uc3m | Universidad Carlos III de Madrid

Master Degree in Statistics for Data Science

2020-2021

Master Thesis

Methods for fuzzy clustering based on linkage: A metaheuristic approach

Alejandro Marcos Rexach

Vanesa Guerrero Lozano

Andrés Modesto Alonso Fernández

Madrid, 25 June 2024

SUMMARY

This master thesis presents the development of new fuzzy clustering algorithms adapted to time series analysis. Traditional clustering methods often have difficulties in identifying underlying patterns in complex time-dependent data sets. To address this problem, the thesis introduces three models based on single, average and complete linkage methods to measure distances between groups of individuals in a fuzzy clustering framework. These models utilize cophenetic distance and generalized cross-correlation to create dissimilarity matrices. The aim is to improve the flexibility and accuracy of clustering by assigning degrees of membership to each data point, rather than strict assignments. Furthermore, the thesis proposes formulations of these three models as mixed-integer nonlinear programs. To tackle larger instances of these models, the metaheuristic algorithm Large Neighborhood Search (LNS) is explained in detail and employed to heuristically solve them, thereby improving computational efficiency.

The models are tested through extensive simulations, demonstrating that the complete linkage model, combined with the cophenetic distance, provides superior cluster classifications. Furthermore, the research confirms the good performance of the model by showing its effectiveness in identifying clear group structures using real data from the Italian electricity market collected between 2014 and 2018.

Keywords: Fuzzy clustering, cophenetic distance, large neighborhood search, linkage, generalized cross-correlation, time series.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Time series clustering is an essential technique in the framework of statistical analysis, which is crucial to identify time-dependent patterns in complex datasets. Data mining in time series includes searching for similar patterns in sets of time series to understand the underlying structures that researchers usually seek. Classical clustering methods usually fail to identify ways in which similar behaviors can be explained, and the search for a new way goes on—those that can reveal the existing patterns.

This project is based on the article by Alonso et al., 2021, in which a new methodology for clustering large datasets made up of time series, taking into account the cross-dependence of the time series to be clustered, is presented. In the aforementioned article, two new clustering classification algorithms were proposed, both based on the use of the cophenetic distance and considering a generalized cross-correlation measure proposed by Alonso and Peña, 2019. These two approaches will be described in Section 2, as they will also be useful for our new proposed clustering algorithms.

Clustering algorithms mainly fall into two categories: Soft clustering (Fuzzy clustering) and Hard clustering (Crisp clustering). On the one hand, Hard clustering is a technique that assigns each data point to exactly one cluster. The assignment is based on a similarity measure, such as the Manhattan or Euclidean distances. Hard clustering algorithms like k-means (see Lloyd, 1982) or hierarchical clustering (see Nielsen, 2016) partition the data into distinct, non-overlapping clusters. Each data point belongs exclusively to a single cluster, and there is no ambiguity in the assignment. On the other hand, Soft clustering, also called fuzzy clustering, makes assignments in a more flexible way. Fuzzy clustering allocates a probability or degree of membership to each cluster rather than designating a single data point to a cluster. This implies that a data point can belong to multiple clusters simultaneously, with varying degrees of membership. Fuzzy clustering algorithms determine the degree of membership that a data point belongs to each cluster. An example of these algorithms is fuzzy c-means (see, for instance, Dunn, 1973).

New fuzzy clustering algorithms will be developed in this project as they offer several advantages over hard clustering algorithms, making them more attractive in real-world applications. Firstly, fuzzy clustering is beneficial when it is difficult to identify clear boundaries between clusters, as it provides more flexibility. Furthermore, it is also useful when the dynamic properties of time series need to be captured; hard clustering methods may fail to capture these properties due to the switching behaviour in the data. Unlike hard clustering, fuzzy clustering provides a non-stochastic uncertainty measure, the membership degree, that captures the fuzzy belonging of data-points to clusters, offering a deeper understanding of the data. Another strong point related to the degrees of membership, is that the fuzzy approach allows for the identification and analysis of scenarios that hard methods might miss, such as second-best clusters. Finally, perhaps the most important advantage of fuzzy clustering is that it maintains the natural relational structure of the data by analysing the degrees of joint membership, which provides more detailed information about the inherent structure of the data.

The aim of this project is to create a new fuzzy clustering algorithm that is capable of finding groups of highly related time series and separating these groups from those series with a lower or no relationship. For this purpose, the single, average and complete linkage are introduced in the framework as different ways of measuring distances between groups of individuals/time series, as well as another key concept, the u threshold, which differentiates when comparing dissimilarities or not, depending on the degree of membership. Four different Mixed Integer Non linear Programming (MINLP) models will be created with these three linkages, and in addition to trying to correctly classify these time series, we will also try to identify which type of linkage works best in different scenarios. In addition, in order to speed up the computational time of the created models, a metaheuristic algorithm based on the Large Neighborhood Search paradigm and the aforementioned MINLP formulations to solve the fuzzy clustering problem in time series analysis will be created. It will provide good solutions to this time series classification problem in a shorter period of time. The new approaches developed in this work are compared with those in Alonso et al., 2021 using both simulated and real data sets.

The remainder of this project is organized as follows. In Chapter 2, the concept of the Generalized Cross Correlation measure (GCC), the Cophenetic Distance (CD) and the three models of the fuzzy clustering problem for time series are introduced. First, the problem is introduced taking into account the single linkage proposing two formulations, a series of disadvantages and problems using this model are exposed, and in order to try to solve them, the models with the complete and average linkage are introduced. Once the problem has been posed, Chapter 3 introduces the description of the Large Neighborhood Search Metaheuristic Algorithm (see Pisinger and Ropke, 2010) and how it is adapted to solve our fuzzy clustering problem. In Chapter 4, a simulation study is carried out with the aim of finding out which of all the proposed formulations performs best in solving the fuzzy clustering problems if we directly use a MINLP solver (Gurobi Optimization, LLC, 2023) and how the metaheuristic algorithm based on the best formulation obtained above performs. For this purpose, several simulated scenarios will be considered. To reinforce the usefulness of our new clustering approaches, in Chapter 5 the models are tested with a real data set, which collects time series of Italian electricity market prices from 2014 to 2018. It should be noted that this dataset is the same as the one used in Alonso et al., 2021, in order to compare with their results. Finally, Chapter 6 presents conclusions and possible future work. In addition, all the code that have been developed for the realisation of the project, in Python language, and that will be referenced throughout the project, can be found at https://github.com/AleeexMR/TFM along with a README.md file that explains the usefulness and use of each script.

2. OPTIMIZATION MODELS FOR FUZZY CLUSTERING

In this chapter we describe the models proposed with the aim of clustering $n \in \mathbb{N}$ individuals (time series) in a fuzzy way using the single, complete and average linkages to compute the distances between groups of individuals. Before starting to explain the formulations of the problem, two important tools used to create dissimilarity matrices that serve as inputs to our fuzzy clustering algorithms need to be introduced: The Generalized Cross-Correlation measure and the Cophenetic Distance.

2.1. The Generalized Cross-Correlation Measure

The Generalized Cross Correlation (GCC) measure, proposed in Alonso and Peña, 2019, allows us to quantify the linear dependency between two stationary time series, x_t and y_t. This measure is based on the correlation matrices of the series and takes into account cross-correlations and autocorrelations up to a specified lag, k.

Mathematically, GCC is computed as:

$$\mathrm{GCC}(x_t, y_t) = 1 - \left(\frac{|R_{yx,k}|}{|R_{xx,k}||R_{yy,k}|} \right)^{\frac{1}{k+1}}, \tag{2.1}$$

where:

- $R_{xx,k}$ and $R_{yy,k}$ are the covariance matrices of x_t and y_t respectively, up to lag k. In other words, they are the square and positive-definite covariance matrices of the standardized vectors of series $X_{t,k} = (x_t, x_{t-1}, ..., x_{t-k})'$ and $Y_{t,k} = (y_t, y_{t-1}, ..., y_{t-k})'$, respectively.
- $R_{yx,k}$ is defined as a combination of $R_{xx,k}$, $R_{yy,k}$ and $C_{xy,k}$, the latter being a square matrix that includes the cross-correlations between both vectors of series:

$$R_{yx,k} = \begin{pmatrix} R_{yy,k} & C_{xy,k}^T \\ C_{xy,k} & R_{xx,k} \end{pmatrix} \tag{2.2}$$

$$= \begin{pmatrix} 1 & \rho_y(1) & \cdots & \rho_y(k) & \rho_{xy}(0) & \rho_{xy}(1) & \cdots & \rho_{xy}(k) \\ \rho_y(1) & 1 & \cdots & \rho_y(k-1) & \rho_{xy}(-1) & \rho_{xy}(0) & \cdots & \rho_{xy}(k-1) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_y(k) & \rho_y(k-1) & \cdots & 1 & \rho_{xy}(-k) & \rho_{xy}(-k+1) & \cdots & \rho_{xy}(0) \\ \rho_{xy}(0) & \rho_{xy}(-1) & \cdots & \rho_{xy}(-k) & 1 & \rho_x(1) & \cdots & \rho_x(k) \\ \rho_{xy}(1) & \rho_{xy}(0) & \cdots & \rho_{xy}(-k+1) & \rho_x(1) & 1 & \cdots & \rho_x(k-1) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{xy}(k) & \rho_{xy}(k-1) & \cdots & \rho_{xy}(0) & \rho_x(k) & \rho_x(k-1) & \cdots & 1 \end{pmatrix}. \tag{2.3}$$

The GCC measure has the following properties:

1. Symmetry: $\text{GCC}(x_t, y_t) = \text{GCC}(y_t, x_t)$.
2. Bounded: $0 \leq \text{GCC}(x_t, y_t) \leq 1$.
3. Perfect Linear Dependency: $\text{GCC}(x_t, y_t) = 1$ if there is a perfect linear relationship between the series.
4. Linear Independence: $\text{GCC}(x_t, y_t) = 0$ if all cross-correlation coefficients are zero.

This measure is particularly useful as it combines the information of both series and their mutual influence over multiple time lags, providing a thorough understanding of their linear dependency.

In order to create a clustering procedure based on the GCC measure, we have to determine the number of lags to be used, estimate the GCC from the data and construct a dissimilarity matrix to be introduced as input to our corresponding clustering problem. Then, given $\{X_1, \ldots, X_N\}$, a sample of N time series, we will construct our dissimilarity matrix as:

$$D_{GCC} = \begin{pmatrix} 0 & 1 - \widehat{GCC}(X_1, X_2) & \cdots & 1 - \widehat{GCC}(X_1, X_N) \\ 1 - \widehat{GCC}(X_2, X_1) & 0 & \cdots & 1 - \widehat{GCC}(X_2, X_N) \\ \vdots & \vdots & \ddots & \vdots \\ 1 - \widehat{GCC}(X_N, X_1) & 1 - \widehat{GCC}(X_N, X_2) & \cdots & 0 \end{pmatrix}, \tag{2.4}$$

where the values $\widehat{GCC}(X_i, X_j)$ are the empirical estimators of the GCC measure between the i-th and j-th series, created applying (2.1) to the time series. The elements of the dissimilarity matrix (2.4) are defined as $1 - \widehat{GCC}(X_i, X_j)$ with the aim of having a large value of dissimilarity when we have independent time series and small values when the series are highly dependent.

2.2. The Cophenetic Distance

When dealing with hierarchical clustering algorithms, cophenetic distances are a very useful tool. These distances allow us to quantify the degree of similarity between observation pairs taking into account the neighbors they have in common within a cluster hierarchy. They are used in clustering algorithms as an easier starting point because they are able to capture more clearly the chain dependence of some groups of time series.

Definition 1. *The cophenetic distance $\widetilde{D}(X_j, X_k)$ between two observations X_j and X_k is defined as*

$$\widetilde{D}(X_j, X_k) = \min_i \max\{D(X_j, X_i), D(X_k, X_i)\},$$

where $D(X_j, X_i)$ is a dissimilarity measure previously defined between X_j and X_i.

Obviously, we will use as $D(X_j, X_i)$ the elements of the GCC dissimilarity matrix (2.4). The algorithm to obtain the cophenetic distance matrix is represented as pseudocode in Algorithm 1:

Algorithm 1 Cophenetic Distance matrix

```
1: Input: Distance matrix $D \in \mathbb{R}^{n \times n}$
2: for $h = 0$ to $n - 1$ do
3:     $CD \leftarrow D$                    ▷ Copy the distance matrix to a new matrix $CD$
4:     for $j = 0$ to $n - 1$ do
5:         for $k = j + 1$ to $n$ do
6:             $CD[j,k] \leftarrow \min(\max(D[[j,k],:], \text{axis} = 0))$
7:             $CD[k,j] \leftarrow CD[j,k]$          ▷ Ensuring the matrix remains symmetric
8:         end for
9:     end for
10:    $D \leftarrow CD$                    ▷ Update $D$ with the newly computed distances
11: end for
12: return $D$                              ▷ Return cophenetic distance matrix
```

The only line of code that can be a bit tricky is line 6, so let us explain it in detail. For each pair of observations X_j and X_k, the submatrix $D[[j,k],:]$ which consists of the rows j and k from the original distance matrix, is computed. Next, we apply the max() function to that submatrix, obtaining a unique vector composed of the maximum value of the columns of the submatrix (this is thanks to the command "axis = 0", if we put "axis = 1", we would obtain a vector composed by the maximum value of the rows of the submatrix). Finally, we obtain the minimum of these maximum values. Doing this we ensure that the smallest maximum distance is chosen, which contains the closest linkage between observations X_j and X_k via any intermediate point.

Let us illustrate this construction in an example. Consider the matrix (2.5), where the first four observations are related to each other by a chain correlation structure, in the same way as the last three observations. Each of the entries in this matrix is assumed to come from the GCC (so we have a dissimilarity matrix with the shape of 2.4):

$$D = \begin{pmatrix} 0 & 0.2 & 1 & 1 & 1 & 1 & 1 \\ 0.2 & 0 & 0.3 & 1 & 1 & 1 & 1 \\ 1 & 0.3 & 0 & 0.4 & 1 & 1 & 1 \\ 1 & 1 & 0.4 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0.75 & 1 \\ 1 & 1 & 1 & 1 & 0.75 & 0 & 0.75 \\ 1 & 1 & 1 & 1 & 1 & 0.75 & 0 \end{pmatrix}. \tag{2.5}$$

Step-by-step illustration for the instance $X_j = X_0$ and $X_k = X_2$:

1. Extracting the submatrix $D[[j,k],:]$:

$$D[[0,2],:] = \begin{pmatrix} D[0,:] \\ D[2,:] \end{pmatrix} = \begin{pmatrix} 0 & 0.2 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0.3 & 0 & 0.4 & 1 & 1 & 1 \end{pmatrix}.$$

2. Computing $\max(D[[0,2],:], \text{axis} = 0)$:

$$\max(D[[0,2],:], \text{axis} = 0) = \begin{pmatrix} 1 & 0.3 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

3. Computing $\min(\max(D[[0,2],:], \text{axis} = 0))$:

$$\min(\max(D[[0,2],:], \text{axis} = 0)) = \min(1, 0.3, 1, 1, 1, 1, 1) = 0.3.$$

The final matrix D_{CD} returned by the algorithm is:

$$D_{CD} = \begin{pmatrix} 0 & 0.2 & \mathbf{0.3} & 0.4 & 1 & 1 & 1 \\ 0.2 & 0 & 0.3 & 0.4 & 1 & 1 & 1 \\ 0.3 & 0.3 & 0 & 0.4 & 1 & 1 & 1 \\ 0.4 & 0.4 & 0.4 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0.75 & 0.75 \\ 1 & 1 & 1 & 1 & 0.75 & 0 & 0.75 \\ 1 & 1 & 1 & 1 & 0.75 & 0.75 & 0 \end{pmatrix},$$

where we can distinguish a much more differentiated cluster structure than in (2.5). There is clearly a chain dependency structure that can be complicated for the correct cluster identification in some procedures. In Figure 2.1 it can be seen how the hierarchical clustering results obtained for D_{CD} reveal much more clearly the differentiation into two groups than the hierarchical clustering results on D.

Summing up, cophenetic distances take into account the larger structural links within the data, in contrast to the GCC measure, which evaluates linear dependencies between pairs of time series directly. They are very useful in identifying pairs of observations that might be indirectly related through a set of intermediate observations. As we will see below, the best fuzzy clustering results obtained in the simulation studies are those in which the initial matrix introduced is that of the cophenetic distances.

Having introduced these two new concepts, let us start defining the three fuzzy clustering modes for the single, complete and average linkages, respectively.

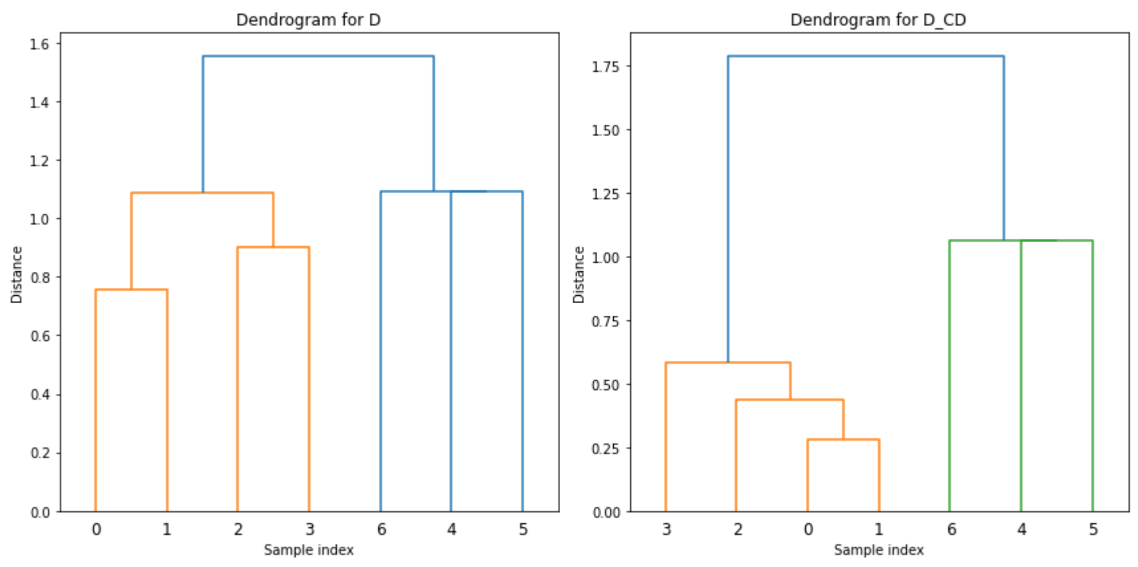


Figure 2.1

Hierarchical clustering results for D and D_{CD} using single linkage

2.3. First Formulation: Single Linkage in Fuzzy Clustering

Let D be a distance (dissimilarity) matrix between n individuals/observations/time series, where $D = (d_{ij})_{i \in I, j \in J}$ and $I, J = \{1, \ldots, n\}$. We assume here that D has been calculated using the procedures in (2.4) or in Algorithm (1). Let $C \in \mathbb{N}$ ($C << n$) be the number of clusters and $u \in [0, 1]$ a given threshold. Let K be the set of indices $K = \{1, \ldots, C\}$.

Our aim is to cluster the n individuals in a fuzzy way using the single linkage to compute the distances between the groups of individuals. In other words, the membership grades $u_{ik} \in [0, 1]$ are to be found by solving a mixed non nonlinear optimization problem.

The objective function (2.6) is the fuzzy clustering objective function for the single linkage:

$$\min \sum_{i \in I} \sum_{k \in K} u_{ik}^m w_{ik}, \tag{2.6}$$

where $m \geq 1$ and w_{ik} is the minimum distance between the i-th individual and all the individuals in cluster k such that its membership grade is at least u, namely

$$w_{ik} = \min\{d_{ij} : u_{jk} \geq u, j \in J, k \in K, j \neq i\}. \tag{2.7}$$

The introduction of the threshold u is a novel aspect of the proposed approach that controls the minimum degree of membership that an individual must have in order to be considered when calculating the w_{ik}. The impact of this definition is significant and varies with the value of u:

- A high value of u implies that only individuals with a strong cluster membership are taken into account when calculating w_{ik}. This makes clustering stricter, as it is based on well-defined memberships. Consequently, the clusters formed tend to be more compact.

- A lower value of u allows individuals with weaker degrees of membership to influence the w_{ik} calculation. This approach is more inclusive and can lead to larger clusters where even less associated individuals contribute to cluster formation. Although this could capture more complex correlations in the data, it might also lead to less distinct clusters that overlap more.

To model this problem as an MINLP, the following variables are defined for $i \in I, j \in J$ and $k \in K$:

$$\delta_{ijk} = \begin{cases} d_{ij} & \text{if } u_{ik} \geq u \\ 0 & \text{otherwise} \end{cases},$$

$$z_{jk} = \begin{cases} 1 & \text{if } u_{ik} \geq u \\ 0 & \text{otherwise} \end{cases},$$

and

$$x_{ijk} = \begin{cases} 1 & \text{if the closest individual to } i \text{ in cluster } k \text{ is individual } j \\ 0 & \text{otherwise} \end{cases}.$$

Using these variables, the problem of finding the membership grades of n individuals to C clusters using the single linkage as a measure of proximity has the following constraints:

1. Firstly, we need to ensure that the sum of membership grades for each individual i across all clusters k equals 1.

$$\sum_{k \in K} u_{ik} = 1 \quad \forall i \in I. \tag{2.8}$$

2. Next, all individuals and clusters should have membership grades ranging from 0 to 1. This requirement ensures that the membership grade is an appropriate proportion.

$$0 \leq u_{ik} \leq 1 \quad \forall i \in I, k \in K. \tag{2.9}$$

3. To ensure that if $z_{jk} = 0$, then $u_{jk} < u$, we impose the following constraint. In this case, a parameter to guarantee strict inequality is $\epsilon > 0$.

$$u_{jk} \leq (u - \epsilon)(1 - z_{jk}) + z_{jk} \quad \forall j \in J, k \in K. \tag{2.10}$$

4. Similarly, we impose the following constraint to guarantee that if $z_{jk} = 1$, then $u_{jk} \geq u$.

$$u_{jk} \geq u z_{jk} \quad \forall j \in J, k \in K. \tag{2.11}$$

5. In each cluster k, there should be exactly one nearest individual j for each individual i. This constraint ensures that there is only one observation j which is at the minimum distance to the observation i in cluster k.
$$\sum_{j \in J, j \neq i} x_{ijk} = 1 \quad \forall i \in I, k \in K. \tag{2.12}$$

6. The variable δ_{ijk} is defined to be d_{ij} if $u_{jk} \geq u$ and $\delta = \max\{d_{ij} : i \in I, j \in J\}$ otherwise.
$$\delta_{ijk} = d_{ij} z_{jk} + \delta(1 - z_{jk}) \quad \forall i \in I, j \in J, k \in K. \tag{2.13}$$

7. The combination of the following constraints ensures that the value of w_{ik} corresponds to the definition in (2.7), where M is a large enough constant.
$$w_{ik} \leq \delta_{ijk} \quad \forall i \in I, j \in J, k \in K, j \neq i \tag{2.14}$$
$$w_{ik} \geq \delta_{ijk} - M(1 - x_{ijk}) \quad \forall i \in I, j \in J, k \in K, j \neq i. \tag{2.15}$$

8. Finally, the non-negativity and binary constraints are imposed on the variables.
$$w_{ik} \geq 0, \quad z_{jk}, x_{ijk} \in \{0, 1\} \quad \forall i \in I, j \in J, k \in K. \tag{2.16}$$

Summing up, the resulting MINLP is:

$$\begin{array}{lll}
\min & \displaystyle\sum_{i \in I} \sum_{k \in K} u_{ik}^m w_{ik} & \\
\text{s.t.} & \displaystyle\sum_{k \in K} u_{ik} = 1 & i \in I \\
& 0 \leq u_{ik} \leq 1 & i \in I, k \in K \\
& u_{jk} \leq (u - \epsilon)(1 - z_{jk}) + z_{jk} & j \in J, k \in K \\
& u_{jk} \geq u z_{jk} & j \in J, k \in K \\
& \displaystyle\sum_{j \in J, j \neq i} x_{ijk} = 1 & i \in I, k \in K \\
& \delta_{ijk} = d_{ij} z_{jk} + \delta(1 - z_{jk}) & i \in I, j \in J, k \in K \\
& w_{ik} \leq \delta_{ijk} & i \in I, j \in J, k \in K, j \neq i \\
& w_{ik} \geq \delta_{ijk} - M(1 - x_{ijk}) & i \in I, j \in J, k \in K, j \neq i \\
& w_{ik} \geq 0, \quad z_{jk}, x_{ijk} \in \{0, 1\} & i \in I, j \in J, k \in K
\end{array} \tag{2.17}$$

2.3.1. Modification of the Formulation

A second simpler formulation, where the variables δ_{ijk} are eliminated, is introduced:

$$
\begin{array}{lll}
\min & \displaystyle\sum_{i\in I}\sum_{k\in K} u_{ik}^m w_{ik} & \\
\text{s.t.} & \displaystyle\sum_{k\in K} u_{ik} = 1 & i \in I \\
& 0 \le u_{ik} \le 1 & i \in I, k \in K \\
& u_{jk} \le (u-\epsilon)(1-z_{jk}) + z_{jk} & j \in J, k \in K \\
& u_{jk} \ge (u+\epsilon) z_{jk} & j \in J, k \in K \\
& \displaystyle\sum_{j\in J} z_{jk} \ge 1 & k \in K \\
& \displaystyle\sum_{j\in J, j\neq i} x_{ijk} = 1 & i \in I, k \in K \\
& w_{ik} \le d_{ij} + M(1-x_{ijk}) & i \in I, j \in J, k \in K, j \neq i \\
& w_{ik} \ge d_{ij} x_{ijk} & i \in I, j \in J, k \in K, j \neq i \\
& x_{ijk} \le z_{jk} & i \in I, j \in J, k \in K, j \neq i \\
& w_{ik} \ge 0, \quad z_{jk}, x_{ijk} \in \{0,1\} & i \in I, j \in J, k \in K
\end{array}
\tag{2.18}
$$

In the new problem, the variables δ_{ijk} are omitted for simplicity, and the correct definition of the variables w_{ik} is obtained using the constraints

$$w_{ik} \le d_{ij} + M(1-x_{ijk}) \qquad i \in I, j \in J, k \in K, j \neq i \tag{2.19}$$

and

$$w_{ik} \ge d_{ij} x_{ijk} \qquad i \in I, j \in J, k \in K, j \neq i. \tag{2.20}$$

In addition, the constraint

$$\sum_{j\in J} z_{jk} \ge 1 \quad k \in K \tag{2.21}$$

is introduced, whereby there must be at least one observation in each cluster such that its membership degree is greater than the threshold u, since there were times when the optimal cluster classification ignored one group.

When these two formulations were tested computationally we obtain very similar and poor results. In what follows, we analyze why this happens and why using single linkage as considered in (2.7) is not appropriate for fuzzy clustering.

2.3.2. Problem with the Single Linkage

To illustrate the problem, the dissimilarity matrix of the seventh scenario in chapter 4 is going to be considered.

$$D_7 = \begin{bmatrix}
0 & 0.19 & 0.19 & 0.19 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0.19 & 0 & 0.19 & 0.19 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0.19 & 0.19 & 0 & 0.19 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0.19 & 0.19 & 0.19 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0.19 & 0.19 & 0.19 & 0.19 & 0.19 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0.19 & 0 & 0.19 & 0.19 & 0.19 & 0.19 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0.19 & 0.19 & 0 & 0.19 & 0.19 & 0.19 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0.19 & 0.19 & 0.19 & 0 & 0.19 & 0.19 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0.19 & 0.19 & 0.19 & 0.19 & 0 & 0.19 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0.19 & 0.19 & 0.19 & 0.19 & 0.19 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0.19 & 0.19 & 0.19 & 0.19 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0.19 & 0 & 0.19 & 0.19 & 0.19 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0.19 & 0.19 & 0 & 0.19 & 0.19 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0.19 & 0.19 & 0.19 & 0 & 0.19 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0.19 & 0.19 & 0.19 & 0.19 & 0
\end{bmatrix},$$

It can clearly be seen that the three clusters have a strong dependency structure among the four, six and five time series, respectively.

Let us denote $u_i = (u_{i1},\ u_{i2},\ u_{i3})$ the three membership grades of the individual i to the three clusters, and $w_i = (w_{i1},\ w_{i2},\ w_{i3})$ contains the minimum distances between the i-th individual and all the individuals in each cluster k such that its membership grade is at least u. Then, for the parameters $u = 0.6$ and $m = 2$, and the idea of the single linkage as measured in 2.7, a solution that will provide us with a perfect fuzzy clustering of the observations would be

$$u_i = \begin{cases} (0.6, 0.2, 0.2) & \text{if} \quad i \in \{1, 2, 3, 4\} \\ (0.2, 0.6, 0.2) & \text{if} \quad i \in \{5, 6, 7, 8, 9, 10\} \\ (0.2, 0.2, 0.6) & \text{if} \quad i \in \{11, 12, 13, 14, 15\} \end{cases}$$

with

$$w_i = \begin{cases} (0.19, 1, 1) & \text{if} \quad i \in \{1, 2, 3, 4\} \\ (1, 0.19, 1) & \text{if} \quad i \in \{5, 6, 7, 8, 9, 10\} \\ (1, 1, 0.19) & \text{if} \quad i \in \{11, 12, 13, 14, 15\} \end{cases}$$

The corresponding objective function is

$$\sum_{i=1}^{15} \sum_{k=1}^{3} u_{ik}^2 w_{ik} = 15 \cdot 3 \cdot (0.6^2 \cdot 0.19 + 0.2^2 \cdot 1 + 0.2^2 \cdot 1) = 6.678.$$

This distribution of this solution is reflected in the Figure 2.2, where there are three clusters, the observations of each cluster are at distance (w_{ik}) 0.19 from the observations of the same cluster, and at distance 1 from the observations of different

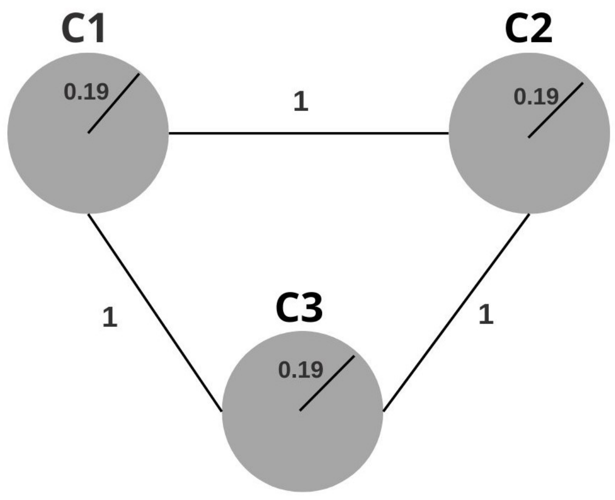


Figure 2.2
Ideal clustering distribution for Scenario 7.

clusters.

But, is this solution a minimum of our problem? The answer is no. Suppose we move an observation from cluster 2 to cluster 1 and another to cluster 3.

Now for an observation i that is in the second cluster, by the definition of w_{ik}, (if we remember it: the minimum distance between the i-th individual and all the individuals in cluster k such that its membership grade is at least u) we got that its part in the objective function is:

$$\sum_{k=1}^{3} u_{ik}^2 w_{ik} = u_{i1}^2 \cdot 0.19 + u_{i2}^2 \cdot 0.19 + u_{i3}^2 \cdot 0.19 = (u_{i1}^2 + u_{i2}^2 + u_{i3}^2) \cdot 0.19,$$

and if we use the same degrees of membership of the previous solution we have that:

$$(0.6^2 + 0.2^2 + 0.2^2) \cdot 0.19 = 0.0836 < 0.1484 = 0.6^2 \cdot 0.19 + 0.2^2 \cdot 1 + 0.2^2 \cdot 1.$$

So the optimization models 2.17 and 2.18, instead of considering the first solution optimal, incorrectly moves individuals belonging to one cluster to another in order to decrease this value of w_{ik} and thus decrease the objective function, but giving an undesirable solution. So we have just shown that the ideal clustering solution is not optimal due to the use of the single linkage.

The same happens when only one observation is moved from one cluster to another. Suppose we move only one observation from cluster 2 to cluster 1. For an observation i that is in the second cluster, we get its part in the objective function:

$$\sum_{k=1}^{3} u_{ik}^2 w_{ik} = u_{i1}^2 \cdot 0.19 + u_{i2}^2 \cdot 0.19 + u_{i3}^2 \cdot 1,$$

and if we use again the same values of the degrees of membership of the first solution proposed we have that:

$$(0.6^2 + 0.2^2) \cdot 0.19 + 0.2^2 \cdot 1 = 0.116,$$

which is still smaller than the value of the "good" proposed solution.

To solve this problem, in the next section, the complete and average linkages are considered instead.

2.4. Second and Third Formulation: Complete and Average Linkage in Fuzzy clustering

The new problem is a modification of (2.18), here we change the way in which the distance between groups of observations is calculated, until now we have used the single linkage, and now we will switch to using the complete linkage (and also for the average linkage separately).

So the measure w_{ik} in (2.7) is now defined as

$$w_{ik} = \max\{d_{ij} : u_{jk} \geq u, j \in J, k \in K, j \neq i\}, \tag{2.22}$$

for complete linkage problem, and

$$w_{ik} = \frac{\sum_{j \in J, j \neq i} d_{ij} \cdot z_{jk}}{\sum_{j \in J, j \neq i} z_{jk}}, \tag{2.23}$$

for average linkage problem.

The expressions (2.24) and (2.25) are the constraints needed to define appropriately the complete and the average linkage, respectively.

- Complete linkage:

$$w_{ik} \geq d_{ij} \cdot z_{jk}, \quad \forall i, j \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, C\}, \; j \neq i. \tag{2.24}$$

- Average linkage:

$$w_{ik} \cdot \sum_{\substack{j=1 \\ j \neq i}}^{n} z_{jk} = \sum_{\substack{j=1 \\ j \neq i}}^{n} d_{ij} \cdot z_{jk}, \quad \forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, C\}. \tag{2.25}$$

Also the following constraint has been introduced in order to ensure that each observation i has at least one cluster k with a high membership value (greater than u).

$$\sum_{k=1}^{C} z_{jk} \geq 1, \quad \forall j \in \{1, \ldots, n\}. \tag{2.26}$$

The final formulation of the problem is (2.27 the problem with the complete link and then 2.28 with the average link):

$$\begin{aligned}
\min \quad & \sum_{i \in I} \sum_{k \in K} u_{ik}^m w_{ik} & & \\
\text{s.t.} \quad & \sum_{k \in K} u_{ik} = 1 & & i \in I \\
& 0 \le u_{ik} \le 1 & & i \in I, k \in K \\
& u_{jk} \le (u - \epsilon)(1 - z_{jk}) + z_{jk} & & j \in J, k \in K \\
& u_{jk} \ge (u + \epsilon) z_{jk} & & j \in J, k \in K \\
& \sum_{j \in J} z_{jk} \ge 1 & & k \in K \\
& \sum_{j \in J, j \ne i} x_{ijk} = 1 & & i \in I, k \in K \\
& x_{ijk} \le z_{jk} & & i \in I, j \in J, k \in K, j \ne i \\
& \sum_{k \in K} z_{jk} \ge 1 & & j \in J \\
& w_{ik} \ge d_{ij} \cdot z_{jk} & & i \in I, j \in J, k \in K, j \ne i \\
& w_{ik} \ge 0, \quad z_{jk}, x_{ijk} \in \{0,1\} & & i \in I, j \in J, k \in K
\end{aligned} \tag{2.27}$$

$$\begin{aligned}
\min \quad & \sum_{i \in I} \sum_{k \in K} u_{ik}^m w_{ik} & & \\
\text{s.t.} \quad & \sum_{k \in K} u_{ik} = 1 & & i \in I \\
& 0 \le u_{ik} \le 1 & & i \in I, k \in K \\
& u_{jk} \le (u - \epsilon)(1 - z_{jk}) + z_{jk} & & j \in J, k \in K \\
& u_{jk} \ge (u + \epsilon) z_{jk} & & j \in J, k \in K \\
& \sum_{j \in J} z_{jk} \ge 1 & & k \in K \\
& \sum_{j \in J, j \ne i} x_{ijk} = 1 & & i \in I, k \in K \\
& x_{ijk} \le z_{jk} & & i \in I, j \in J, k \in K, j \ne i \\
& \sum_{k \in K} z_{jk} \ge 1 & & j \in J \\
& w_{ik} \cdot \sum_{\substack{j=1 \\ j \ne i}}^{n} z_{jk} = \sum_{\substack{j=1 \\ j \ne i}}^{n} d_{ij} \cdot z_{jk} & & i \in I, k \in K \\
& w_{ik} \ge 0, \quad z_{jk}, x_{ijk} \in \{0,1\} & & i \in I, j \in J, k \in K
\end{aligned} \tag{2.28}$$

Before starting to test all the models proposed so far in different directed scenarios, using both the GCC measure and the cophenetic distances to generate the dissimilarity matrices introduced as input, we will introduce the metaheuristic Large

Neighbourhood Search (LNS), which will allow u to find good quality solutions to the above proposed formulations in a reasonable computational time instead of putting them directly into a solver.

3. THE METAHEURISTIC LARGE NEIGHBORHOOD SEARCH

Large Neighborhood Search (LNS) is a metaheuristic technique proposed by Shaw, 1998, mostly used to solve challenging combinatorial optimization problems. This search algorithm falls within the larger category of Very Large Scale Neighborhood (VLSN) search algorithms. The fundamental principle of LNS is to repeatedly destroy and repair solutions in order to improve them iteratively. Finding excellent local optima is made more likely by this method's ability to explore the set of feasible solutions.

The main reason for implementing this metaheuristic algorithm is to reduce the computational times of the models defined in the previous chapter, which, as we will see in Chapter 4, are too high.

3.1. Basic Concepts and Definitions

Given a minimization optimization problem, which can be stated as:

$$\min \quad o(x) \quad \text{subject to} \quad x \in X, \tag{3.1}$$

where:

- X is the set of all feasible solutions. Recall that a solution is feasible if it satisfies all the constraints of the problem.
- $o : X \to \mathbb{R}$ is a function that assigns a cost to each solution, the objective function.

Our goal is to find a solution x' such that $o(x') \leq o(x) \; \forall x \in X$.

Now, we introduce the concept of **neighborhood**. For any given solution $x \in X$, a neighborhood $N(x)$ is a subset of X. Formally, N is a function that maps a solution to a set of solutions and in simpler terms, $N(x)$ contains solutions that are close to x according to some criteria.

A solution x is said to be **locally optimal** with respect to a neighborhood N if $o(x) \leq o(x') \; \forall x' \in N(x)$, i.e, there is no better solution within the neighborhood of x.

A neighborhood search algorithm can be stated using these definitions. The algorithm uses as input an initial solution x, then, it finds the best solution x^* in the neighborhood of x, in other words, it computes $x^* = \text{argmin}_{x^{**} \in N(x)}\{o(x^{**})\}$. If the value objective function of the new solution found is lower than the value of the old one ($o(x^*) < o(x)$), the update $x = x^*$ is carried out. The neighborhood of the new updated solution is searched for improve the current solution and this procedure is repeated until a stopping criterion, such as a maximum number of iterations, is reached.

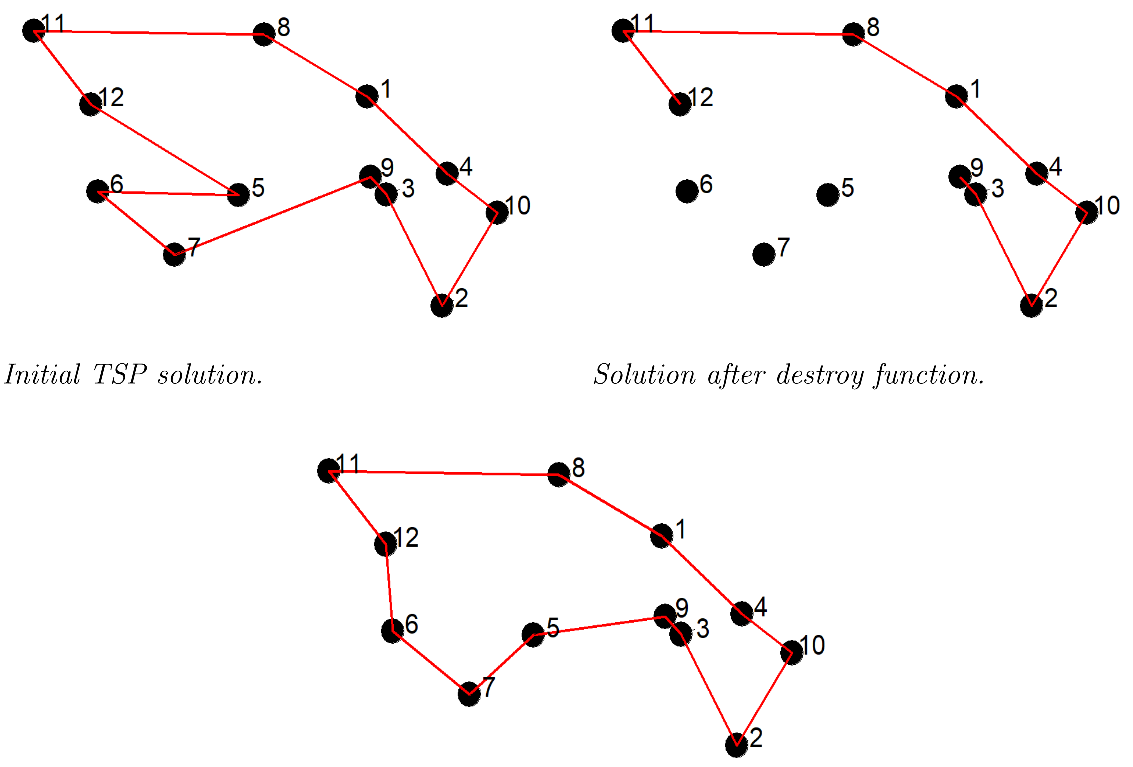
As mentioned above, the LNS belongs to the class of VLSN algorithms. Ahuja et al., 2002 defined that an algorithm belongs to this family if the neighborhoods searched during the algorithm grow exponentially with the instance size or if the neighborhood is too large to be searched explicitly in practice. It makes sense that looking through a large neighborhood would get better results than looking through a tiny one. However, in practice small neighborhoods which can be explored more quickly, might offer equivalent or better quality solutions if they are placed in a metaheuristic framework. Examples of such behavior are found in Bent and Van Hentenryck, 2006. This shows that VSLN algorithms are not "magic", but that they can deliver very good results for the right applications. Now let us focus on the Large Neighborhood Search algorithm.

3.2. Large Neighborhood Search

In the LNS metaheuristic, the neighborhood is defined implicitly by a **destroy** and a **repair** methods. On the one hand, the destroy method destroys part of the solution, it usually contains a stochastic component in order to eliminate different parts of the solution each time it is called. On the other hand, the repair method rebuilds the destroyed solution. Then, the neighborhood $N(x)$ of a solution x is defined as the set of solutions that can be obtained by applying the destroy method first, followed by the repair method.

To visually illustrate the methods of destruction and repair, let us consider as an example the Travelling Salesman Problem (TSP), where a salesman has to travel through a set of cities (nodes), with the constraints that she visits each city only once and that she starts and ends in the same city. The objective is to minimise the distance travelled (edge costs). An instance of the problem can be found in Figure 3.1, where labels represent cities, and city x is not necessarily closest to city $x + 1$. The top left figure shows a TSP solution (with 12 nodes) that is not optimal before applying the destroy function. The top right figure shows the solution without some edges that have been deleted. The bottom figure shows the solution after the repair function has reinserted the tour between the cities. This figure has been done in Rstudio.

Now we present the LNS in more detail. The pseudocode for the algorithm is shown in Algorithm 2, where the functions $r()$ and $d()$ refers to the repair and destroy methods, respectively. The algorithm can be found in Pisinger and Ropke,



Initial TSP solution. *Solution after destroy function.*

Solution after repair function.

Figure 3.1
Example of destroy and repair procedures for a TSP instance.

2010.

The variable x_b refers to the best solution found during the search, x is the current solution, and x_t is a temporary solution that can be discarded or used as the new current solution x if some acceptance criterion is met. With the repair function $r()$, we obtain a feasible solution from $d(x)$, which is basically a copy of x which is partly destroyed. In line 1, a feasible solution x is introduced to start the main loop and in line 2 the best solution is also initialised. From line 3 to line 10 is the main loop, where in each iteration a new temporary solution x_t is generated by applying the neighborhood move (the destroy and repair functions to x). In line 5 and 6 the new solution is subjected to an acceptance process (function accept(x_t,x)) which is: x_t is always accepted if $o(x_t) \leq o(x)$ and is accepted with probability $exp\{-(o(x_t) - o(x))/T\}$ if $o(x) < o(x_t)$, where T is initialised with a $T_0 > 0$ and is gradually decreased with iterations to avoid accepting too many solutions at advanced iterations of the algorithm.

3.3. Defining our Algorithm

In this section, we will define how we create the initial feasible solution, and the destroy and repair method in order to solve the fuzzy clustering problem using the

Algorithm 2 Large Neighborhood Search

```
1: **Input:** A feasible solution $x$
2: $x_b = x$
3: **repeat**
4:     $x_t = r(d(x))$
5:     **if** accept($x_t$, $x$) **then**
6:         $x = x_t$
7:     **end if**
8:     **if** $o(x_t) < o(x_b)$ **then**
9:         $x_b = x_t$
10:    **end if**
11: **until** stop criterion is met
12: **return** $x_b$
```

formulations (2.18), (2.27) and (2.28).

In order to create a feasible solution to start with the algorithm, we plug the formulation into an off-the-shelf optimization solver and take the best solution found after some seconds of execution, which in general is far to be optimal. Other options for obtaining a first initial solution may also be valid, for example, using the fuzzy c-means algorithm or simply running a hard clustering, taking that solution and adapting the degrees of membership to make it feasible based on threshold u.

The most important part of defining the destroy method is the percentage of destruction: if the percentage of destruction is small and only a small part of the solution is eliminated, we may have problems exploring the search space as the effect of the large neighborhood may be lost, and if it is too large, then the LNS almost degrades to repeated optimization. The percentage of destruction is applied to the total number of variables u_{ik} in the problem.

In our case, the destroy function is very simple. Given the constraint (2.8), whereby the sum of the membership degrees of any individual has to be 1, if we were to adjust the destroy function to randomly choose a subset of u_{ik} indices, it is possible that sometimes only one of the K indices of an individual i would be eliminated, and considering that the repair function will give us feasible solutions again, the constraint (2.8) would make this eliminated value the same again, so that the sum of its degree memberships is 1. To avoid this and to be able to search for new solutions in a larger search space, the destroy function has been defined in such a way that if randomly the index $(i,\ k)$ of u_{ik} is chosen to be destroyed, then at least another index $k' \in K$ of the same individual i will be chosen too, destroying the respective $u_{ik'}$. So on and so forth until the destruction percentage quota is filled. Once all the elements u_{ik} have been destroyed, the associated w_{ik}, x_{ijk} and z_{jk} values are also eliminated. When we refer to delete, what we actually say is that they become decision variables in the optimisation model while those that are not destroyed become fixed at the value of the current best found solution.

For the repair function one has a great deal of freedom in defining it. On the one hand, you can choose a method that is optimal in the sense that it constructs the optimal solution using the partial solution that you have after applying the destruction method. On the other hand, you can create a heuristic and assume that you are happy with a "good" solution, rather than the optimal one. In our case, we have chosen the optimal repair, which is slower than the heuristic, but with it we are more likely to reach more powerful solutions in a few iterations.

Our repair function starts by fixing all the variables that have not been destroyed in the previous step and then the “partial” optimization problem is solved. Once the solution of this “partial” problem is obtained, we unfix the variables that were fixed previously for the start of the next iteration. So this repair method tries to improve the current solution while respecting the fixed values.

To fully understand our destruction and repair functions, let us look at a visual example. Suppose that we have nine individuals that are organised in three clusters and that the optimal solution would be to classify the first three individuals in one cluster, the next three in another, and the last three in the last cluster. Then, in this context, the shape of our final solution (membership grades) is displayed in Figure 3.2:

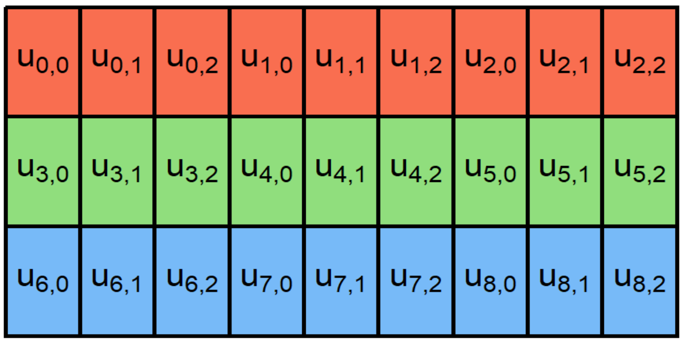
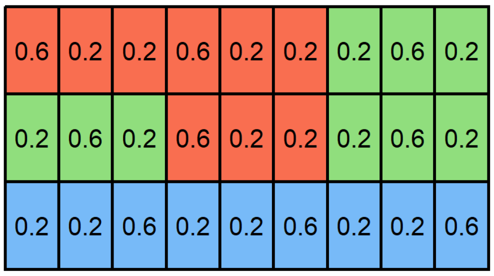

$u_{0,0}$ $u_{0,1}$ $u_{0,2}$ $u_{1,0}$ $u_{1,1}$ $u_{1,2}$ $u_{2,0}$ $u_{2,1}$ $u_{2,2}$
$u_{3,0}$ $u_{3,1}$ $u_{3,2}$ $u_{4,0}$ $u_{4,1}$ $u_{4,2}$ $u_{5,0}$ $u_{5,1}$ $u_{5,2}$
$u_{6,0}$ $u_{6,1}$ $u_{6,2}$ $u_{7,0}$ $u_{7,1}$ $u_{7,2}$ $u_{8,0}$ $u_{8,1}$ $u_{8,2}$

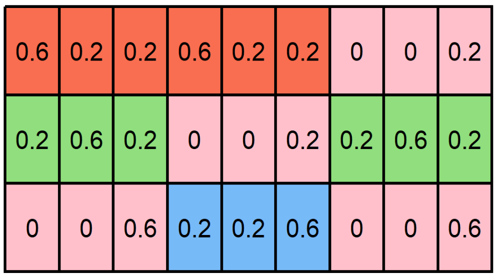
Figure 3.2
Structure of the individual in our small example to explain the functions of destroying and repairing.

The red color refers to individuals in the first cluster, the green color to individuals in the second cluster and the blue color to individuals in the last cluster. An instance of the LNS algorithm with this small example is the Figure 3.3. The 0 cells represent the u_{ik} that have been affected by the destroy function and the pink cells represent that the individuals (u_{i0},u_{i1},u_{i2}) affected do not belong to any cluster at that moment. This figure has been done in Rstudio.

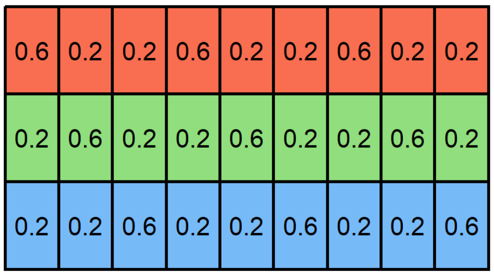
In the simulation exercises of Chapter 4, this algorithm will be tested in the same scenarios in which the models created in chapter 2 are tested to see if it meets our objective of improving computational time and obtaining good cluster



Current solution.



Solution after destroy function.



Solution after repair function.

Figure 3.3
Example of destroy and repair procedures in the context of our problem.

classifications. It should be mentioned that only the metaheuristic algorithm based on the complete problem formulation (2.27) is tested in the simulation study because this formulation provides the best fuzzy clustering results, but it should be noted that this LNS approach can be used for all formulations, single (2.18), complete (2.27) and average (2.28).

4. SIMULATION STUDY

In this chapter, a simulation study will be carried out using Monte Carlo experiments to test the three models developed so far.

Practically all the code necessary for the implementation of the models and the Monte Carlo experiments has been developed in Python (version 3.11) using the Gurobi solver Gurobi Optimization, LLC, 2023 (Rstudio was used only for the evaluation of two metrics that appear in the first table and the creation of some graphics) and a PC Lenovo Legion Y530-15ICH with 8 GB of RAM has been used. The evaluation metric used throughout the chapter will be the Adjusted Rand Score (ARI) (Hubert, 1985), although the Separation Index (SI) and the Normalized Gamma index (NGI) are also used at the beginning. These last two indices have been calculated in Rstudio with the library FPC (Hennig, 2010) while the ARI has been calculated in Python, using the sklearn library (Pedregosa et al., 2011). For more information on the metrics check Alonso et al., 2021.

The number of repetitions in the Monte Carlo experiments will be R=100 for the Subsection 4.1 and R=5 for the Subsection 4.2.2 due to the high computational time it takes to run the algorithm in these scenarios. The values of the model parameters are $m = 2$, which makes the models mixed integer quadratic, and therefore the Gurobi solver is used, M is the maximum value of the dissimilarity matrix that is entered as input and ϵ=0.001. On the other hand, to calculate the dissimilarity matrix using the GCC (2.1), we use $k = 0$. Finally, in the simulations using the LNS the percentage of destruction used have been 0.30 for section 4.1.3 and 0.15 for section 4.2.2.

Another thing to keep in mind is that a parameter is used to set a time limit for the algorithms, when this time limit is completed, the algorithm returns the best solution found so far. The time limit parameter used in the experiments of Subsection 4.1 has been 100 seconds, while the one used in the experiments of Subsection 4.2 has been 10800 seconds (3 hours). These parameters have been set after several tests in order to obtain good solutions in a reasonable time.

4.1. Montecarlo Experiments on Time Series Clustering: Cross-Dependent Scenarios

The different approaches presented in this work are tested in seven different small scenarios, we focus on groups of dependent time series with the primary objective of clustering them based on their cross-dependence. We analyze a set of fifteen time series. To construct the dissimilarity matrices, we use the formula (2.1) with $k = 0$, the simplest case, so:

$$1 - \mathrm{GCC}_{X,Y}(0) = 1 - \rho_{XY}(0)^2 := 1 - \rho(X,Y)^2,$$

where X and Y are two time series and $\rho_{XY}(0)$ is their cross-correlation at lag 0.

The seven scenarios proposed in Alonso et al., 2021 are now presented, each with three clusters characterized by specific non-null cross-correlations. All other possible cross-correlations are assumed to be zero.

1. **Scenario S1**:

   $$\rho(i, i+1) = 0.5 \text{ for } i = 1, \dots, 3 \text{ and } i = 5, \dots, 9.$$

   Four and six time series with chained dependencies make up the first and second clusters, respectively. The third cluster contains five independent time series.

2. **Scenario S2**:

   $$\rho(i, i+1) = 0.5 \text{ for } i = 1, \dots, 4, i = 5, \dots, 9 \text{ and } i = 11, \dots, 14.$$

   There are four, six, and five time series with chained dependencies in each of the three clusters.

3. **Scenario S3**:

   $$\rho(i, i+1) = 0.5 \text{ for } i = 1, \dots, 3,$$

   and

   $$\rho(i, j) = 0.9 \text{ for } i = 5, \dots, 9 \text{ and } j = i+1, \dots, 10.$$

   There are four time series with chained dependencies in the first cluster, six time series with strong dependencies in the second cluster, and five independent time series in the third cluster.

4. **Scenario S4**:

   $$\rho(i, i+1) = 0.5 \text{ for } i = 1, \dots, 3 \text{ and } i = 11, \dots, 15,$$

   and

   $$\rho(i, j) = 0.9 \text{ for } i = 5, \dots, 9 \text{ and } j = i+1, \dots, 10.$$

   There are four time series with chained dependencies in the first cluster and five in the third. The second cluster exhibits strong dependencies among its six time series.

5. **Scenario S5**:

   $$\rho(i, j) = 0.9 \text{ for } i = 1, \dots, 3 \text{ and } j = i+1, \dots, 4 \text{ and } i = 5, \dots, 9$$
   $$\text{and } j = i+1, \dots, 10.$$

   The four and six time series of the first and second clusters, respectively, show strong relationships. The third cluster consists of five independent time series.

6. **Scenario S6**:

$$\rho(i,j) = 0.9 \text{ for } i = 1, \ldots, 3 \text{ and } j = i+1, \ldots, 4 \text{ and for } i = 5, \ldots, 9$$
$$\text{and } j = i+1, \ldots, 10,$$

   and

$$\rho(i, i+1) = 0.5 \text{ for } i = 11, \ldots, 14.$$

   The four and six time series of the first and second clusters, respectively, show strong relationships. The third cluster has five time series with chained dependencies.

7. **Scenario S7**:

$$\rho(i,j) = 0.9 \text{ for } i = 1, \ldots, 3 \text{ and } j = i+1, \ldots, 4 \text{ and for } i = 5, \ldots, 9$$
$$\text{and } j = i+1, \ldots, 10 \text{ and for } i = 11, \ldots, 14 \text{ and } j = i+1, \ldots, 15.$$

   There are strong relationships between the four, six, and five time series in each of the three clusters.

The seven scenarios are illustrate in Figure 4.1, created in Rstudio using igraph package (Csardi and Nepusz, 2006). The series are represented as nodes in the ellipse. When two nodes are connected by a line, there is a non-null cross-correlation between the two series. The proposed methodology is expected to be able to obtain optimal cluster classifications in all scenarios, but especially in the latter ones, where the dependence between the series of each cluster is stronger.

4.1.1. Results using Single Linkage Model

In Table 4.1, the Adjusted Rand Index (ARI), The Separation Index (SI) and the Normalized Γ index (NGI) are reported for the first four scenarios for different values of threshold u and for $C = 3$ clusters. To obtain these values, we introduce the formulation (2.18) of the model with the single linkage in Gurobi. This time we use as input the matrices created with the GCC measure. The results are not as expected, in fact, as the scenario increases, i.e. the more dependency between the cluster series, the worse the results are. Scenarios 5, 6 and 7 have not been reported because ARI values close to zero were obtained, concluding that the classification is almost randomly done. These results were what led us to think that something in the formulation of the problem with the single linkage was wrong, and after some digging to find out what the problem was, it was found and reported in the Section 2.3.2.

We could recreate the same table but introducing as input the matrices of the different scenarios by applying the cophenetic distance, but the results obtained by doing this are just as bad for the reason explained above, so let us move on to the results of the complete and average linkage models.

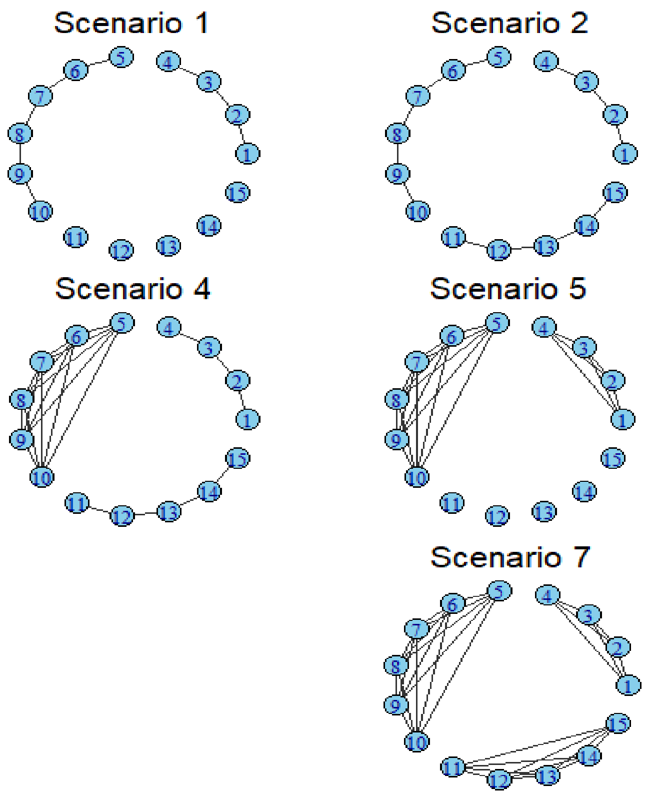


Figure 4.1
Dependency structure illustration of scenarios S.1–S.7.

u	Measure	Scenario			
		S1	S2	S3	S4
0.5	ARI	0.5325	0.1761	0.5424	0.0125
	SI	0.8750	0.7500	0.1900	0.1900
	NGI	0.6654	0.6462	0.5424	0.4398
0.55	ARI	0.5956	0.2811	0.5961	0.2654
	SI	0.7500	0.7500	0.1900	0.1900
	NGI	0.6632	0.6685	0.4038	0.4795
0.6	ARI	0.7711	0.4021	0.3499	0.0633
	SI	0.7500	0.7500	0.1900	0.1900
	NGI	0.6825	0.6548	0.4519	0.4484

Table 4.1
Validation measures of the resulting clustering in different scenarios with different values of u, using the model (2.18) with the dissimilarities matrices created using the GCC measure.

4.1.2. Results using Complete and Average Linkage Models

On this case, only the scenarios where there are no independent series have been used, i.e. scenarios 2, 4, 6 and 7. Again, in this section we are using the Gurobi solver with the formulations 2.27 and 2.28. Furthermore, only the ARI has been used as an evaluation metric because it is easy to interpret and obtain, and it should be noted that all the tables have a row that also shows the ARI obtained using the model in the previous section, in order to compare the three types of model (single, average and complete).

Table 4.2 shows the ARI of the resulting cluster classification for scenario 7, using the GCC measure, the three types of linkage, different values of u, and also setting the number of clusters at the start of the algorithm as 2, 3 and 4, with three being the actual number of clusters (this is done with the intention of seeing how the algorithm works when we do not have the correct information on the number of clusters). With the complete linkage the results for C=3 are perfect and with the average linkages, almost perfect, while for C=2 and 4, lower ARI values are obtained (as expected since the algorithm cannot return the optimal classification), in some cases quite good, and always better than those of the single linkage.

On the other hand, for Tables 4.3, 4.4 and 4.5, the value of $u = 0.6$ has always been used, since the results in the previous table are very similar for all considered u-values. In addition, the results using as initial dissimilarity matrix the cophenetic distances have been introduced.

Turning to these last four tables, we can conclude several things:

- The complete and average linkage always outperform the single linkage.
- Bearing in mind that in the cases C=2 and C=4, the classifications can never be perfect, the ARI value obtained in the cases of complete and average linkages is quite high for Tables 4.3 and 4.4, both for the cophenetic distance and for the GCC.
- Focusing on the C=3 and the complete and average linkages, the cluster classifications obtained for scenario 6 (Table 4.3) and scenario 4 (Table 4.4) using the cophenetic distance have been perfect, while for scenario 2 (Table 4.5), the most diffuse, a very high ARI (0.7920 and 0.7711) is obtained. On the other hand, the results using the GCC get worse as the chain dependence increases, from a perfect classification in scenario 6 (4.3), to an ARI of 0.0799 in scenario 2 (4.5).
- Another thing to note is that the computational time using the average linkage is much longer than using the complete linkage. As mentioned above, the *TimeLimit* parameter used has been 100 seconds, but good solutions are obtained earlier in most of the cases using the complete linkage model, something that does not happen using the average.
- In order to improve the time it takes for the algorithm to return a good classification, before introducing the metaheuristic algorithm, an attempt has been

made to direct the search for solutions by introducing the fuzzy c-means clustering solution (first proposed by Dunn, 1973) as the initial solution (using the Python package fcmeans, Dias, 2019). Very similar solutions are obtained, a little better in some cases, but the time it takes to give a good solution is the same. So introducing initial solutions to reduce computational time has been discarded.

- Comparing these results with the ARI tables in Alonso et al., 2021 (only looking at the results for C=3, which is the only number of clusters used in the article) we can conclude that with the models using the complete and average linkage together with the cophenetic distance we obtain better or equal ARI values for scenarios 4, 6 and 7 and somewhat lower for scenario 2. Something similar happens if we look at the results of the complete and average models but now using the GCC, we obtain better or equal values of ARI for scenarios 6 and 7, while worse for scenarios 4 and 2, which are the ones with more chain dependence.

Linkage	u	$C = 2$	$C = 3$	$C = 4$
Single	$u = 0.55$	0.0650	0.0650	0.0650
	$u = 0.60$	0.0650	0.0650	0.0650
	$u = 0.65$	0.0650	0.0605	0.0650
Complete	$u = 0.55$	0.5502	1.0000	0.7311
	$u = 0.60$	0.6002	1.0000	0.8052
	$u = 0.65$	0.6101	1.0000	0.7051
Average	$u = 0.55$	0.2500	0.9499	0.5805
	$u = 0.60$	0.2503	0.9498	0.6056
	$u = 0.65$	0.2431	0.9514	0.5471

Table 4.2
ARI of the resulting clustering in Scenario 7 with different values of u, different numbers of clusters, R=100 replications and different types of linkage.

4.1.3. Results using the LNS Algorithm with the Complete Linkage Model

In this section we have tested the LNS algorithm based on the complete linkage model formulation, this is because it is the model that has given us the best results in the previous sections, but the other formulations could be used as well. On this case it has been decided to use for the experiments always the value of $C = 3$, the number of true clusters in the data. In addition, several values of the threshold u are used, in order to obtain as much information as possible and to make a further

Linkage	Distance	$C = 2$	$C = 3$	$C = 4$
Single	GCC	0.0931	0.0052	0.0021
	CD	0.0650	0.0650	0.0650
Complete	GCC	0.6145	1.0000	0.7629
	CD	0.6145	1.0000	0.8545
Average	GCC	0.6145	0.8552	0.6120
	CD	0.6145	1.0000	0.6326

Table 4.3
ARI of the resulting clustering in Scenario 6 with $u = 0.6$, different numbers of clusters, different types of linkage, R=100 replications and using cophenetic distances or GCC.

Linkage	Distance	$C = 2$	$C = 3$	$C = 4$
Single	GCC	0.0901	0.0069	0.0324
	CD	0.0069	0.0069	0.0651
Complete	GCC	0.6145	0.4239	0.3936
	CD	0.6145	0.9866	0.7750
Average	GCC	0.6145	0.4239	0.6041
	CD	0.6145	0.8961	0.7575

Table 4.4
ARI of the resulting clustering in Scenario 4 with $u = 0.6$, different numbers of clusters, different types of linkage, R=100 replications and using cophenetic distances or GCC.

analysis of the impact of the value of u. The algorithm has a maximum number of iterations of 500.

We can observe in Table 4.6 that the ARI obtained for the models in which the cophenetic distance is used is equal to or better than the models in which the dissimilarity matrix is the GCC. But in both cases good cluster classifications are obtained.

However, the ARI's obtained are not better than those in Tables 4.5, 4.4, 4.3 and 4.2, but it is what we expected and in fact they are quite similar. Our goal in creating the metaheuristic algorithm was to reduce the computational time while maintaining good cluster classification, and we have successfully achieved this, as can be seen in the last column of the table (remember that the time needed to obtain good cluster classifications with the previous models in these same scenarios was 100 seconds). To get an idea, the average execution time (looking at the values of each execution of the algorithm in the tables) was 44.5, less than half that of the previous sections. Moreover, as we can see in the Table 4.2, the solution with which we obtain the best objective function is found before the 100th iteration. This is

Linkage	Distance	$C = 2$	$C = 3$	$C = 4$
Single	GCC	0.0069	0.0330	0.0480
	CD	-0.0792	-0.0331	0.1093
Complete	GCC	0.0120	0.0799	0.0204
	CD	0.6145	0.7920	0.6905
Average	GCC	0.0102	0.0063	0.2407
	CD	0.3647	0.7711	0.6041

Table 4.5
ARI of the resulting clustering in Scenario 2 with $u = 0.6$, different numbers of clusters, different types of linkage, R=100 replications and using cophenetic distances or GCC.

u	Distance	Scenario			
		S2	S4	S6	S7
0.6	GCC	0.1484 (55 s)	0.6099 (47 s)	1.0000 (42 s)	1.0000 (43 s)
	CD	1.0000 (51 s)	1.0000 (44 s)	1.0000 (43 s)	1.0000 (43 s)
0.7	GCC	0.0799 (55 s)	0.6099 (44 s)	0.8202 (40 s)	1.0000 (39 s)
	CD	1.0000 (51 s)	1.0000 (44 s)	1.0000 (43 s)	1.0000 (39 s)
0.8	GCC	0.0163 (55 s)	0.6099 (47 s)	1.0000 (38 s)	1.0000 (36 s)
	CD	1.0000 (52 s)	0.8202 (45 s)	1.0000 (36 s)	1.0000 (36 s)

Table 4.6
ARI and computational time (in seconds) of the resulting clustering in different scenarios with different values of u, using the LNS algorithm with the cophenetic distances or GCC and with a percentage of destruction of 0.3.

something that is repeated a lot in these scenarios, the best solutions are usually obtained before the first 100 iterations, so if we decrease this limit of iterations, we could again improve the computational time.

As a final remark, let us note that as the value of u increases, that is, as we force individuals to belong to a cluster more strongly, the results do not get worse, which tells us that the cluster structure in the data is very clear.

4.2. Montecarlo Experiments on Time Series Clustering: Factor Model Scenarios

We now turn to testing the algorithm with the factor models with cluster structure proposed by Ando and Bai, 2013. The particularities of these models are that the different clusters have a much stronger group structure because it is assumed that

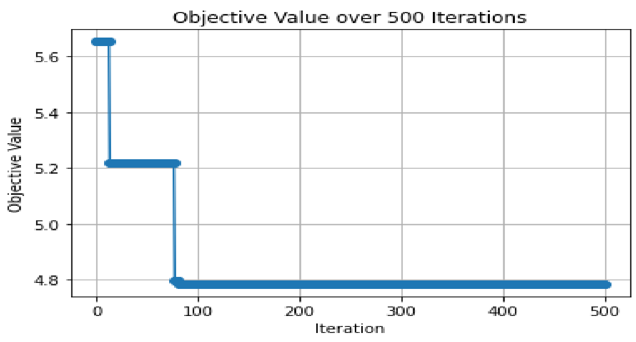


Figure 4.2
Improvement of the objective function over the 500 iterations of the LNS algorithm. Scenario 6, u=0.7, CD and percentage of destruction 0.3.

each cluster shares a common factor, and that the size of the groups is much larger, making the data created resemble real-life time series examples.

Let us define the three factor model scenarios or data-generating processes (DGP). For all scenarios, suppose we have C clusters in a set of N time series. The group membership is denoted by $G = \{g_1, g_2, ..., g_N\}$, and the number of elements in the group j is N_j, then three scenarios with different settings are defined:

- **Factor scenario 1**: In the first DGP, the model is

$$y_{it} = X_{it}\beta + F_{g_i,t}\lambda_{g_i} + \epsilon_{it} \qquad\qquad i = 1, .., N,\ t = 1, ...T,$$

  where $F_{g_i,t} = (f_{1t}, f_{2t}, ..., f_{Ct})$, and f_{jt} is the r_j-dimensional group-specific pervasive factor, a vector of elements that follows a Normal distribution with mean j and standard deviation 1. Each λ_j, element of the factor loading matrix Λ_{g_i} follows a normal distribution with zero mean and standard deviation g_i. The noise vector ϵ_i is multivariate normal with zero mean and covariance matrix I_N. The number of columns of X_i (observable data) is $p = 80$, with $q = 3$ true predictors, and each of its elements is uniformly distributed between $[-2, 2]$. The non-zero parameter values of β are set to $[1, 2, 3]$ in the first three elements, making

$$\beta = (1, 2, 3, 0, \ldots, 0).$$

  We set (for every scenario, in fact) $C = 3$ groups, with group-specific factors having $r_1 = 3$, $r_2 = 3$, and $r_3 = 3$, and each group will have $N_1 = N_2 = N_3 = 100$ units, so $N = 300$, what will lead to a 300×300 dissimilarity matrix. Moreover, the parameter T is fixed at 100.

- **Factor scenario 2**: The second DGP incorporates non-homoskedastic errors with cross-sectional dependence. The model has the same structure than before

$$y_{it} = X_{it}\beta + F_{g_i,t}\lambda_{g_i} + \epsilon_{it} \qquad\qquad i = 1, .., N,\ t = 1, ...T,$$

but now

$$\epsilon_{it} = 0.9e^1_{it} + \delta_t 0.9e^2_{it}.$$

Here, $\delta_t = 1$ if t is odd and 0 otherwise. The vectors e^1_t and e^2_t have length N and their elements follow a multivariate normal distribution with mean 0 and covariance matrix S, where $S_{ij} = 0.3^{|i-j|}$. There is no serial correlation and the noise terms are independent. The rest of the objects in the model (the group-specific factors and their loading matrices, the design matrix X_i, and the parameter vector β) are constructed in the same way as in the previous scenario.

- **Factor scenario 3**: The particularity of the third DGP is that allows the noise terms to have serial and cross-sectional correlations. The model is again

$$y_{it} = X_{it}\beta + F_{g_i,t}\lambda_{g_i} + \epsilon_{it} \qquad\qquad i = 1, .., N,\ t = 1, ...T,$$

where

$$\epsilon_{it} = 0.2\epsilon_{i,t-1} + e_{it},$$

and e_t follows a multivariate normal distribution with zero mean and covariance matrix S, with $S_{ij} = 0.3^{|i-j|}$. The other variables remain as described in Factor scenario 1.

These three DGPs have been implemented in Python and can be found in the script called $factorial\_data.py$ in the github repository. Before discussing the results obtained, it should be mentioned that the table of results using the model with the single linkage has been omitted because the results obtained were as bad as those in Table 4.1.

4.2.1. Results using the Complete and Average Linkage Models

The Tables 4.7, 4.8, 4.9 and 4.10 contain the ARIs obtained using the formulations (2.28) and (2.27) with the Gurobi solver. Some insights:

- Because the computational time using the complete linkage is much less than using the average linkage, the former has been used for obtaining Tables 4.7, 4.8 and 4.9, while the average linkage model has been only used in the Table 4.10 (scenario 1). Even so, the program takes an average of two hours to find a good solution for the complete models and three hours for the average ones. It should be remembered that the time limit established for the program executions has been 3 hours, and it is likely that increasing this time can improve the solutions in Table 4.10.
- As we can see in the Tables 4.7, 4.8 and 4.9, focusing on the column of $C = 3$, the ARI obtained for the cases where the cophenetic distance is used is almost

Threshold	Distance	$C = 2$	$C = 3$	$C = 4$
u=0.6	GCC	0.0021	0.5350	-0.0032
	CD	0.3514	0.9508	0.6638
u=0.7	GCC	0.0145	0.3739	0.0532
	CD	0.3173	0.9402	0.7012
u=0.8	GCC	0.0024	0.2734	-0.1238
	CD	0.3639	0.9601	0.6910

Table 4.7
ARI of the resulting clustering in factor Scenario 1 with different numbers of clusters, different values of u, R=5 replications, using cophenetic distances or GCC and the complete linkage model.

Threshold	Distance	$C = 2$	$C = 3$	$C = 4$
u=0.6	GCC	0.0024	0.3201	-0.0312
	CD	0.2542	0.8503	0.7331
u=0.7	GCC	0.1231	0.0104	0.0975
	CD	0.2783	0.9402	0.8638
u=0.8	GCC	-0.0081	0.0012	0.0194
	CD	0.3514	0.8014	0.7001

Table 4.8
ARI of the resulting clustering in factor Scenario 2 with different numbers of clusters, different values of u, R=5 replications, using cophenetic distances or GCC and the complete linkage model.

perfect, taking a value around 0.95 in most cases. On the other hand, using the GCC, the results are poor.

- Another interesting aspect, again focusing on the column of $C = 3$, is that as we increase the value of u, i.e. as we force individuals to belong to a cluster with a higher degree of membership, the results are just as good (using the cophenetic distance).

- Turning now to the cases of $C = 2$ and $C = 4$, as expected, the results obtained are not good. The ARI of all the cells with GCC are more or less zero, which means that the classification in clusters obtained is almost random. On the other hand, the ARI increases with the use of the cophenetic distance, not giving reasonable classifications, but not enough to be considered random. ARI values of around 0.33 are obtained in the case $C = 2$ and 0.66 in the case $C = 4$. Moreover, as the parameter $C = 4$ allows to obtain almost perfect cluster classifications, contrary to $C = 2$, it is worth mentioning that the ARI

Threshold	Distance	$C = 2$	$C = 3$	$C = 4$
u=0.6	GCC	0.0316	0.1357	0.0051
	CD	0.2107	0.9328	0.6041
u=0.7	GCC	0.0071	0.0043	-0.0132
	CD	0.3274	0.9511	0.6326
u=0.8	GCC	0.1003	0.0193	0.6610
	CD	0.3180	0.8945	0.7575

Table 4.9
ARI of the resulting clustering in factor Scenario 3 with different numbers of clusters, different values of u, R=5 replications, using cophenetic distances or GCC and the complete linkage model.

Threshold	Distance	$C = 2$	$C = 3$	$C = 4$
u=0.6	GCC	0.0184	-0.0068	0.0002
	CD	-0.0012	0.0003	0.0007
u=0.7	GCC	-0.0017	0.0127	-0.1004
	CD	0.1201	0.3642	0.0969
u=0.8	GCC	0.0001	-0.0308	-0.0146
	CD	-0.0004	0.5415	0.0168

Table 4.10
ARI of the resulting clustering in factor Scenario 1 with different numbers of clusters, different values of u, $R = 5$ replications, using cophenetic distances or GCC and the average linkage model.

obtained with $C = 4$ is always better than the one obtained with $C = 2$.

- Finally, for the realisation of the table using the model with the average linkage, due to the high computational cost, only Scenario 1 was used and, as we can see, the results are always slightly worse than those of the complete model. Again, the best ARI is obtained when the number of clusters is correct ($C = 3$) and when the cophenetic distance is used, but in this case the results obtained are much worse because with the average linkage it takes much longer to find good solutions.

- Comparing our results with the tables of the Alonso et al., 2021 factor models (only the $C = 3$ column), we conclude that using the complete model together with the cophenetic distance the ARI values obtained are very similar (almost perfect), while our ARI values obtained with the complete model and the GCC are significantly worse in all scenarios. Using the average model we obtain worse results in all scenarios using both the cophenetic distance and the GCC.

- To summarise this section of the factor models, we can conclude that better cluster classifications are always obtained with the cophenetic distance than with the GCC, which are almost perfect. Moreover, as happened in the section on cross-dependence models, if we change the number of clusters to one that is not the true one, the model with the GCC fails to detect the structure of the clusters, obtaining random classifications, while the models with the cophenetic distance detect this structure and provide reasonable classifications. Finally, it is concluded that the use of the model with the average linkage is best used on smaller data, due to its high computational cost and its worse performance compared to the model using the complete linkage.

4.2.2. Results using the LNS Algorithm with the Complete Linkage Model

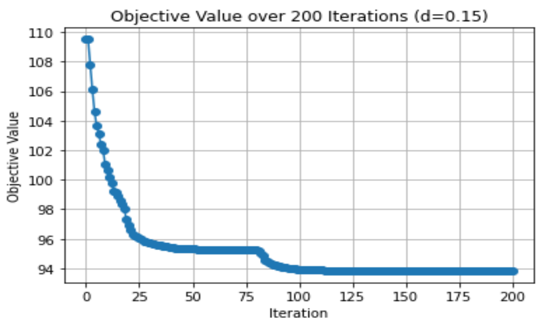
Table 4.11 shows the ARI and the computational time of the executions of the LNS algorithm with the complete linkage model (again, only the formulation of the complete model has been used as it gives the best results when introduced in the Gurobi solver) for different values of u and in the three different factor scenarios and again, the results obtained using the cophenetic distance are quite satisfactory, the classifications are close to being similar than in the previous section, but we do notice a notable improvement in the computational time. We went from taking 10800 seconds (3 hours) to obtain good solutions, to taking on average 690 seconds (0.2 hours). On the other hand, as has become common, the ARIs using GCC are not good.

Another interesting aspect that we can see in the table, and which differentiates it from the others, is that in the CD cases, as the value of u increases, i.e. as we force individuals to belong to a cluster with a higher value of degree of membership, the cluster classifications are better. This may be because the scenarios created in this section have very strong group structure, so that individuals belong to a certain cluster with a large membership degree value, so by increasing the threshold u, what we do is to limit the search by orienting it to find such a strong cluster distribution in an easier way.

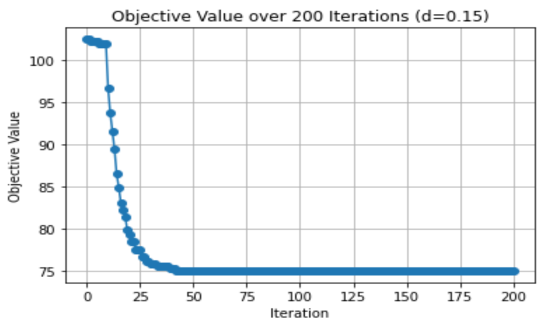
This time, because in the previous LNS experiments we realised that most of the improvement of the objective function takes place in the first iterations, we have limited the number of maximum iterations to 200. The evolution of the improvement of the objective function for the cophenetic distance executions with $u = 0.7$ for scenarios 1, 2 and 3 can be seen in Figure 4.3.

u	Distance	Scenario		
		FS1	FS2	FS3
0.6	GCC	-0.0021 (598 s)	0.0713 (583 s)	0.0009 (506 s)
	CD	0.4905 (572 s)	0.4129 (3230 s)	0.4281 (1827 s)
0.7	GCC	0.1297 (634 s)	0.0012 (534 s)	0.1086 (592 s)
	CD	0.6600 (1033 s)	0.8867 (525 s)	0.8561 (825 s)
0.8	GCC	0.0305 (544 s)	0.0002 (518 s)	0.0001 (486 s)
	CD	0.9899 (670 s)	0.9509 (641 s)	0.9320 (545 s)

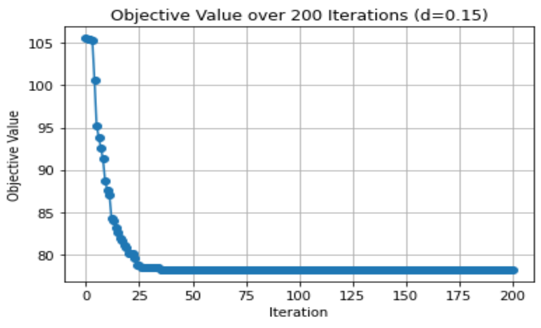
Table 4.11
ARI and computational time (in seconds) of the resulting clustering in different scenarios with $u = 0.6$, $u = 0.7$, and $u = 0.8$, using the LNS algorithm with the complete linkage, cophenetic distances or GCC and with a percentage of destruction of 0.15. The algorithm has a maximum number of iterations of 200.



Scenario 1



Scenario 2



Scenario 3

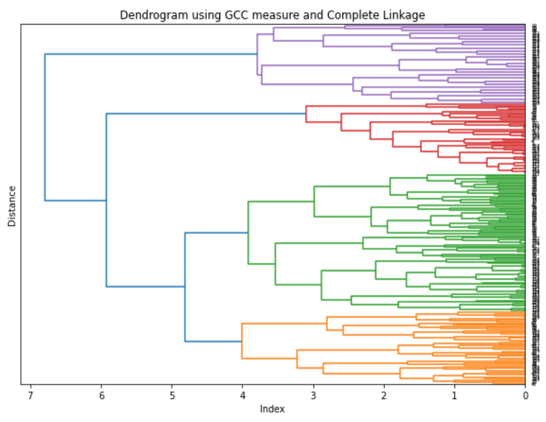
Figure 4.3
Improvement of the objective function over 200 iterations for 3 scenarios using the cophenetic distances model and u=0.7.

5. REAL DATA CASE

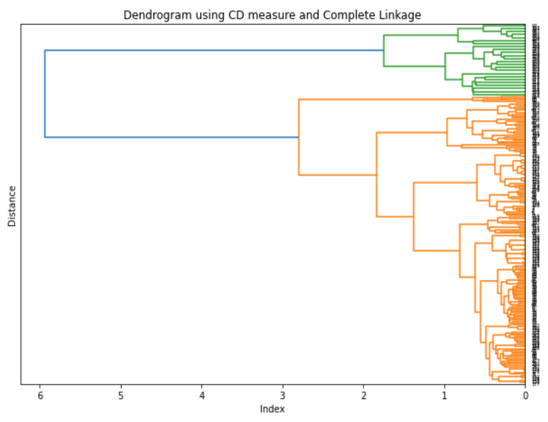
The aim of this chapter is to test our new approach for fuzzy clustering on a real data set. To do this, we will introduce to Gurobi the model created with the complete linkage (2.27) using both the cophenetic distance and the GCC measure.

The dataset chosen, in order to know if our algorithm performs good cluster classifications and if it chooses a reasonable number of clusters, is the same as the one used in Alonso et al., 2021. The data can be found in www.mercatoelettrico.org or in the repository https://github.com/AleeexMR/TFM.

Italy is divided into ten load areas: Centro-Nord (CNOR), Centro-Sud (CSUD), Nord (NORD), Sardegna (SARD), Sicilia (SICI), Sud (SUD), Brindisi (BRNN), Foggia (FOGN), Priolo Gargallo (PRGP), and Rossano (ROSN). Some correspond to geographical areas and others to areas where production is restricted. The latter are BRNN, PRGP, FOGN and ROSN. The database consists of 240 time series, each time series corresponding to electricity prices in one of the ten regions for one of the 24 hours of the day. The data were collected from 1 January 2014 to 31 December 2018. Each series has data for 365 (or 366) days of the five years, so there are 1826 data points.



With GCC measure



With CD measure

Figure 5.1
Dendrograms obtained with the GCC and CD measures, respectively, and the complete linkage for the Italian load areas

The dendrograms in Figure 5.1 have been created using hierarchical clustering with the GCC and the cophenetic distances, respectively. The dendogram with GCC suggests that there are four clusters while the other suggests two clusters, as in Alonso et al., 2021. Furthermore, the clustering of the hierarchical procedure using the cophenetic distance is the same as in the previously mentioned article. One cluster is formed by the SICI and PRGP series and the other by the rest of the series.

Table 5.1 shows the silhouette scores when using our fuzzy algorithm on the data. On the one hand, using the model with the GCC measure, the silhouette scores are generally low. The highest value corresponds to C=2 clusters and is 0.2221, followed by C=4 (0.1693). From C=4 onwards, the silhouette scores decreases and becomes negative, indicating a worse clustering. The worst values are obtained with $k = 6, 7, 8, 9$ and 10, indicating that increasing k beyond 5 significantly worsens the quality of the clustering.

On the other hand, with the cophenetic distance, silhouette scores are generally higher up to k=5 compared to those obtained in the GCC. Again, as the number of clusters increases, the quality of the clusters is worse. And finally we can clearly conclude that the optimal number of clusters is 2 (silhouette score of 0.5413), which again correspond to the SICI and PRGP series on the one hand, and the rest of the series on the other hand (same result as in Alonso et al., 2021).

u	Measure	Number of clusters								
		C=2	C=3	C=4	C=5	C=6	C=7	C=8	C=9	C=10
0.7	GCC	0.2221	0.1187	0.1693	0.04403	0.0094	0.0038	-0.0057	-0.0330	-0.0230
	CD	0.5413	0.2322	0.1436	0.1329	-0.0137	-0.0425	0.0527	-0.0734	-0.0852

Table 5.1
Silhouette scores for different number of clusters

Figure 5.2 shows the silhouette plot for the algorithm execution using the cophenetic distance and two clusters. A silhouette coefficient close to 1 indicates that the points are well grouped in their own cluster and away from other clusters, coefficients close to 0 mean that the points are on the boundary between two clusters and if they are negative, the points are misclassified. The dashed red line indicates the average value of the silhouette coefficient, which is a good value, between 0.5 and 0.6. It indicates that the clusters are reasonably well separated and are internally cohesive.

Figures 5.3 and 5.4 show the series of prices corresponding to hour 10 of, the eight regions corresponding to a cluster, and those of SICI and PRGP. We can observe that there is a common pattern in the series of the first cluster, while those of the second cluster are identical for that hour, and with each other (series of cluster 1 vs series of cluster 2), they are quite different.

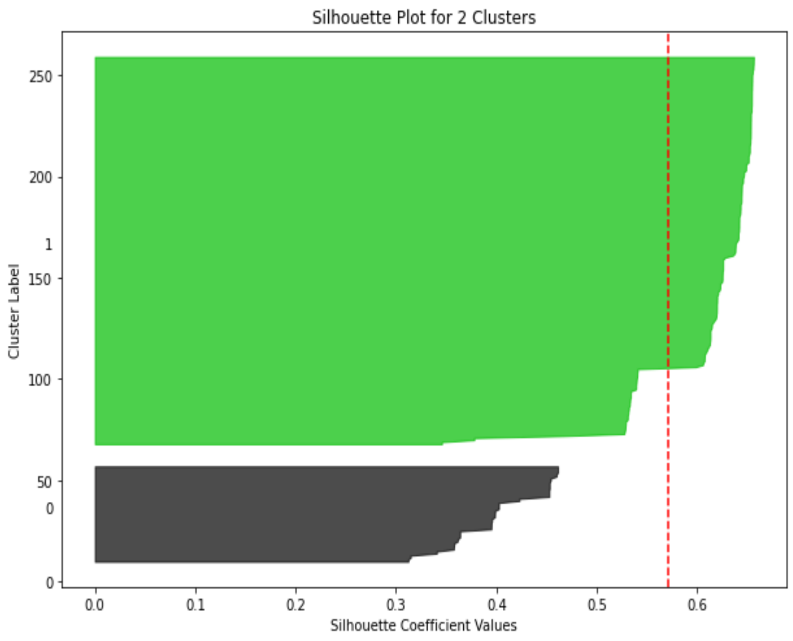


Figure 5.2
Silhouette plot for k=2 using the cophenetic distance showing the distribution of the silhouette coefficients for each of the two clusters.

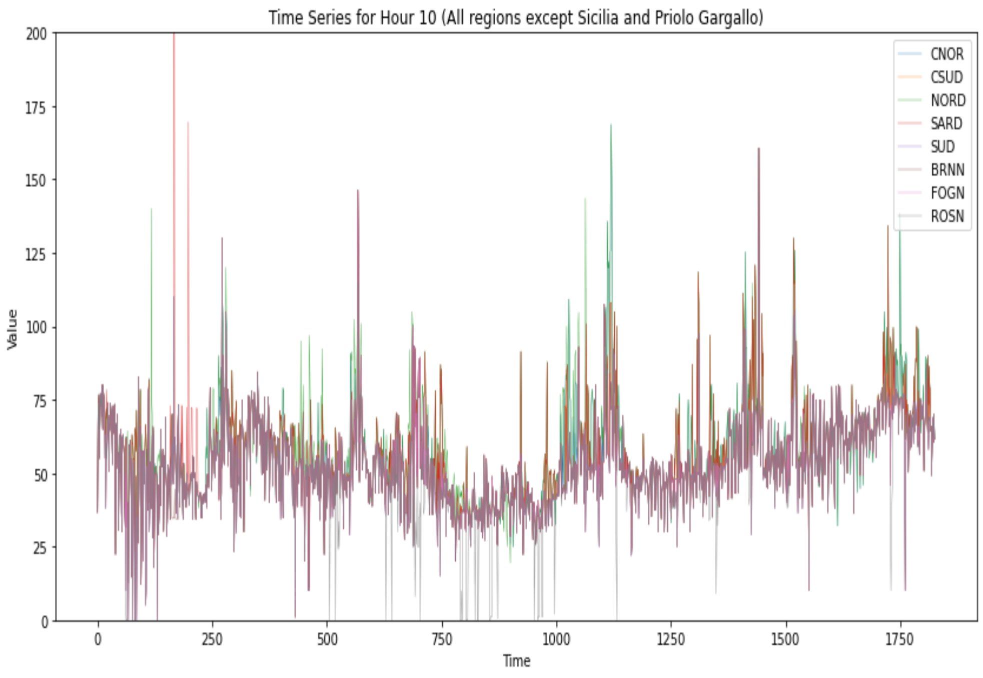


Figure 5.3
Electricity price evolution in hour ten of the different areas of cluster 1: CNOR, CSUD, NORD, SARD, SUD, BRNN, FOGN and ROSN.

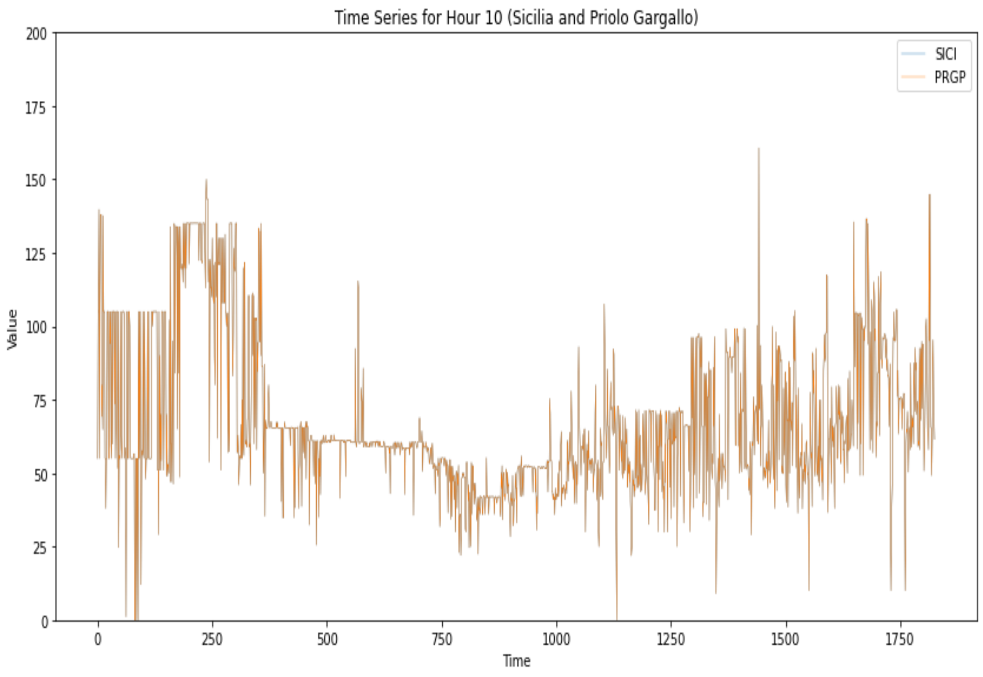


Figure 5.4
Electricity price evolution in hour 10 of the different areas of cluster 2: SICI and PRGP.

6. CONCLUSIONS AND FUTURE WORK

During the course of this project, new fuzzy clustering models have been developed that take into account simple, average or complete linkages as a way of measuring the distance between individuals. After modeling these problems as mixed integer non-linear optimization problems, we identified that single linkage was not capable of performing good cluster classifications.

One of the major limitations of the proposed formulations is the computational time needed by off-the-shelf solvers to find good-quality solutions, therefore, a new metaheuristic algorithm, the Large Neighborhood Search, has been proposed to solve these formulations in a heuristic way with the aim of reducing the time in which classifications are obtained and without damaging the quality of the clustering.

The experimentation/simulation study has been divided into two stages. In the first, the three models created (single, complete and average linkages) were tested in different small scenarios characterised by their cross-dependencies obtaining results either by entering the formulations directly into a solver or with the metaheuristic algorithm. In the second, factorial scenarios were created, characterised by their larger size and strong group structure. Roughly speaking, the conclusions of these simulations show that the model created with the complete linkage together with the use of the cophenetic distance is the one that provides the best cluster classifications, although the model with the average linkage solved by an off-the-shelf optimization solver and the LNS based on the complete linkage also performs reasonably well. Furthermore, this simulation study shows the excellent performance of our new fuzzy clustering method, leaving aside the model of the single linkage.

Finally, seeing that the best model obtained was the one using the complete linkage, we tested it with real data and shown that our fuzzy approach enables us to clearly determine the number of groups. This number is inconclusive with classical clustering methods (Alonso et al., 2021).

As a possible future work, one could try to improve the metaheuristic approach by introducing an algorithm called Adaptive Large Neighbourhood Search (Ropke and Pisinger, 2006). With this strategy, different destruction and repair techniques might be used in the same search dynamically adjusting the frequency of use of each method based on its performance during the execution of the algorithm.

BIBLIOGRAPHY

Ahuja, R., Ergun, Ö., Orlin, J., & Punnen, A. (2002). A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics, 123, 75–102. https://doi.org/10.1016/S0166-218X(01)00338-9

Alonso, A. M., D'Urso, P., Gamboa Pinilla, D. C., & Guerrero, V. (2021). Cophenetic-based fuzzy clustering of time series by linear dependency. International Journal of Approximate Reasoning. 137, 114–136. https://doi.org/10.1016/j.ijar.2021.07.006

Alonso, A. M., & Peña, D. (2019). Clustering time series by linear dependency. Statistics and Computing, 29, 655–676. https://doi.org/10.1007/s11222-018-9830-6

Ando, T., & Bai, J. (2013). Panel data models with grouped factor structure under unknown group membership. Social Science Research Network, 31. https://doi.org/10.2139/ssrn.2373629

Bent, R., & Van Hentenryck, P. (2006, January). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers & Operations Research, 33, 875-893. https://doi.org/https://doi.org/10.1007/978-3-540-45193-8_9

Csardi, G., & Nepusz, T. (2006). The igraph software package for complex network research. InterJournal, Complex Systems, 1695. https://igraph.org

Dias, M. L. D. (2019). Fuzzy-c-means: An implementation of fuzzy C-means clustering algorithm. https://doi.org/10.5281/zenodo.3066222

Dunn, J. C. (1973). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. Journal of Cybernetics, 3, 32–57. https://doi.org/10.1080/01969727308546046

Gurobi Optimization, LLC. (2023). Gurobi Optimizer Reference Manual. https://www.gurobi.com

Hennig, C. (2010). Fpc: Flexible procedures for clustering. R Package Version, 2, 0-3. https://doi.org/10.32614/CRAN.package.fpc

Hubert, A. (1985). Comparing partitions. Journal of Classification, 2, 193–218. https://doi.org/https://doi.org/10.1007/BF01908075

Lloyd, S. P. (1982). Least squares quantization in PCM. IEEE Trans. Inf. Theory, 28, 129–136. http://dblp.uni-trier.de/db/journals/tit/tit28.html#Lloyd82

Nielsen, F. (2016). Hierarchical clustering. Introduction to HPC with MPI for Data Science. Springer International Publishing. https://doi.org/10.1007/978-3-319-21903-5_8

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikitlearn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830. https://doi.org/https://doi.org/10.48550/arXiv.1201.0490

Pisinger, D., & Ropke, S. (2010). Large neighborhood search. Handbook of Metaheuristics, 399-419. https://doi.org/10.1007/978-1-4419-1665-5_13

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science, 40, 455–472. https://doi.org/10.1287/trsc.1050.0135

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. Lecture Notes in Computer Science, 1520, 417 - 431. Springer. https://doi.org/doi.org/10.1007/3-540-49481-2_30