some dimension reduction techniques such as `MDS` (multidimensional scaling), `QAP` (Quadratic Assignment Problem) and `Spectral`. The `barycentric` and `nested` method will also be used (implementation in java). The ConvoMap algorithm will not be integrated in the framework but used as a comparison for the results produced.

Distance metric (d): The distance-based methods require the creation of one or two dissimilarity matrices. We use the Hamming distance as metric for binary and categorical matrices; the Euclidean distance for numerical matrices.

Linkage: `HC` and other dendrogram-based (`OLO`, `GW`) methods require a linkage. The Complete linkage is used.

5.2 Datasets description

We have seen in Chapter 2 that there are three ways to classify the data. We will attempt to evaluate the framework on a wide range of datasets. We will use both binary, categorical, and numerical data. We will however place greater emphasis on binary data. For the latter we will evaluate both one-mode (networks) and two-mode (tabular) data on artificial matrices but also on real world data.

Binary data (tabular)

We introduced this thesis by presenting different generic patterns (Figure 1.1). We propose 4 different models that feature a specific pattern. These data are slightly

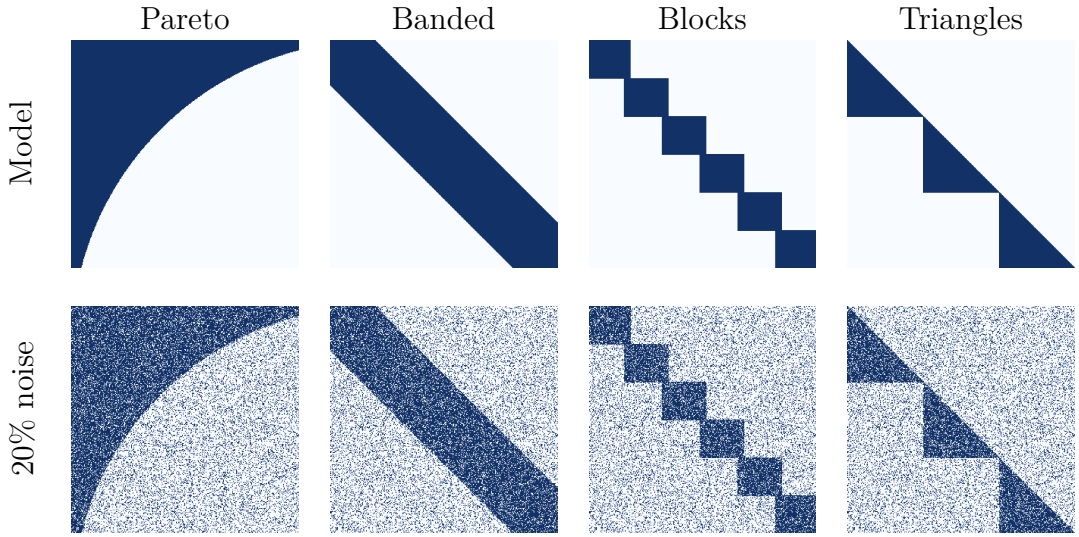


Figure 5.1: Artificial binary matrices

different from those presented by Bollen et al. Each matrix is of size 300×300. Random noise has been introduced, meaning that for each pixel we invert the value with a probability p. These models are depicted in Figure 5.1.

- **Pareto**: represents a nested pattern. Instead of having a triangular structure, the pattern is characterized by a more curved shape.
- **Banded**: characterized by a band of 60 pixels wide.
- **Blocks**: represents a diagonal of 6 blocks/clusters. There is a slight overlap between the adjacent blocks.
- **Triangles**: This pattern is more complicated to reorder. It is composed of several triangles aligned on the main diagonal.

We will also study two real-world datasets, both characterized by a nested structure:

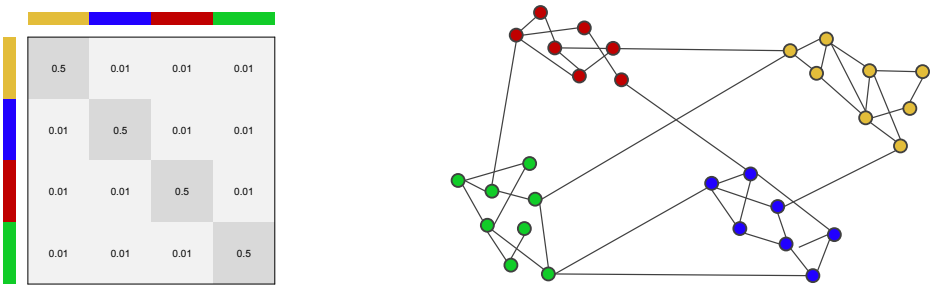
- **Mammals** [25]: represents a table that shows the presence of European mammal species (124 columns) at a given location (2200 rows). The value in the table is positive if a particular species has been seen at a specific location.
- **BACI (exportations)** [26]: this dataset is derived from a database of international trade at the product level (more than 5,000) and covering virtually every country in the world. This dataset also contains the export and import flows. We extracted from this data two-mode matrices `<countries, products>` where each point indicates that a country has exported a particular product. We have a matrix for each year from 1995 to 2018, of size 200×5000.

Binary data (networks)

As a reminder, one-mode binary data represent undirected graphs or networks modeling relationships between entities. These data in matrix form are called adjacency matrices. We will study four artificial models and four real datasets that represent sparse networks.

Stochastic block model

In order to generate artificial networks, we propose to use a model called **Stochastic Block Model** (SBM) [28, 29, 30]. This approach allows to generate random graphs that contain communities (or clusters, blocks). These communities group nodes that are connected to other communities in a similar way (Figure 5.2).



(a) Block-matrix (b) Generated graph

Figure 5.2: Assortative stochastic block modeling

Let's assume that we want to generate a graph with n nodes and k communities by using a SBM. We define a matrix $\mathbf{P}$ (Block-matrix) of size $k \times k$ where $\mathbf{P}_{u,v}$ gives the probability that a node belonging to community u (C_u) is connected to a node belonging to community v (C_v). We also define a memberships vector which indicates for each node which community it belongs to. Thus, if node $i \in C_u$ and node $j \in C_v$, the entry $\mathbf{M}_{i,j} \sim Bernoulli(\mathbf{P}_{u,v})$. The structural pattern of the network is hence determined by the block-matrix.

0.1	0.01	0.01	0.01	0.01
0.01	0.1	0.01	0.01	0.01
0.01	0.01	0.1	0.01	0.01
0.01	0.01	0.01	0.1	0.01
0.01	0.01	0.01	0.01	0.1

(a) Assortative

0.3	0.15	0.05	0.01	0.01
0.15	0.3	0.15	0.05	0.01
0.05	0.15	0.3	0.15	0.05
0.01	0.05	0.15	0.3	0.15
0.01	0.01	0.05	0.15	0.3

(b) Ordered

0.05	0.33	0.25	0.15
0.33	0.05	0.33	0.25
0.25	0.33	0.05	0.33
0.15	0.25	0.33	0.05

(c) Disassortative

0.55	0.3	0.15	0.08
0.3	0.3	0.15	0.08
0.15	0.15	0.15	0.08
0.08	0.08	0.08	0.08

(d) Core-periphery

Figure 5.3: Block-matrices for different communities structures

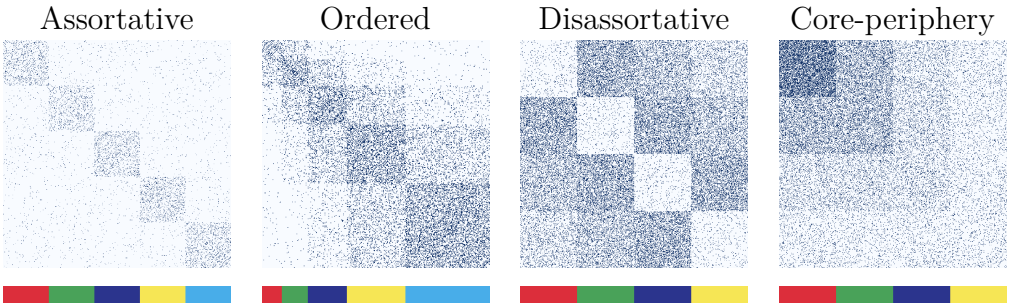


Figure 5.4: Artificial networks

We consider 4 popular structures generated by different SBMs as artificial data presented in [27, 28]. Figure 5.3 represents the block-matrices and Figure 5.4 represents the generated network matrices with their ground-truth community label (colored bar). This ground-truth information will be used to evaluate the quality of the order (how well the matrices are partitioned) with an accuracy score.

- **Assortative**: Nodes are essentially connected if they belong to the same community. They constitute cohesive clusters. Inter-cluster connections are very weak. ($k = 5$, $n = 500$)
- **Disassortative**: Communities are strongly connected to each other but intra-community nodes are weakly connected together. ($k = 4$, $n = 400$)
- **Ordered**: Very similar to assortative model. There is however a slight overlap between the consecutive communities ("linear hierarchical order"). This kind of pattern can describe sub-communities within communities and is more realistic. For this model we have created communities of different sizes. ($k = 5$, $n = 350$)
- **Core-periphery**: Represents an onion structure where communities constitute layers. There is a strongly connected central community and other communities that are less and less densely connected but still keeping a certain level of connection with the core. ($k = 4$, $n = 400$)

Real networks

In addition, we will evaluate the framework on real world networks. We took care to select networks where the communities (labels) or ground-truth is known in order to evaluate the quality of the order in the same way as the artificial data.

- **Political blogs** [31]: Represents political blogs for the 2004 US elections. There are 2 communities (liberal or conservative oriented blog). The original

graph is directed but we have symmetrized the data (undirected). We removed the nodes with less than 5 connections. ($k = 2$, $n = 852$)

- **Mail departments** [32]: Represents email addresses in a large European institution. It models the relationship between these addresses (sent an e-mail). The dataset originally contains 42 departments, but has been preprocessed to remove nodes with less than 5 connections and keep the 6 largest communities. ($k = 6$, $n = 329$)
- **Indian village** [33]: The dataset contains data from 76 villages (South India). Each village has an adjacency matrix where the entries represent the interaction links between the different households (one node = one household). The caste of the household is known and will be used as a ground truth community label. ($k = 3$, $n = 356$)
- **News** [34]: provide the adjacency matrix for 600 corpus of news documents divided into 3 categories. The links model the similarity between these documents. We applied a threshold value of 0.05 in order to binarize the matrix.($k = 3$, $n = 600$)

Numerical

For numerical data, we will simulate artificial matrices representing blocks with dense areas. To generate these data [35, 36], we use two normal distributions: $X_p \sim \mathcal{N}(1, 1)$ for the clusters and $X_n \sim \mathcal{N}(-1, 1)$ for outside the blocks. Two choices are possible to model the noise. Either we reduce the gap between the distributions; or we introduce a random noise which consists in inverting $p\%$ of the values. Other more complex models could be designed, the objective here being to simulate the kind of matrices that could be treated in bioinformatics for gene-expression analysis (micro-array experiment) for example. When executing the framework, the values are temporarily normalized in the interval [0, 1].

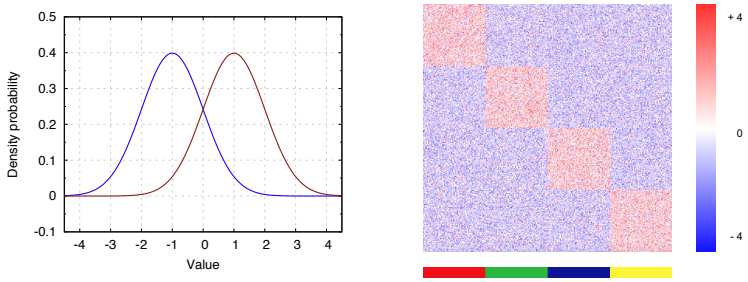


Figure 5.5: Synthetic numerical data generation

Categorical

To test our algorithm on categorical data, we will use 3 artificial datasets Figure 5.6. The first model is composed of 3 categories and has a *cluster* structure. One cluster per category and a random background. A noise of 45% is added to the clusters. The second model is quite similar and composed of 6 categories with one *rectangle* per category. A noise of 50% has been added. Finally, the third model is divided into 4 *triangles* (one per category) randomized with a noise of 40%.

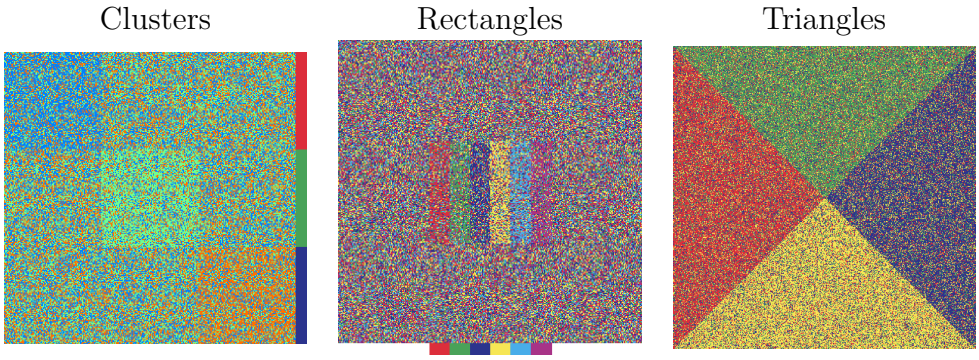


Figure 5.6: Categorical models

It is obvious that these models are only of interest to test our framework and see how it adapts to this type of data, especially the advanced thresholding presented in the previous chapter. The real world categorical data are in fact composed of categories that are specific for each column like a database table (gender, age, situation, school grade, ...). We will not focus on this type of data.

5.3 Questions

The experiments will be guided by the following questions:

Q1 How does the iterative framework improve existing methods in terms of **quality** for tabular binary data (**Q1a**) and for sparse networks (**Q1b**)?

Q2 How does the iterative framework compare to existing methods with respect to **execution time**?

Q3 What is the impact of **noise** on the ordering?

Q4 How well does the **smoothing** refinement improve the results?

Q5 For which type of data is a **thresholding** necessary?

Q6 How does the framework adapt to **numerical** matrices?

Q7 How does the framework adapt to **categorical** matrices?

5.4 Results

Q1a: improvements for tabular data

This question is answered by comparing the orders produced by the conventional (*basic*) methods and their embedded version through the *iterative* framework. We benchmark the methods mentioned in section 5.1 for the 4 artificial datasets (Figure 5.1) with a noise level $p = 0.2$ (we use the same PRNG seed). Before reordering these matrices, the order of the rows and columns is randomized. To evaluate the quality of the produced matrices, we will use 2 criteria. First, the visual aspect of the matrix (which is a subjective aspect). Secondly, the evaluation (or stopping) criterion $\mathbf{K_c}$ (the same one used in ConvoMap).

The obtained scores are summarized in Table 5.3 with the original score for the unshuffled matrix as reference. The Tables 5.1 and 5.2 show the visual results for some of these methods. This makes it easy to verify that the evaluation score is a good proxy for evaluating the quality of an ordering.

Table 5.1: Comparative table Pareto and Banded

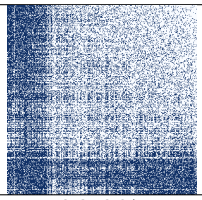
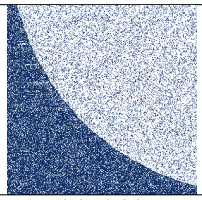
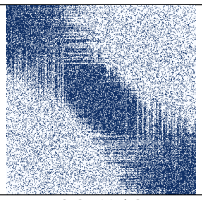
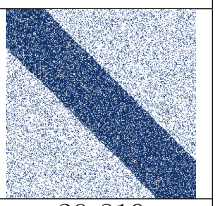
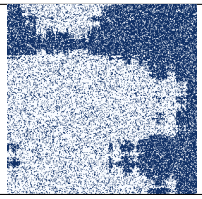
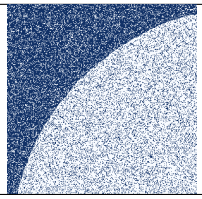
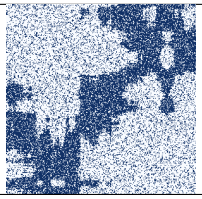
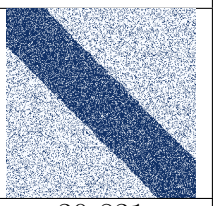
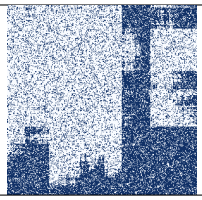
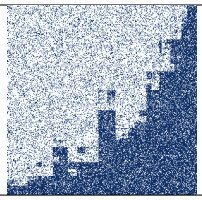
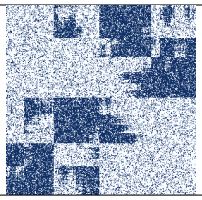
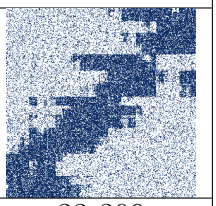
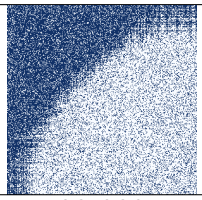
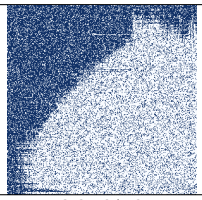
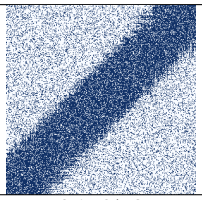
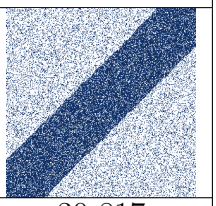
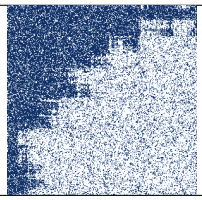
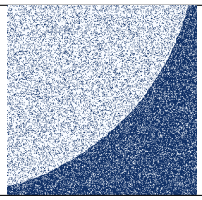
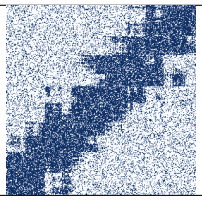
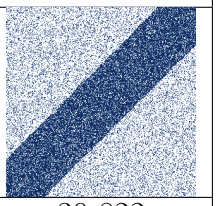
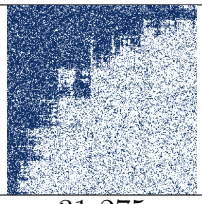
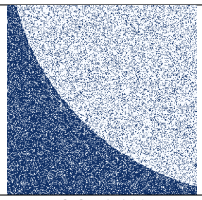
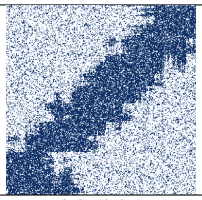
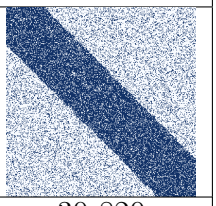
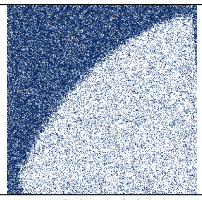
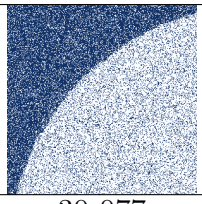
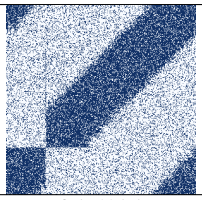
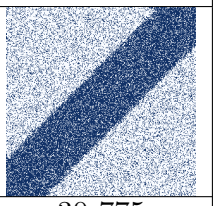
Methods	Pareto		Banded	
	basic	iterative	basic	iterative
Barycentric				
	36 087	30 168	33 549	30 819
TSP				
	32 469	30 155	34 076	30 831
HC				
	32 823	31 403	34 636	33 309
MDS angle				
	30 898	30 870	31 079	30 817
GW				
	31 258	30 115	32 686	30 822
OLO				
	31 275	30 115	32 081	30 820
QAP BAR				
	30 162	30 077	31 511	30 775

Table 5.2: Comparative table Blocks and Triangles

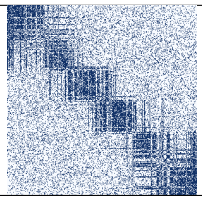
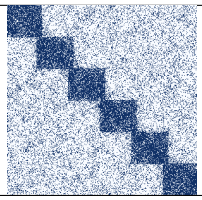
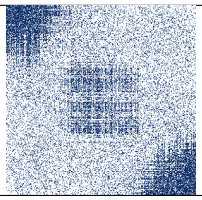
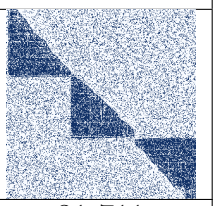
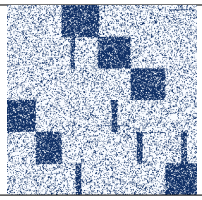
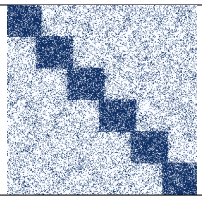
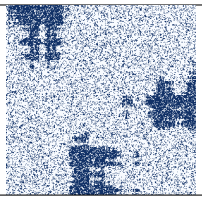
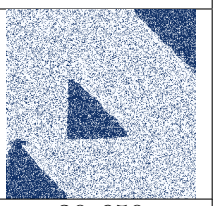
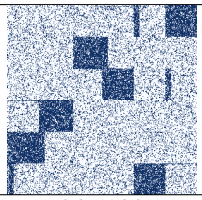
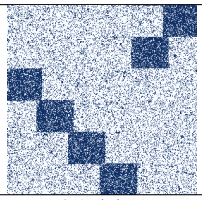
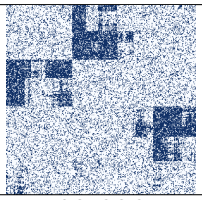
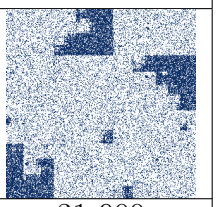
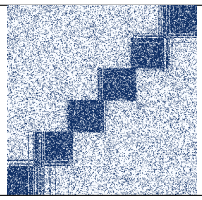
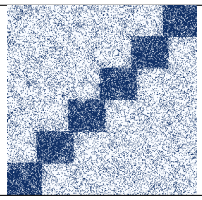
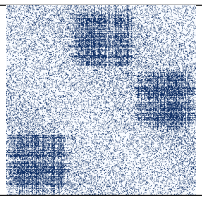
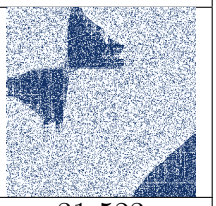
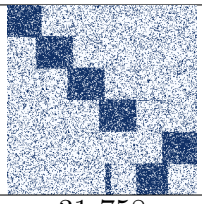
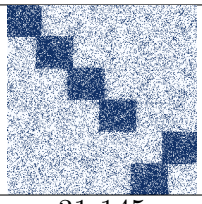
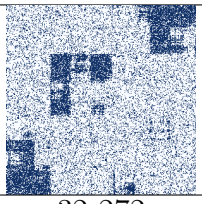
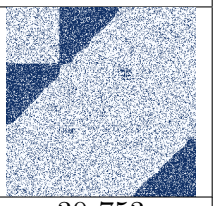
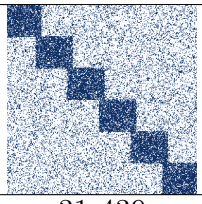
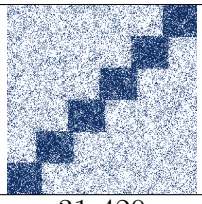
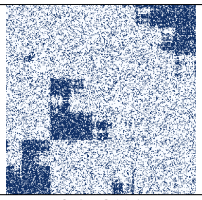
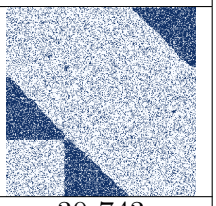
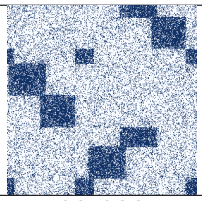
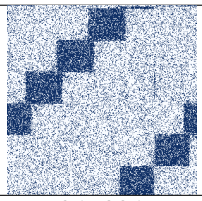
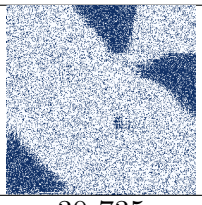
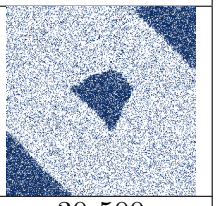
Methods	Blocks		Triangles	
	basic	iterative	basic	iterative
Barycentric				
	33 851	31 057	32 908	31 711
TSP				
	32 814	31 040	32 313	30 650
HC				
	32 563	31 775	32 926	31 900
MDS angle				
	31 673	31 006	33 509	31 523
GW				
	31 758	31 145	32 272	30 753
OLO				
	31 439	31 420	31 951	30 743
QAP BAR				
	32 029	31 691	30 735	30 500

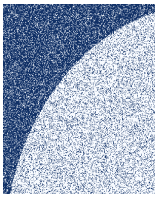
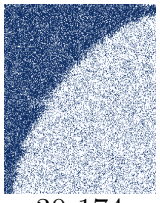
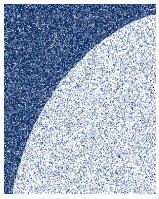
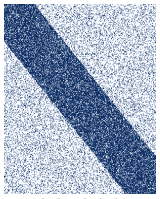
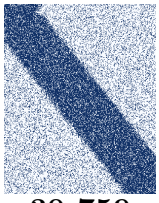
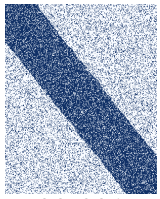
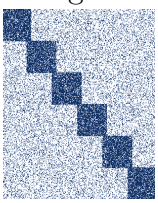
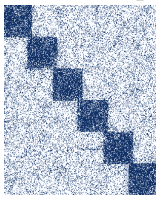
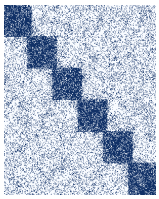
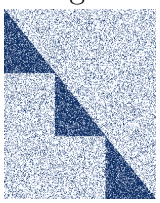
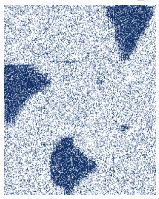
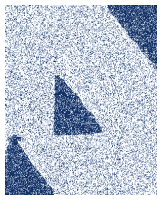
Table 5.3: Error scores comparison. The scores better than the original order are indicated in bold. The best score for a given dataset is highlighted.

Method	Pareto 20%		Banded 20%		Blocks 20%		Triangles 20%	
	basic	iterative	basic	iterative	basic	iterative	basic	iterative
original	30 123		30 839		31 732		31 213	
ConvoMap	30 174	-	**30 759**	-	**31 151**	-	31 259	-
`barycentric`	36 087	30 168	33 549	**30 819**	33 851	**31 057**	32 908	31 711
`TSP`	32 469	30 155	34 076	**30 831**	32 814	**31 040**	32 313	**30 650**
`HC`	32 823	31 403	34 636	33 309	32 563	31 775	32 926	31 900
`GW`	31 258	**30 115**	32 686	**30 822**	31 758	**31 145**	32 272	**30 753**
`MDS`	30 690	**30 118**	35 698	35 698	34 475	34 465	32 721	32 079
`MDS angle`	30 898	30 870	31 079	**30 817**	**31 673**	**31 006**	33 509	31 523
`Spectral`	30 344	**30 091**	34 765	32 502	35 932	33 437	35 587	31 437
`QAP 2SUM`	30 298	**30 116**	34 553	31 386	34 338	32 533	35 834	33 057
`QAP BAR`	30 162	**30 077**	31 511	**30 775**	32 029	**31 691**	**30 735**	**30 500**
`OLO`	31 275	**30 115**	32 081	**30 820**	**31 439**	**31 420**	31 951	**30 743**

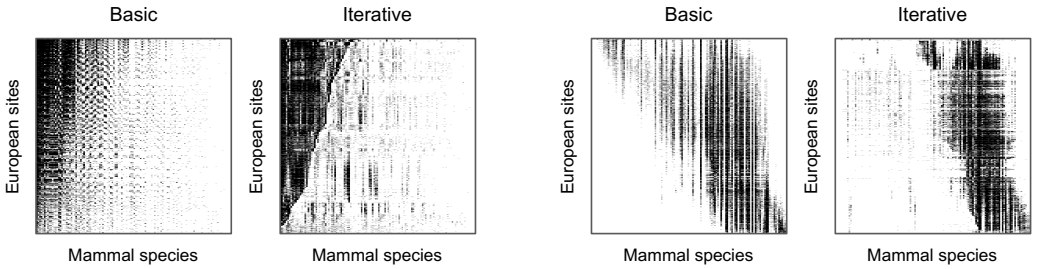
Since the framework uses the basic solution as a starting point with the objective of improving it, it is not surprising that the score of the iterative version is consistently better. The basic method as shown previously is unable to produce a good order, only pieces of patterns that we called embryonic and are difficult to interpret. The iterative method is however able to find the **exact original** pattern for almost all methods. $\mathbf{K_c}$ is in fact slightly better than the score of original data for more than 50% of the cases. We had formulated two assumptions in Chapter 4; a method is only able to recover the original pattern if it can do so on a noise-free matrix. The `HC` method for example is not able to recover a linear banded structure, so it cannot be improved by the framework. This experiment shows that the framework is able to overcome the obstacle of noise and improve methods that were previously ineffective.

Table 5.4 compares the results of the framework with the ConvoMap approach presented in the previous work by Bollen et al. [2]. The order produced is similar. This is not surprising since both methods are based on convolution. The iterative framework produces these results in a much shorter time (minutes versus hours). This is explained by the fact that ConvoMap uses the evaluation criterion as a scoring function through a randomized local search algorithm; the generation of moves and the continuous recalculation of the score seriously affects the performance. In our framework, we use repeatedly faster (based) methods on noiseless models that are easier to reorder.

Table 5.4: Comparative results for ConvoMap and iterated algorithm. TSP is used as base method.

Pareto			Banded		
Original	ConvoMap	Iterative	Original	ConvoMap	Iterative
					
30 123	30 174 20 841 s	30 155 98 s	30 839	**30 759** 22 754 s	30 831 109 s
Blocks			**Triangles**		
Original	ConvoMap	Iterative	Original	ConvoMap	Iterative
					
31 732	31 151 40 489 s	**31 040** 75 s	31 213	31 259 18 551 s	**30 650** 148 s

Finally, we evaluate the framework on two real data sets (mammals and BACI exports). Figure 5.7 shows the mammals matrix reordered with the `nested` and `barycentric` methods. As expected, the former is more efficient because the inherent structure of the pattern is nested. The distribution of mammals forms a hierarchical structure where the distribution of these species increases as the climate becomes warmer. However, in both cases, there is a clear improvement thanks to the framework. The nested pattern is for example much more visible, we observe for instance small clusters outside the primary structure.



(a) Nested ordering (b) Barycentric ordering

Figure 5.7: European mammals dataset: comparison between basic and iterative.

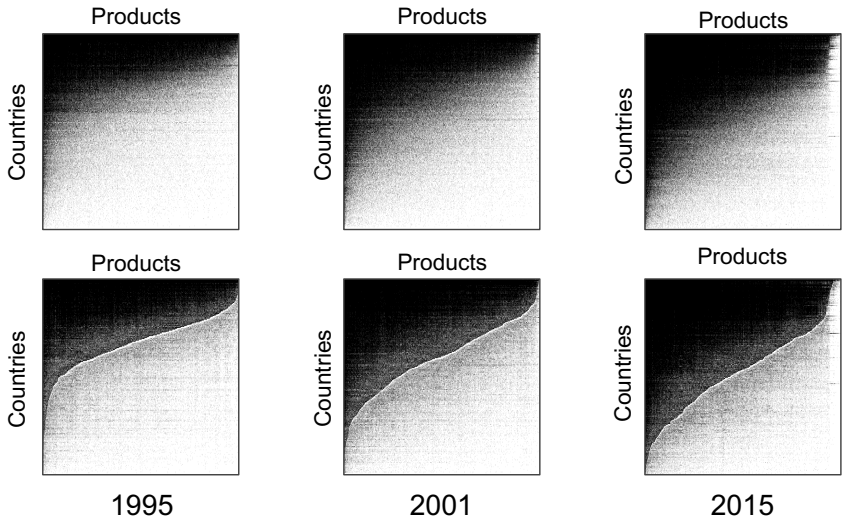


Figure 5.8: Nested ordering of BACI dataset

Figure 5.8 shows an application of the framework on BACI export data (basic first line, iterative second). We applied the algorithm on all matrices from 1995 to 2018 with the nested method. As a reminder, an entry `<country, products>` $= 1$ means that the country exported the product that year. It is an interesting finding that the distribution of these products display a nested structure like for ecological networks [37]. The wealthiest developed countries decide to diversify their exports instead of developing only on highly specialized products [8]. The experience shows that the framework considerably improves the quality of the image produced. We can distinguish a very clear distribution with a sharp demarcation. We can also

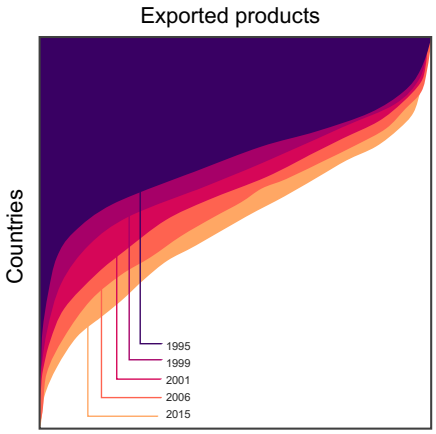


Figure 5.9: Evolution of nestedness

visualize the evolution through the years of the nestedness of the distribution (Figure 5.9).

Q1b: improvements for networks data

To evaluate this type of data, the same experimental process will be used. Instead of using $\mathbf{K_c}$, we asses the quality by using an accuracy score based on the communities ground-truth. Each node (row/col) belongs to a community; either from the SBM or from the real dataset metadata. The better the ordering, the more likely these labels will be ordered together (well clustered matrix). The calculation process for the accuracy is described in Figure 5.10. For each position, the most common label among the *10-nearest* (rows) neighbors is calculated. If the most common label corresponds to the original row label it is a *match*. The ratio $\frac{matches}{nodes}$ is the accuracy score. For the figure example, we take the row and its *4-nearest* neighboring rows to calculate the most common label. Ties are randomly drawn.

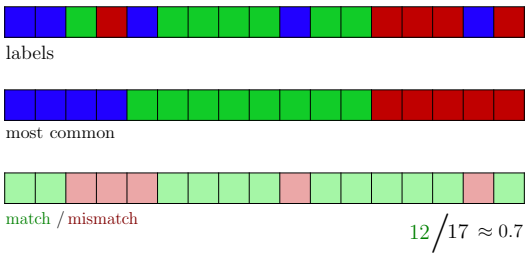
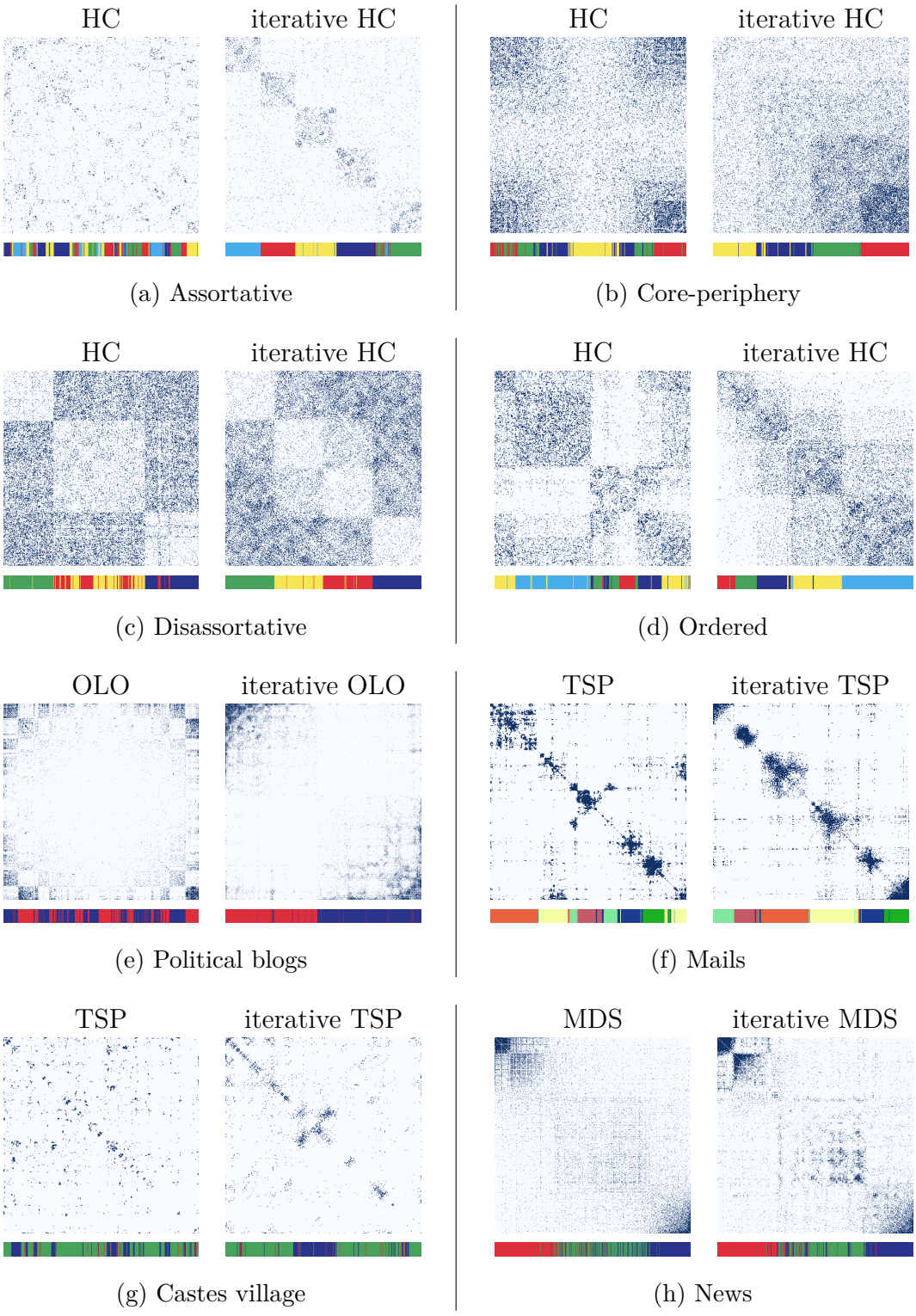


Figure 5.10: Accuracy of labels with *4-nearest*

In this evaluation, the algorithm initially intended as a visualization tool, is derived from it purpose as a community detection algorithm. Table 5.5 and Figure 5.12 summarize the experiments. We benchmarked the four synthetic and real datasets with five different methods by comparing the basic and iterated versions (once again, the basic solution used as input solution for the framework). It turns out that almost all the methods are improved with respect to the accuracy score (numbers in bold).

These results are confirmed visually in Figure 5.11. There is a clear improvement between the basic and iterated version for the quality of the images produced. The basic methods are unable to correctly partition the matrices and offer a coherent view of the clusters and communities. The problem here is more related to the sparsity of the data rather than the noise. The ground-truth bar is also in adequacy with the accuracy score we have formalized.



(a) Assortative (b) Core-periphery

(c) Disassortative (d) Ordered

(e) Political blogs (f) Mails

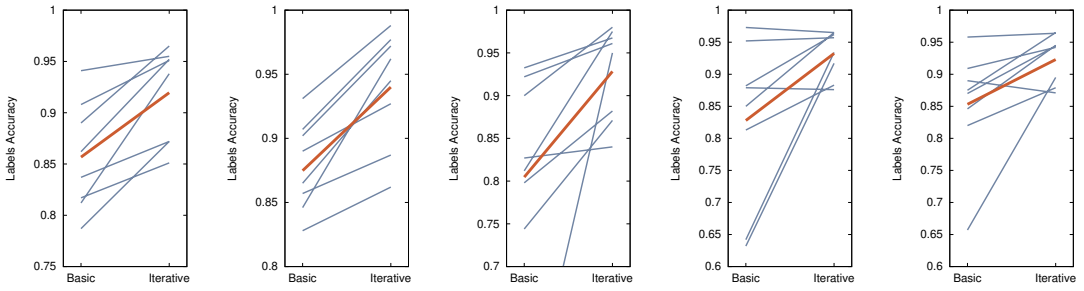
(g) Castes village (h) News

Figure 5.11: Results for synthetic and real networks with ground truth community labels

Table 5.5: Experiments for synthetic and real networks reordering.

Dataset	n	k	HC		OLO		GW		MDS		TSP	
			b	i	b	i	b	i	b	i	b	i
a) Artificial data												
Assortative	500	5	81.2	**93.8**	84.6	**96.2**	50.2	**95.0**	64.2	**93.4**	87.0	**94.4**
Core-periphery	400	4	86.2	**95.2**	86.5	**94.5**	93.2	**96.7**	95.2	**95.7**	65.7	**89.5**
Dissasortative	400	4	89.0	**96.5**	90.7	**97.7**	81.2	**97.5**	85.0	**96.5**	87.5	**96.5**
Ordered	350	5	90.8	**95.1**	93.1	**98.8**	90.0	**98.0**	88.2	**96.2**	84.6	**94.5**
b) Real data												
Political blogs [31]	852	2	94.1	**95.5**	90.2	**97.2**	92.2	**96.1**	**97.3**	96.5	95.8	**96.4**
EU-emails [32]	329	6	78.7	**87.2**	89.0	**92.7**	82.7	**84.0**	63.2	**91.7**	90.9	**94.2**
Village castes [33]	356	3	81.7	**85.1**	82.8	**86.2**	74.4	**87.1**	**87.9**	87.6	82.0	**87.9**
News [34]	600	3	83.7	**87.2**	85.7	**88.7**	79.8	**88.2**	81.3	**88.3**	**89.0**	87.1

Figure 5.11a to 5.11d shows the results for the artificial data with the HC method. The results are visually similar for the other methods. Using Hierarchical Clustering on an adjacency matrix of a graph in order to discover communities is not common and very inefficient as shown with the experiment. However, thanks to the framework, we can boost this method and discover the communities in a visual way. Figure 5.11e to 5.11h shows the results for real-world graphs. We show the results for other based methods. Once again, the framework manages to identify clusters correctly. The image is more interpretable and the noise granularity is smoothed. We notice for example: correctly polarized political parties, identification of a caste in the village, and so on ...



(a) HC (b) OLO (c) GW (d) MDS (e) TSP

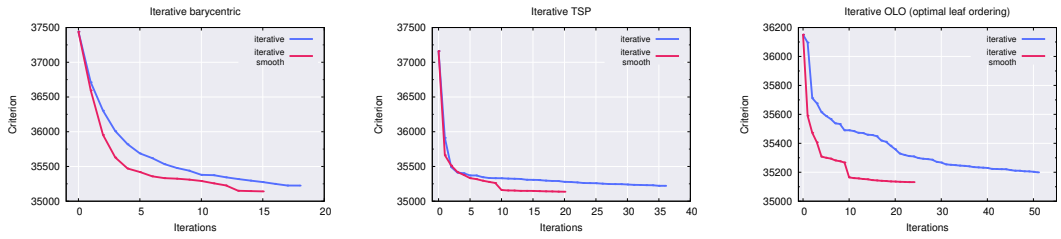
Figure 5.12: Accuracy improvement for each based method.

Q2: improvements for execution time

We have already shown in Table 5.4 the execution time for different datasets. Experiments have shown that the type of data (sparse or noisy) has little impact and that it is essentially (1) the based method used, (2) the size of the matrix, (3) the size and number of kernels in κ that are determining factors. If n is the matrix size (assumed square) and k the kernel size, the computation of the convolution is therefore $\mathcal{O}(n^2k^2)$.

The execution time is intrinsically linked to the number of iterations (Figure 5.13). The selected based methods from the R library are optimized and their execution time is more or less the same. Since these methods are repeated a limited number of times, the framework rarely exceeds 3 minutes of execution for 300×300 matrices.

An important point is that the more the execution progresses, the smaller the improvement will be. Most of the optimization takes place during the first iterations of the framework. Execution times could be much shorter (30 sec) if we introduce an improvement threshold or an early stopping strategy.



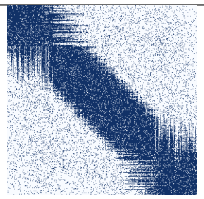
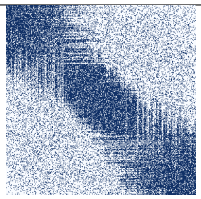
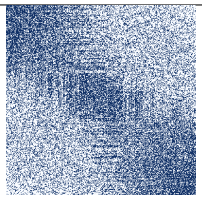
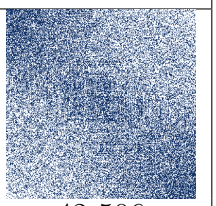
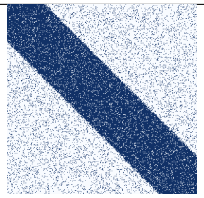
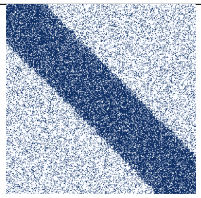
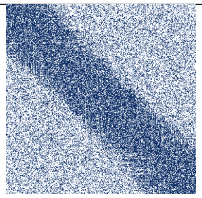
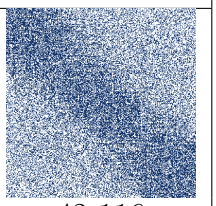
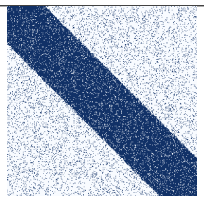
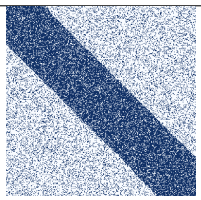
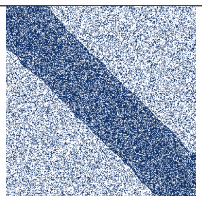
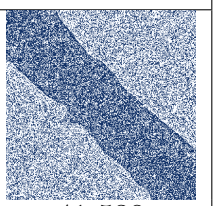
(a) Barycentric (b) TSP (c) OLO

Figure 5.13: Number of iterations and smoothing impact (band dataset 25% noise)

Q3: impact of noise

We will study the impact of noise on the ordering output. To do so, we take a banded dataset and analyze the results using the `barycentric` method for different noise levels (p =`0.1`, `0.2`, `0.3`, `0.35`). Table 5.6 shows the difference between the basic method (1st row) and its iterated version, smoothing enabled (3rd row). We notice that the framework is quite noise resilient since it can handle a level up to 30 - 35% and recovers the initial structure.

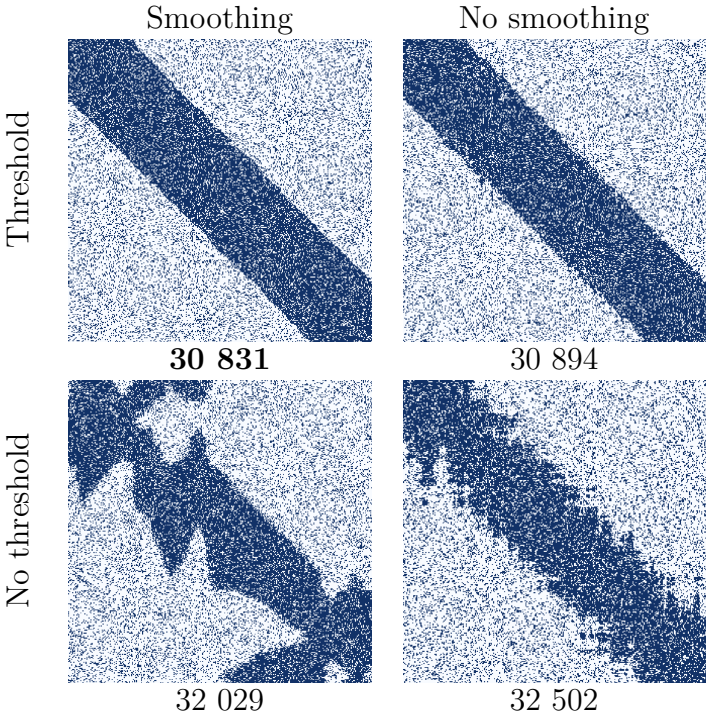
Table 5.6: Comparative results for barycentric methods on the banded dataset with different noise levels.

	10 % noise	20 % noise	30 % noise	35 % noise
barycentric				
criterion	22 179	33 549	40 510	42 506
iterative simple				
criterion	19 868	30 933	39 186	42 116
iterative smooth				
criterion	19 863	30 819	38 759	41 538

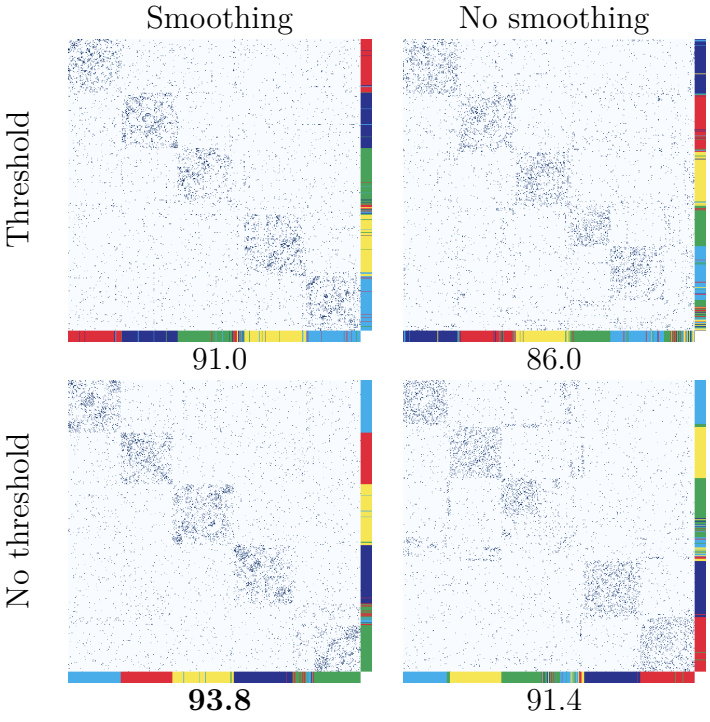
Q4: impact of smoothing

On the same Table (5.6), we study the effect of smoothing refinement on the output ordering. The 2nd row of the table proposes the order produced with smoothing deactivated. We note that beyond 20%, this improvement is necessary to obtain a clearer shape pattern. The objective is to minimize the difference between the matrix and its thresholded version.

To deepen the question, Figure 5.13 shows the impact of smoothing on the execution time or number of iterations on the same dataset (banded 25%) and for 3 different methods with the same κ set (not reordered). An important finding is that the smoothing refinement does not only have an impact on the quality, but also on the speed of the algorithm. The solution converges much faster; smoothing can speed up the framework by 50%.



(a) Tables



(b) Networks

Figure 5.14: Impact of thresholding and smoothing

Q5: use of thresholding

We stated in the experimental setup that dense tabular data should be reordered with the thresholding active and inactive for sparse networks. We will explain the intuition through the following experiment (Figure 5.14). The first figure shows the `TSP` method used on a 20% banded dataset with the evaluation criterion. The second shows the `HC` method on an assortative dataset with the accuracy score.

For dense data, it is better to use the thresholding because it extracts the whole pattern without noise and it is easier to reorder it (instead of having a continuous spectrum). Extracting the dissimilarity matrix from a non-thresholded matrix and apply it through a TSP algorithm does not give an optimal solution.

For sparse data, the thresholding only coarsens the points, while the blurred version contains more information about the surrounding pixels. (This phenomenon is illustrated in the Figure 4.4 in Chapter 4). The smoothing couples very well with thresholding; reordering a matrix $\mathbf{M}$ with a numerical template matrix $\mathbf{T}$ is particularly efficient for smoothing the picture. The choice to use thresholding is therefore motivated by the degree of sparsity of the data.

Q6: adaptation for numerical data

We evaluated the numerical dataset described in section 5.2 by varying the noise up to 32.5%. We generated 8 clusters on a 512×512 two-mode matrix. The `HC` method is used. The Table A.1 in Appendix summarizes the results. We use the accuracy score (based on the row ground truth) to evaluate the quality of the order produced. To obtain these results, thresholding and smoothing are activated.

The experiment shows that the basic method is unable to detect blocks above 20% of noise. But integrated into the framework, it can detect them up to 30%. This illustrates that the developed approach can also be used to reorder numerical data. It can therefore have a practical use since many matrices, especially heatmaps in bioinformatics, are commonly ordered with `HC` type of methods.

Q7: adaptation for categorical data

We have described 3 artificial models for categorical data. The results of the experiments are available in Tables A.2, A.3, A.4 of the appendix. Several methods have been used. Once again, we notice a clear improvement between the *basic* version and its *iterated* version. This proves that the thresholding technique we introduced is effective. For Table A.4 we recover the original pattern but for Table

A.2, the patterns are slightly distorted. Other work must be carried out to reorder categorical data from the real world. These data are more complex to visualize (find a suitable color code for each categories and make the image still understandable, different categories for each column).

CHAPTER 6

Discussion

6.1 Future work

In this chapter, we will introduce some possible adjustments and improvements that can be brought to the framework.

Turning the framework into a R package

The method developed is intended to be used as a data visualization tool. Since the implementation uses many methods from the R seriation library, it would be natural to implement the framework directly in R for reasons of both performance and simplicity. A prototype example is depicted in Figure 6.1.

```
> list_kernels <- list("gaussian3x3", "gaussian9x9", "gaussian15x15")
> my_matrix <- as.matrix(data)
> ordered_matrix <- iterative_ordering(x = my_matrix,
+         method = "TSP",
+         mode = 2,
+         distance = "Euclidean",
+         set = list_kernels,
+         criterion = "gaussian49x49",
+         threshold = TRUE,
+         smooth = FALSE,
+         iterations = 50 )
```

Figure 6.1: Speculative R code snippet for the framework

	Pareto		Banded		Blocks		Triangles	
	fixed	order	fixed	order	fixed	order	fixed	order
Exponential ↑	45%	66%	35%	63%	35%	63%	25%	50%
Linear ↑	43%	66%	33%	68%	32%	59%	28%	53%
Exponential ↓	64%	64%	70%	70%	33%	33%	64%	64%
Linear ↓	65%	62%	50%	61%	45%	50%	37%	37%

Table 6.1: Proportion of the solutions that are accepted.

Another way to optimize the framework would be to introduce parallelization. For example: to compute simultaneously different candidate solutions with different kernel types and even different methods. We can consider a set of reordering methods in order to diversify the produced order. We could also improve the parameterization by using a different method for the preliminary and iterative ordering.

Kernel replacement and generation

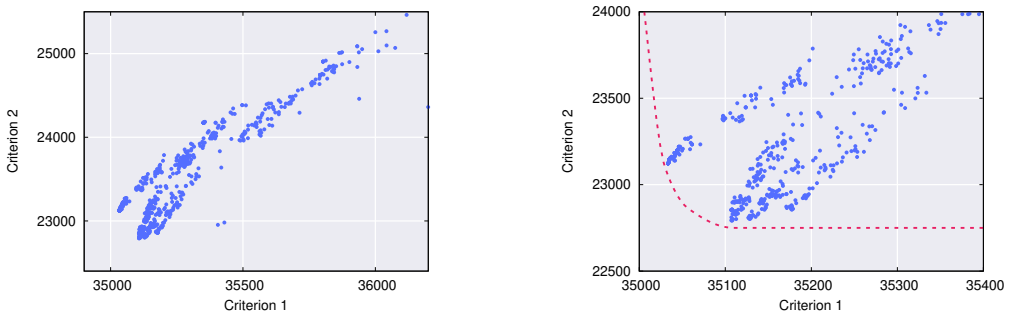
We claimed that the κ set, used for convolution, remains unordered during execution to make the results more predictable. However, experiments show that a significant fraction of the generated candidate solutions do not improve the best solution. To make the framework more efficient, it would be necessary to penalize the kernels that do not improve the solution. A *naive* approach would be to put them at the end of the set. A more complex solution would be to grant a penalty point system and to reorder it according to these points.

Table 6.1 shows the percentage of candidates that improved the solution. The "fixed" column means that κ is not reordered and the "order" means that it is *naively* reordered. We evaluate four different sets κ ($n \in \{3, 5, 9, 15, 25\}$) of different types and orders (increasing, decreasing). It is straightforward that the type of kernel, the order, and the way κ is reordered can seriously affect the proportion of kernels that fail. This illustrates the interest of developing a robust *replacement policy*. Finally, we could also generate the kernels of the set automatically by analyzing the noisiness or sparsity of the data case by case.

Evaluation criterion

We have seen the based method φ_b as a black box. The same principle can be applied to the evaluation criterion. We could integrate a new evaluation criterion that is not necessarily based on convolution. The use of several criteria can also

be considered; this problem is known as *multi-objective optimization*. Figure 6.2 shows for example different candidate solutions for the same dataset (banded 25%) obtained with different methods (different trajectory lines). We observe on the two axes, two evaluation criteria (different kernels) that need to be minimized.



(a) Candidates solutions (b) Pareto front

Figure 6.2: Bi-objective optimization

Partitions

The calculation of the convolution and reordering become more and more expensive as the size of the matrix increases. Some network data are composed of thousands of nodes. It would therefore be impossible to reorder their adjacency matrix. A solution would be to partition (Figure 6.3) the matrix into several sub-matrices and to reorder them independently in order to recompose them. A community detection algorithm could therefore be applied in a preliminary way to obtain these sub-matrices. We would thus create a composite tool allowing to visualize large networks.

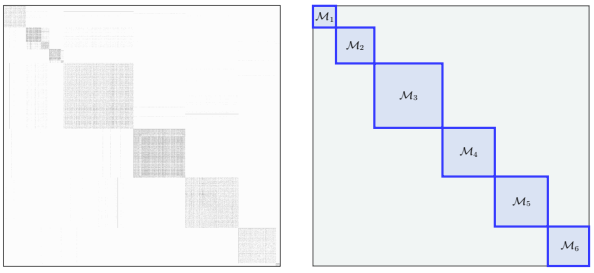


Figure 6.3: Large network partition [38]

6.2 Conclusion

In this work, we studied the *matrix reordering problem*, a task consisting in permuting columns and rows of a data table in order to visualize and discover interesting and informative underlying patterns. We explored the different methods presented in the literature and established their weaknesses.

Experiments have shown that these methods often fail to correctly reorder sparse or noisy matrices. The order produced does not reveal any coherent and interpretable pattern.

The main contribution of this thesis is the creation of an *iterative framework* that allows to integrate any method and make it more robust to noise by embedding it into an iterative process where the data is temporarily simplified by means of convolution.

We evaluated our algorithm on many synthetic and real-world datasets to compare them to standard methods. Experiments have shown that this approach is efficient and allows to recover the initial pattern. We have also seen that the algorithm easily adapts to any type of data, whether it is numerical, binary or artificial.

Some points remain to be improved. We have focused a lot on the results produced, but there is still work to be done in terms of optimization. By combining different methods and automating the generation / selection of kernels we could better automate the process.

There is also work to be done on the evaluation criterion: for example finding a criterion not based on convolution and less computationally expensive for very large matrices.

APPENDIX A

Numerical and categorical results

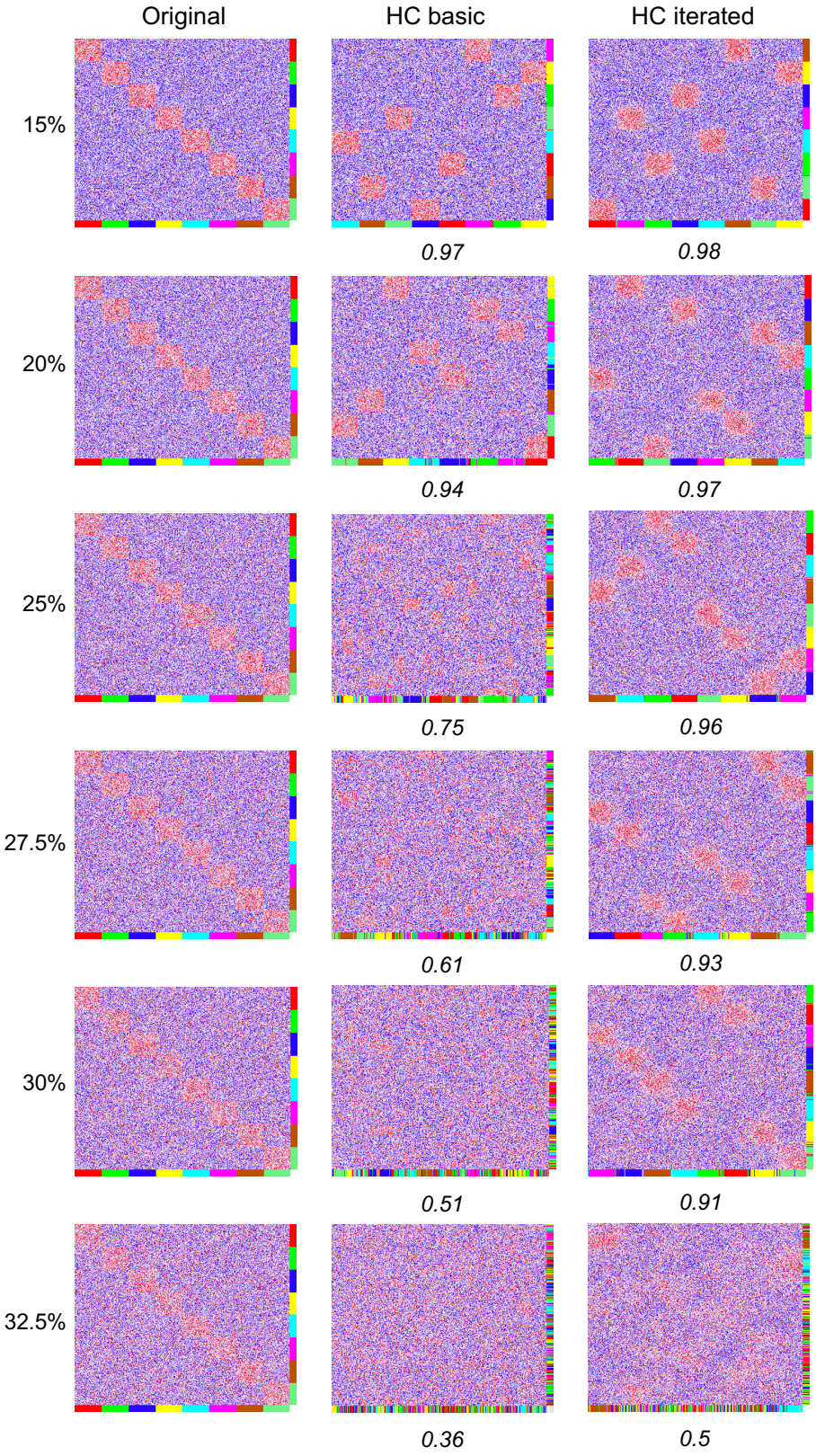


Table A.1: Results for synthetic numerical data with `HC`

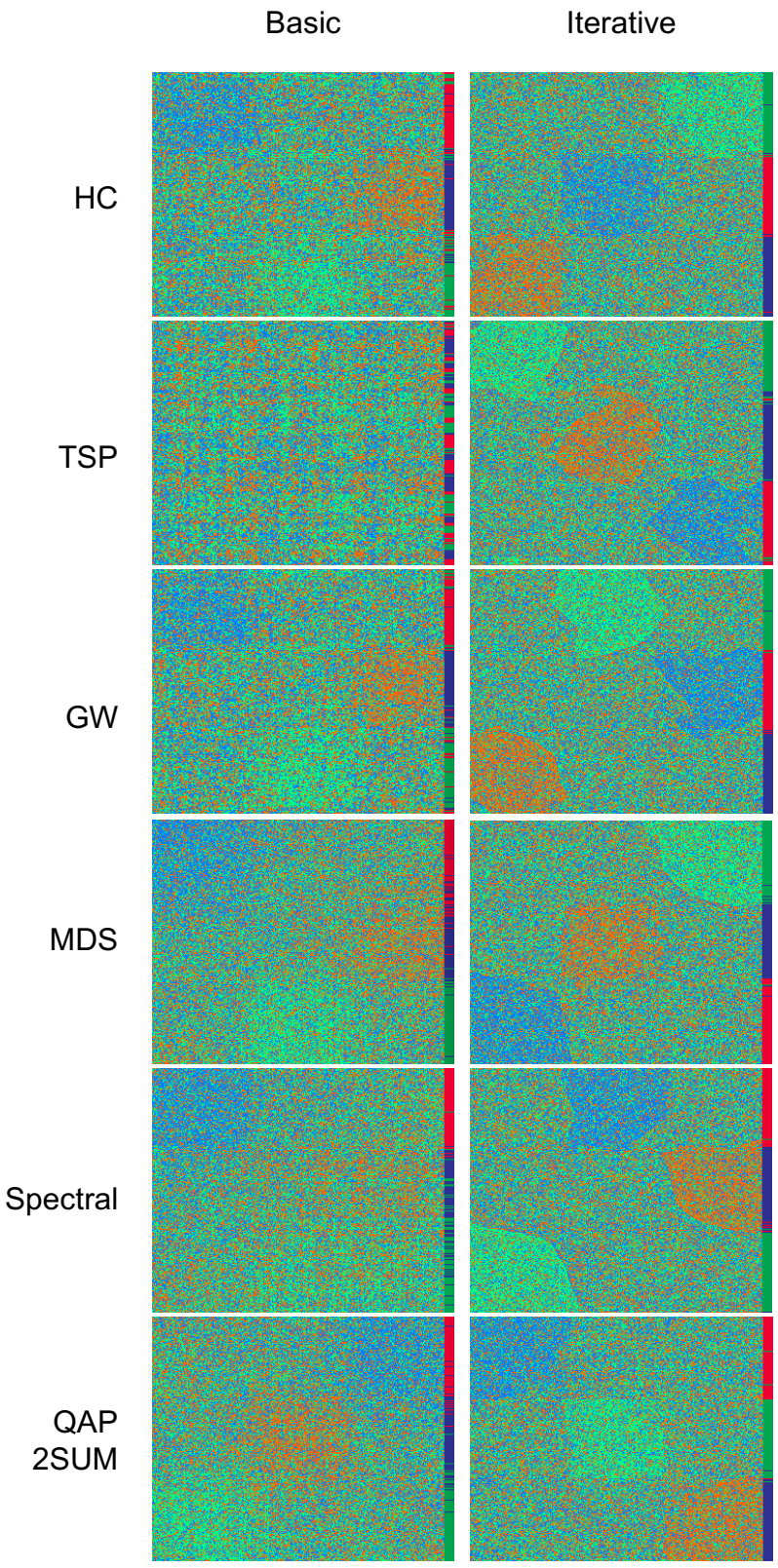


Table A.2: Results for synthetic categorical data for different methods (1st model)

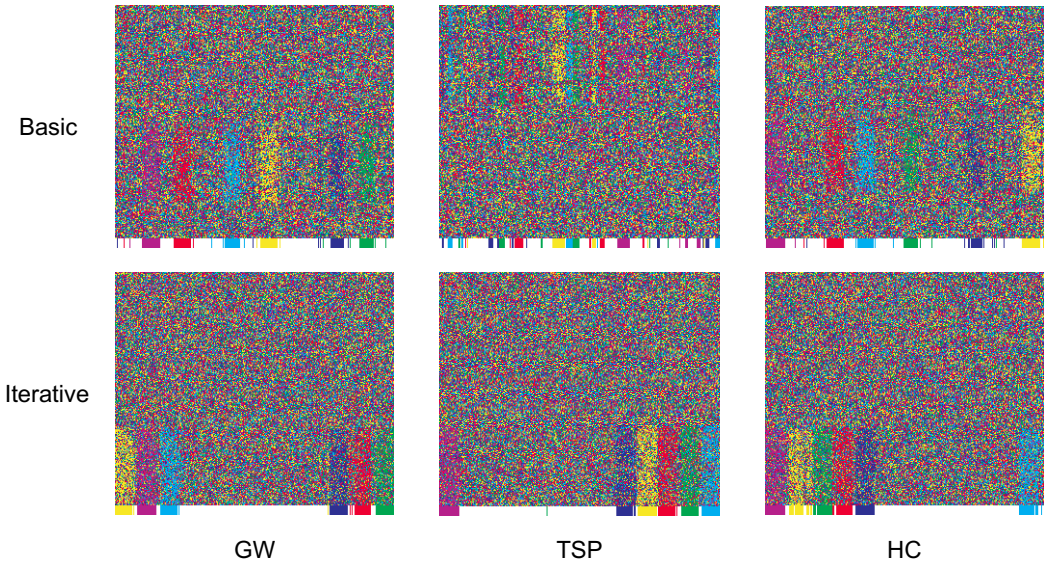


Table A.3: Results for synthetic categorical data (2nd model)

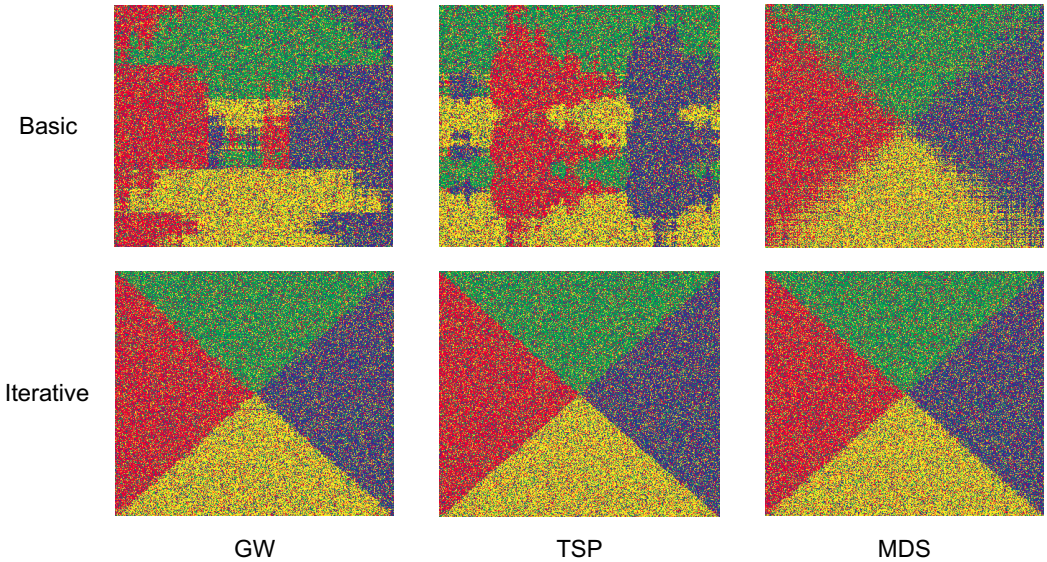


Table A.4: Results for synthetic categorical data (3rd model)

Bibliography

[1] Bollen, T., Leurquin, G. and Nijssen, S. (2018). ConvoMap: Using Convolution to Order Boolean Data. *17th International Symposium on Intelligent Data Analysis* (pp. 62-74). Springer

[2] Bollen, T. and Leurquin, G. Faithful visualization of categorical data. *Ecole polytechnique de Louvain, Universite catholique de Louvain*, 2017. Prom. : Nijssen, S., http://hdl.handle.net/2078.1/thesis:10643

[3] Fogel, F., Jenatton, R., Bach, F. and d'Aspremont, A. (2013). Convex Relaxations for Permutation Problems. *Advances in Neural Information Processing Systems*. 36. 10.1137/130947362.

[4] Natik, A. (2019). Consistency of the spectral seriation algorithm.

[5] Recanati, A. (2018). Relaxations of the Seriation problem and applications to de novo genome assembly.

[6] Liiv, I. (2010). Seriation and matrix reordering methods: An historical overview. *Statistical Analysis and Data Mining*, 3(2):70–91.

[7] Liiv, I. Pattern Discovery Using Seriation and Matrix Reordering: a Unified View, Extensions and an Application to Inventory Management, *Volume 39 de Thesis on Informatics and System Engineering*, TUT Press, 2008

[8] Mariani, M. S., Ren, Z.-M., Bascompte, J. and Tessone, C. (2019). Nestedness in complex networks: Observation, emergence, and implications. *Physics Reports*, Volume 813, 2019, Pages 1-90

[9] Brualdi, R. and Sanderson, J. Nested species subsets, gaps, and discrepancy. *Oecologia 119*, 256–264 (1999).

[10] Burns, K. Meta-community structure of vascular epiphytes in a temperate rainforest. *Botany*. 86(11): 1252-1259.

[11] Junttila, E. (2011). Patterns in permuted binary matrices. Master's thesis, University of Helsinki.

[12] Garriga, G. Junttila, E. and Mannila, H. (2008). Banded structure in binary matrices. *Knowledge and Information Systems*. 28. 197-226.

[13] Makinen, E. and Siirtola, H. (2005). The barycenter heuristic and the reorderable matrix. *Informatica (Slovenia)*, 29(3):357–364.

[14] Roy, S., Chakrabarti, P., Kumar, S., Chakrabarty, K. and Bhattacharya, B. (2015). Layout-Aware Mixture Preparation of Biochemical Fluids on Application-Specific Digital Microfluidic Biochips. *ACM Transactions on Design Automation of Electronic Systems*.

[15] Gower, J.C. and Legendre, P. Metric and Euclidean properties of dissimilarity coefficients. *Journal of Classification*, Volume 3, 5–48 (1986).

[16] Hahsler, M., Buchta, C. and Hornik, K. (2020). Seriation: Infrastructure for Ordering Objects Using Seriation. R package version 1.2-9. https://cran.r-project.org/web/packages/seriation/index.html

[17] Behrisch, M., Bach, B., Henry Riche, N., Schreck, T. and Fekete, J-D. (2016). Matrix Reordering Methods for Table and Network Visualization. *Computer Graphics Forum*. 35. 10.1111/cgf.12935.

[18] Hubert, L.J. Some applications of graph theory and related nonmetric techniques to problems of approximate seriation: The case of symmetric proximity measures. *British Journal of Mathematical Statistics and Psychology*, 27:133–153, 1974.

[19] Daniel, M. fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python, *Journal of Statistical Software*, 53 (2013), no. 9, 1–18, http://www.jstatsoft.org/v53/i09/.

[20] Gruvaeus, G. and Wainer, H. (1972). Two additions to hierarchical cluster analysis. *British Journal of Mathematical and Statistical Psychology*. 25. 200 - 206. 10.1111/j.2044-8317.1972.tb00491.x.

[21] Bar-Joseph, Z., Gi ord, D.K. and Jaakkola, T.S. (2001). Fast optimal leaf ordering for hierarchical clustering , *Bioinformatics*, 17(1), 22-29.

[22] N. Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62-66, Jan. 1979, doi: 10.1109/TSMC.1979.4310076.

[23] Liu, D. and Jian Y. Otsu Method and K-means. (2009). *Ninth International Conference on Hybrid Intelligent Systems*, 344-349.

[24] Hahsler, M., and Hornik, K. (2007). TSP-Infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, 23(2), 1-21. https://cran.r-project.org/web/packages/TSP/index.html

[25] Mitchell-Jones, A. J., Mitchell, J., Amori, G., Bogdanowicz, W., Spitzenberger, F., Krystufek, B., Vohralik, V., Thissen, J., Reijnders, P., Ziman, J., et al. (1999). *The atlas of European mammals*, volume 3. *London Academic Press*.

[26] G. Gaulier and S. Zignago. Baci: International Trade Database at the Product-Level (the 1994–2007 Version). CEPII Working Paper, 2010–23, 2010.

[27] Clauset, A. "Modular networks I: structure." Class lecture, Biological Networks CSCI 3352 (Fall 2019), University of Colorado at Boulder. http://tuvalu.santafe.edu/~aaronc/courses/3352/csci3352_2019_L5.pdf

[28] Clauset, A. "Inferring large-scale structural patterns." Class lecture, Biological Networks CSCI 3352 (2017), University of Colorado at Boulder. http://tuvalu.santafe.edu/~aaronc/courses/5352/csci5352_2017_L6.pdf

[29] Lee, C. and Wilkinson, D. J. (2019). A review of stochastic block models and extensions for graph clustering. *Applied Network Science*, 4(1), 1-50.

[30] Faskowitz, J., Yan, X., Zuo, XN. et al. Weighted Stochastic Block Models of the Human Connectome across the Life Span. *Sci Rep 8*, 12997 (2018).

[31] Adamic, L. A., and Glance, N. (2005, August). The political blogosphere and the 2004 US election: divided they blog. *in Proceedings of the 3rd international workshop on Link discovery*, (pp. 36-43).

[32] Yin, H., Benson, A.R., Leskovec, J., Gleich and D.F. Local Higher-order Graph Clustering. *In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[33] Banerjee, A., Chandrasekhar, A.G., Duflo, E. and Jackson, M.O. 2013, The Diffusion of Microfinance, Harvard Dataverse, V9, https://doi.org/10.7910/DVN/U3BIHX

[34] Ding, C.H., He, X., Zha, H., Gu, M., Simon, H.D. (2001). A min-max cut algorithm for graph partitioning and data clustering. *In Proceedings 2001 IEEE international conference on data mining*, pages 107–114.

[35] Branders, V., Schaus, P., and Dupont, P. (2017). Mining a sub-matrix of maximal sum. *arXiv preprint arXiv:1709.08461*.

[36] Le Van, T., Van Leeuwen, M., Nijssen, S., Fierro, A. C., Marchal, K., and De Raedt, L. (2014, September). Ranked tiling. *In Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 98-113). Springer, Berlin, Heidelberg.

[37] Alves, L. G., Mangioni, G., Cingolani, I., Rodrigues, F. A., Panzarasa, P., and Moreno, Y. (2019). The nested structural organization of the worldwide trade multi-layer network. *Scientific reports*, 9(1), 1-14.

[38] McAuley, J. J., and Leskovec, J. (2012, December). Learning to discover social circles in ego networks. *In NIPS* (Vol. 2012, pp. 548-56).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**