Cluster Detection by Lifting with Application to Phylogenetics

Nebahat Bozkus

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy

The University of Leeds

Department of Statistics

May 2018

I would like to dedicate this thesis to my mother Suzan Bozkuş and my father Yaşar Bozkuş for their endless support in my life.

Acknowledgements

I would like to firstly thank my supervisor Dr. Stuart Barber. For four years, his guidance and endless support in all the stages of my study have been priceless. I am also thankful to the Ministry of National Education Turkey for funding my PhD study.

I am grateful to my mother Suzan Bozkuş, my father Yaşar Bozkuş and my siblings Gökhan, Merve and Sefa for their endless love, support and encouragement during all my life. I am specially thankful to my father for encouraging me to follow my dreams even in his last minutes in this life. I would like to also thank Dr. Neveen Al Saeed for being more than a friend in Leeds for me. Without having our breaks from the study, it would not be easy to complete my PhD study. Finally, I would like to thank all my friends and colleagues for their support and understanding.

Abstract

In this thesis, we propose a new algorithm which automatically detects the number of clusters in a tree structure data set by denoising some generalized node values in the tree using lifting “one coefficient at a time” (LOCAAT) algorithm introduced by Jansen et al. (2001). Our algorithm can be applied to any multidimensional data set using compactness value as a node value or to phylogenetic data sets, DNA sequences, using either compactness value or dissimilarity score as a node value. Compactness value is defined as the average distance from the centroid of each possible cluster in the tree, and the dissimilarity score is the average number of loci, where at least one of them does not share the same nucleotide between sequences under the node of interest.

For multidimensional data sets, we consider each node in the tree as a possible location of a cluster after denoising the tree by LOCAAT. Thus, for each possible cluster, we check how much departure we can allow from the centroid of the cluster to assign the objects under the node of interest as a cluster. Then if a node and all its child nodes are denoised less than or equal to the allowed amount of departure from the centroid of their clusters, a cluster is located at this node. We also propose another version of our algorithm based on non-decimated lifting (Knight & Nason, 2009) in which we generate a probability of being clustered for each node. If a node and all its child nodes have a probability of being clustered less than or equal to the probability of acceptance, $\theta \in [0, 1]$, a cluster is located at this node. We provide a comparison study between our algorithms and some available internal cluster validity indices (CVIs) in the literature using some artificial data sets and a real data set. In addition, we compare the performance of each method using some available external cluster validity scores.

For phylogenetic data sets, we check the performance of our algorithms and other CVIs using both compactness value and dissimilarity score as a node value. To be able to compute compactness value for a phylogenetic tree, we need to find the position of each specie in $\mathbb{R}^p$ using multidimensional scaling (MDS), and then we can find which species share the similar features using our algorithm. If we use the dissimilarity score as a node value, we will cluster similar species together by finding how much difference we can allow between species. We check the performance of our algorithms using some artificial and a real data sets.

In the final part of our thesis, we propose a visualization tool for cophylogenetic data sets. We only consider the associated two phylogenetic trees case, and we apply our algorithm to both host and parasite trees separately to provide a summary of these data sets. We check the performance of our algorithm using two well-known cophylogenetic data sets.

Contents

List of figures

List of tables

Chapter 1

Introduction

Clustering is a popular area in statistics. It helps to summarize a data set by finding related groups of objects, so one of the common questions in the clustering literature is how many clusters are in a data set. There are many available methods which attempt to find the most suitable partitioning structure in terms of their own rules; however, none of them captures the true components of all types of data structure with a high performance. They generally struggle to partition a data set if it is uniquely shaped, or if different objects from different groups are closely located. In addition, when we build a dendrogram of a multidimensional data set, the available methods find the number of clusters, so we can cut the dendrogram from any place which gives the number of clusters obtained by the method of use. In this thesis, we discuss a new method which finds one of the best representative clustering pattern and which finds the exact place of each cluster in a dendrogram.

Our proposed method is based on hierarchically built trees. It partitions the data set using a denoising method. We know that all data sets are noise corrupted, and when we hierarchically find the tree representation of a data set, each agglomeration step also adds some extra noise to the tree. In hierarchical clustering, the tree is built by iteratively merging the closest pair of objects at a time, and a new object to represent these merged pairs is created. Thus, the distance between the new object and others is based on the estimated distance from the objects at the previous stage which adds the extra noise to the tree. We assume that we can find a better representative partitioning to any data set by denoising some representative function values on all nodes in the tree.

In this study, we explore the behaviour of a recently developed wavelet-like method called lifting (Sweldens, 1998) which is also known as second generation wavelets. It is based on the wavelet transform, but it has fewer restrictions than the wavelet transform which we discuss in detail in Chapter 3. It is applicable for more general data sets which have a neighbourhood structure, and it estimates the function value of a particular point in the space using the function values of the neighbours. Then the estimated values are easily denoised using similar methods to those used in wavelet shrinkage, since the lifting transform finds detail coefficients analagous to those obtained by the wavelet transform. However, the lifting transform introduced by Sweldens (1998) can not deal with multi-dimensional data sets. Jansen et al. (2001) later introduced the lifting "one coefficient at a time" (LOCAAT) algorithm which can easily deal with many data structures including multidimensional data points and networks. Our aim in this study is to denoise hierarchically built trees to find the clustering structures, so we use the LOCAAT algorithm on networks (Jansen et al., 2009).

Finding related objects do not only attract the attention of statisticians, but it is also a popular area in evolutionary studies. The relationship between related species is explained by phylogenetic trees, which are binary trees. Thus, these trees take a form equivalent to the trees we are interested in. We propose that we can easily apply our lifting-based clustering algorithm to phylogenetic data sets which are the DNA sequences of related species. The only difference in these data sets is the process of finding the distances between sequences and the representative function values for each node on the tree.

Overall, we propose a new method which finds the exact place of a cluster in any binary tree. We separately discuss how to apply our algorithm on any multidimensional data sets and on phylogenetic data sets. We give a chapter by chapter review of this thesis in the following section.

1.1 Thesis overview

In this thesis, we start with a brief introduction to the wavelet transform and wavelet shrinkage methods in Chapter 2, then we discuss different versions of the lifting transform in Chapter 3. After that we describe phylogenetic reconstruction methods in Chapter 4. Then we propose our lifting-based clustering algorithm both for multidimensional and phylogenetic data sets in Chapters 5 , 6 and 7.

In Chapter 2, we start our discussion of the wavelet transform by describing multiscale analysis using Haar wavelets. Then, we detail Haar wavelets and multiresolution analysis. Next to Haar wavelets, we also add a brief discussion of one of the well known wavelets family, Daubechies' compactly supported wavelets. Then we discuss the discrete wavelet transform in detail. We also summarize another wavelet transform, the non-decimated wavelet transform. Next, we describe different wavelet shrinkage methods, and we finish our discussion of wavelets by a small simulation study which compares the different wavelet transforms with different wavelet shrinkage methods. Even though we do not use wavelet transform directly in this thesis, giving a brief discussion on this topic will help us easily understand the lifting transform which is the base of our clustering methods introduced in Chapters 5 , 6 and 7.

In Chapter 3, we discuss the LOCAAT algorithm in detail. We start our discussion by reviewing how to find the lifting transform of one dimensional data sets by LOCAAT, and we describe its working structure on a toy data set. Then we detail two different lifting transformation methods which are based on LOCAAT: adaptive lifting (Nunes et al., 2006) and non-decimated lifting (Knight & Nason, 2009). We compare the behaviour of different lifting methods and different wavelet transformations using a simulation study based on some popular wavelet test functions introduced by Donoho & Johnstone (1994) (Blocks, Bumps, Doppler and Heavisine functions) and by Nason & Silverman (1994) (piecewise polynomial function). Then we describe LOCAAT on multidimensional data sets, and we complete the chapter with a tree-structured toy example, where we go through each step of this transformation in detail.

In Chapter 4, we describe the phylogenetic reconstruction process by detailing two different methods: parsimony and distance methods. Then we discuss some of basic evolutionary models which help us to understand final phylogenetic reconstruction methods: probabilistic methods. Reviewing different phylogenetic construction methods helps us understand the structure of phylogenetic trees, and directs us to apply our algorithm to phylogenetic trees in Chapter 7.

After detailing some methods which our clustering method is based on, we can start introducing our method in Chapter 5. We start this chapter with a brief discussion of hierarchical clustering, and we introduce some of the available internal cluster validity indices (CVIs) and some of the similarity scores (external cluster validity scores) which help us to check the performance of CVIs when we know the true partitioning of a data set. We also describe another clustering method: mixture model-based clustering (Mclust; Fraley & Raftery 2002). Even though we aim to compare the partitioning found by different methods using hierarchically built trees, it will be also interesting to explore the behaviour of partitioning found by Mclust. Then we introduce how we find the clusters in a multidimensional data set by denoising. We start our algorithm by finding the pairwise Euclidean distances between different objects, then we build the dendrogram of the data set using hierarchical clustering. Building the tree brings us joined pairs in each agglomeration step with the edge lengths, and we need a function value for each node on the tree along with the information we obtain from hierarchical clustering. We propose that we can denoise the compactness value which is defined as the average distance from the centroid of each cluster. If the denoised detail coefficients of a node and all its child nodes are less than or equal to a small threshold, we can place a cluster at this node. In this way, we introduce a new clustering method which finds the exact location of each cluster in a tree. After introducing our clustering algorithm, we provide a simulation study, where we compare the performance of our proposed method, Mclust and some of the available CVIs in the literature in terms of various similarity scores. We generate four different data structures, where two of them include components coming from normal distributions, and other two data structures are more complex including uniquely shaped components. We also check the performance of different methods on a real data set.

Our proposed method in Chapter 5 is semi-automatic because it includes an arbitrary choice of a threshold to decide where clusters are located in a tree. However, we wish to present an algorithm which automatically finds the location of each cluster by omitting the arbitrary threshold choice step. In Chapter 6, we propose an updated version of our algorithm which picks its own threshold. If the denoised compactness value of any node and all its child nodes are less than or equal to the universal threshold, described in Section 2.10.2, we can place a cluster at this node. In addition, we introduce another version of our clustering algorithm based on the non-decimated lifting transform instead of LOCAAT. This version of our algorithm provides a probability of being clustered at each node on the tree. Then we complete the chapter by repeating the same comparative study as the previous chapter by adding the results of our updated algorithm based on LOCAAT and non-decimated lifting.

After completing the application of our algorithm to multidimensional data sets, we discuss how we can find clusters in phylogenetic trees in Chapter 7. We start by finding the pairwise distances between DNA sequences by counting the loci which do not share the same nucleotide (matching distance), or we can find phylogenetic distances using one of the evolutionary models described in Chapter 4. After finding distances between sequences, we can build a tree using hierarchical clustering, and we can use the compactness value as a node value. However, to be able to find compactness value, a multidimensional representation of each sequence must be found using multidimensional scaling. For phylogenetic data sets, we also propose another node value, a dissimilarity score. For each node, we compare all the species under the node of interest locus by locus, and compute the average number of loci for which at least one species does not have the same nucleotide as all the others. Our algorithm for phylogenetic data sets is applied to three artificial data sets, where two of them include only mutation history (with no sub-populations), and one of them includes both mutation and migration history (with three sub-populations). We compare the performance of our proposed methods (based on LOCAAT with manual and automatic threshold choices, and based on non-decimated lifting transform), Mclust and other CVIs. In addition, we check the behaviour of different distance methods and different function values. We also provide a comperative analysis of a real data set. Finally, we check the behaviour of our clustering algorithm on cophylogenetic data sets. Our aim is to propose a visualization tool, so we do not propose a test which looks for the different cophylogenetic events. We check the behaviour of our algorithm using two real cophylogenetic data sets.

To conclude, we summarize our methods and findings in Chapter 8, and discuss some potential further work.

Chapter 2

Wavelets

In Chapter 5, we propose a method based on lifting "one coefficient at a time" (LOCAAT) algorithm (Jansen et al., 2001) to find exactly where clusterings happen in a dendrogram. Lifting is a denoising method which has the wavelet theory behind it, so we briefly discuss wavelet methods in statistics and some of the wavelet shrinkage methods in this chapter before going through the details of lifting algorithm in Chapter 3. For further reading, early studies from Daubechies (1992), Chui (1997), Mallat (1998) and Vidakovic (1999) can be used. In addition, Nason (2008) also provides a summary of wavelets with its application in **R** software. While the early studies provide the detailed theory of wavelets, Nason (2008) provides more application and comments on results.

2.1 Introduction

Suppose that we have a function corrupted by noise, $f(x)$, and we would like to estimate the function $g(x)$ by denoising the function $f(x)$. One of the methods to find the denoised function, $g(x)$, is wavelet shrinkage, introduced by Donoho & Johnstone (1994) and Donoho et al. (1995). The general form of the model which is used in the wavelet-based function estimation is

$$f(x_i) = g(x_i) + \varepsilon_i, \tag{2.1}$$

where $x_i = i/n, \varepsilon_i \sim N(0, \sigma^2)$, and $i = 1, \ldots, n$. The general idea in wavelet shrinkage is to first get the wavelet transform of the observed data, $f(x_i)$, then wavelet coefficients are denoised using one of the wavelet shrinkage methods. Finally, the inverse transformation of the denoised wavelet coefficients are applied to find the estimate of $g(x)$.

In this chapter, details of the wavelet transform and different wavelet shrinkage methods are described. Before discussing the discrete wavelet transform in detail, we look at multiscale analysis in Section 2.2, Haar wavelets in Section 2.3, multiresolution analysis in Section 2.4, vanishing moments in Section 2.5, and another wavelet family, Daubechies' compactly supported wavelets in Section 2.6. After that we discuss the discrete wavelet transform (DWT) and boundary conditions in Sections 2.7 and 2.8, respectively. In addition, we provide another transformation called the non-decimated wavelet transform (NDWT) in Section 2.9. Then we discuss wavelet shrinkage methods in Section 2.10. We start this section with a discussion of the ideal risk in the wavelet domain in Section 2.10.1, and we continue with the universal thresholding in Section 2.10.2, Bayesian wavelet shrinkage in Section 2.10.3 and the application of wavelet shrinkage methods on wavelet coefficients obtained by the NDWT in Section 2.10.4. We finalize the chapter with a simulation study where we compare the performance of the DWT and the NDWT in terms of different shrinkage methods.

2.2 Multiscale analysis

In this section, we discuss multiscale analysis and the DWT using Haar wavelets described in Section 2.3, and we deal with wavelet analysis of series, but we also consider the wavelet analysis of functions in Section 2.3. Let us start by defining the discrete vector of data

$$y = (y_1, y_2, \ldots, y_n)^T, \quad i = 1, 2, \ldots, n,$$

where $y_i \in \mathbb{R}$, and n is a dyadic integer, 2^J ($J \geq 0$).

Multiscale information is obtained using the vector y, so detail coefficients for different locations d_k are

$$d_k = y_{2k} - y_{2k-1}, \tag{2.2}$$

where $k = 1, 2, \ldots, n/2$. We would also like to know detail coefficients for different scales, where the scale parameter is $j = J - 1, \ldots, 0$, and $j = J - 1$ represents the finest level. We can rewrite Equation (2.2) as $d_{J-1,k} = y_{2k} - y_{2k-1}$, giving detail coefficients for the finest level. Thus, decreasing j means j goes from the finest level to coarser levels. To find detail coefficients for coarser levels, we need scaled local averages at the finest level, $c_{j,k}$, which are defined as

$$c_{j,k} = y_{2k} + y_{2k-1}, \tag{2.3}$$

where $k = 1, 2, \ldots, n/2$. We can also rewrite Equation (2.3) as $c_{J-1,k} = y_{2k} + y_{2k-1}$, giving scaling coefficients for the finest level. Thus, detail coefficients for the remaining levels are

$$d_{j,\ell} = c_{j+1,2\ell} - c_{j+1,2\ell-1}, \tag{2.4}$$

where $j = J - 2, J - 3, \ldots, 0$, and $\ell = 1, \ldots, n/2^{(J-j)}$. Detail coefficients $d_{J-1,k}$ and $d_{J-2,k}$ represent the 'scale one' and 'scale two' differences, respectively, and d_{jk} represents the detail coefficient at kth location in jth level. Scaling coefficients for coarser level j are

$$c_{j,\ell} = c_{j+1,2\ell} + c_{j+1,2\ell-1}, \tag{2.5}$$

where $j = J-2, J-3, \ldots, 0$, and $\ell = 1, \ldots, n/2^{(J-j)}$.

Example 2.2.1. *Assume that we have a sequence $y = (2,3,8,10,8,2,8,2)^T$, where T represents the transpose of a vector, and the length of the sequence is 8 ($n = 8$). We can find the wavelet and scaling coefficients for the finest level using*

$$d_{J-1,k} = y_{2k} - y_{2k-1},$$
$$c_{J-1,k} = y_{2k} + y_{2k-1},$$

where $n = 2^J$ and $k = 1, \ldots, n/2$, so $J = 3$ and $k = 1, \ldots, 4$. Hence, detail and scaling coefficients for the finest level ($J-1 = 2$) are

$$d_{2,k} = (1,2,-6,-6)^T,$$
$$c_{2,k} = (5,18,10,10)^T.$$

We can also find detail and scaling coefficients for the following coarser level using

$$d_{J-2,\ell} = c_{J-1,2\ell} - c_{J-1,2\ell-1},$$
$$c_{J-2,\ell} = c_{J-1,2\ell} + c_{J-1,2\ell-1},$$

where $\ell = 1, \ldots, n/4$, so $\ell = 1, 2$. Hence, detail and scaling coefficients for the following coarser level ($J-2 = 1$) are

$$d_{1,\ell} = (13,0)^T,$$
$$c_{1,\ell} = (23,20)^T.$$

Using Equations (2.4) *and* (2.5)*, detail and scaling coefficients for the coarsest level are found as*

$$d_0 = -3, \quad c_0 = 43.$$

After finding detail and scaling coefficients, we can represent output coefficients as a vector of detail coefficients and the scaling coefficients at the coarsest level, so

$$d = (d_{J-1,1}, d_{J-1,2}, \ldots, d_{J-1,k}, \ldots, d_{1,1}, d_{1,2}, d_0, c_0)^T.$$

Thus, we can write detail coefficients in Example 2.2.1 as $d = (1,2,-6,-6,13,0,-3,43)^T$.

The algorithm which we described until now is called *general pyramid algorithm*, and it is a kind of DWT algorithm, where coefficients $d_{j,k}$ and $c_{j,k}$ are called wavelet coefficients and father wavelet coefficients, respectively. Before the detailed explanation of DWT, we first look at what the inverse, sparsity and energy are for wavelets briefly.

Inverse. As we see in Example 2.2.1, the original series is rebuilt using detail coefficients $d_{j,k}$ and c_0. We can invert the transform using the combination of Equations (2.4) and (2.5), so the inversion can be done via Equations (2.6) and (2.7):

$$c_{j-1,2k} = (c_{j-2,k} + d_{j-2,k})/2, \tag{2.6}$$

and

$$c_{j-1,2k-1} = (c_{j-2,k} - d_{j-2,k})/2. \tag{2.7}$$

Sparsity. When many of the elements are zero in a sequence, this sequence is called a sparse set.

Energy. We would like the energy of the input ($\mathbb{L}_2$ norm of the vector y) to be equal to the energy of the output in each resolution level ($\mathbb{L}_2$ norm of the wavelet coefficients). For illustration, using the data from Example 2.2.1, $||y||^2 = \sum_{i=1}^{n=8} y_i^2 = 313$, and $||d||^2 = \sum_{i=1}^{n=8} d_i^2 = 2104$. While the energy of the input and the output sequences need to be equal, the energy of the output (wavelet coefficients) is nearly seven times larger than the norm of the input. Thus, we need a transformation to make the energy of the both input and output sequences equal.

2.2.1 Discrete Haar wavelets

To solve the energy problem, we can adapt Equations (2.2) and (2.3) to include a multiplier, α. Hence, Equation (2.2) becomes

$$d_k = \alpha(y_{2k} - y_{2k-1}),$$

and Equation (2.3) becomes

$$c_k = \alpha(y_{2k} + y_{2k-1}),$$

where $k = 1, 2, \ldots, n/2$. We try to make the energy of the output (c_k and d_k) and the input (y_{2k} and y_{2k-1}) equal. Thus, the norm of the output is

$$\begin{aligned} d_k^2 + c_k^2 &= \alpha^2(y_{2k}^2 - 2y_{2k}y_{2k-1} + y_{2k-1}^2) + \alpha^2(y_{2k}^2 + 2y_{2k}y_{2k-1} + y_{2k-1}^2), \\ &= 2\alpha^2(y_{2k}^2 + y_{2k-1}^2). \end{aligned}$$

The norm of the input is $(y_{2k}^2 + y_{2k-1}^2)$, so the norm of the output and the input are equalized if $2\alpha^2 = 1$. Hence, $\alpha = 1/\sqrt{2}$. By this normalization step, wavelet coefficients and father wavelet coefficients can be calculated using Equations (2.8) and (2.9), respectively:

$$d_k = (y_{2k} - y_{2k-1})/\sqrt{2}, \tag{2.8}$$

$$c_k = (y_{2k} + y_{2k-1})/\sqrt{2}, \tag{2.9}$$

and we can generalize Equation (2.8) as

$$d_k = \sum_{\ell=-\infty}^{\infty} g_\ell y_{2k-\ell},$$

where

$$g_\ell = \begin{cases} 2^{-1/2}, & \text{for } \ell = 0, \\ -2^{-1/2}, & \text{for } \ell = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Example 2.2.2. *We can rearrange Example 2.2.1 using a normalization step. Wavelet coefficients and father wavelet coefficients for the finest level can be written as*

$$d_{J-1,k} = (y_{2k} - y_{2k-1})/\sqrt{2},$$
$$c_{J-1,k} = (y_{2k} + y_{2k-1})/\sqrt{2}.$$

Thus, the new wavelet coefficients and father wavelet coefficients of vector y at the finest level are

$$d_{j,k} = \left(\frac{\sqrt{2}}{2}, \sqrt{2}, -3\sqrt{2}, -3\sqrt{2}\right)^T,$$
$$c_{j,k} = \left(\frac{5\sqrt{2}}{2}, 9\sqrt{2}, 5\sqrt{2}, 5\sqrt{2}\right)^T,$$

where $j = 2$, and $k = 1, \dots, 4$. The normalized wavelet and father wavelet coefficients for the coarser levels can be defined as

$$d_{j,\ell} = (c_{j+1,2\ell} - c_{j+1,2\ell-1})/\sqrt{2},$$
$$c_{j,\ell} = (c_{j+1,2\ell} + c_{j+1,2\ell-1})/\sqrt{2},$$

where $j = J-2, J-1, \dots 0$, and $\ell = 1, 2, \dots, n/2^{J-j}$. Thus, coefficients for the coarser levels are

$$d_1 = (6.5, 0)^T, \qquad c_1 = (11.5, 10)^T,$$
$$d_0 = \frac{-1.5\sqrt{2}}{2}, \qquad c_0 = \frac{21.5\sqrt{2}}{2}.$$

Thus, for Example 2.2.2, $d = (\frac{\sqrt{2}}{2}, \sqrt{2}, -3\sqrt{2}, -3\sqrt{2}, 6.5, 0, \frac{-1.5\sqrt{2}}{2}, \frac{21.5\sqrt{2}}{2})^T$, and the norm of the input y equals to the norm of the output d. The plot of wavelet coefficients by levels is given in Figure 2.1.

2.2.2 Matrix representation

Our output vector is calculated using an input vector with some basic steps which are explained in Section 2.2.1. Thus, we can compute the output vector from the input vector

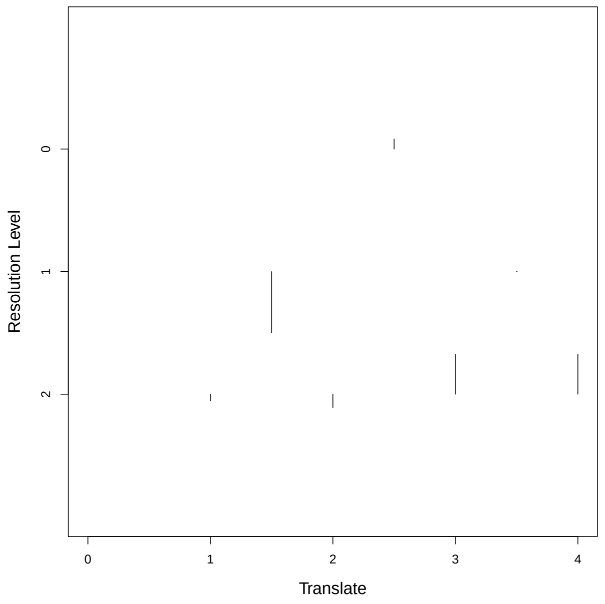


Figure 2.1: Wavelet coefficients plot for the DWT of y. The resolution level axis represents the level j. The finest and coarsest levels are at the bottom and top, respectively. The small vertical lines on the plot represent the wavelet coefficients which are different from zero. The translate axis represents the location parameter, k. Thus, the translate axis shows the coefficients' place in the original sequence.

by defining the matrix, W. Hence, for $y = (2,3,8,10,8,2,8,2)^T$, the vector which is used in Example 2.2.2, W, is

$$W = \begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1/\sqrt{2} & 1/\sqrt{2} \\ -1/2 & -1/2 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1/2 & -1/2 & 1/2 & 1/2 \\ -\sqrt{2}/4 & -\sqrt{2}/4 & -\sqrt{2}/4 & -\sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 \\ \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 \end{bmatrix}. \tag{2.10}$$

It is easy to see that $d = Wy$. The matrix W is an orthogonal matrix, so

$$W^T W = W W^T = I_n,$$

where I_n is the identity matrix, and it can be shown that

$$\|d\|^2 = d^T d = (Wy)^T (Wy) = y^T W^T W y = y^T I_n y = y^T y = \|y\|^2. \tag{2.11}$$

Thus, the length of the vector d is equal to the length of the vector y, and this relationship is called Parseval's relationship.

The matrix given in Equation (2.10) is unique for the Haar wavelet transformation, so for other wavelets, the matrix W should be chosen for that wavelet transformation. Not all wavelets are in orthogonal form; there are also some non-orthogonal wavelets.

We have done a discussion on DWT using Haar wavelets in this section, and we provide a detailed explanation on Haar wavelets in the following section.

2.3 Haar wavelets

2.3.1 Scaling and translation notation

We assume that $p(x)$ is a known function, where $x \in \mathbb{R}$. The dyadically scaled and translated form of $p(x)$ can be represented as $p_{j,k}(x)$, and for simplicity, we illustrate function $p(x)$ and $p_{j,k}(x)$ as p and $p_{j,k}$, respectively. Thus,

$$p_{j,k} = 2^{j/2} p(2^j x - k),$$

where j and k are integers, and $p_{j,k}$ has the same energy as p.

2.3.2 Fine-scale approximation

In Section 2.2, we only deal with discrete sequences, but in mathematics, wavelets generally work on functions. Assume we have a function $f(x)$, where $x \in [0, 1]$. To find the Haar wavelet transform of function $f(x)$, we need to choose a starting point for the finest-scale. Using the fixed finest-scale, we construct discrete series. While we obtain the wavelet coefficients by doing some calculations using the connected pair of serial elements in discrete wavelet transform, finding wavelet coefficients is different for the Haar wavelets on function. We compute integrals of the function over connected pairs of intervals.

The Haar father wavelet at scale 2^J is $\phi(2^J x)$, where $\phi(x)$ is

$$\phi(x) = \begin{cases} 1, & x \in [0, 1], \\ 0, & \text{otherwise.} \end{cases}$$

The father wavelet coefficients at the finest level, $c_{J,k}$, are

$$c_{J,k} = \int_0^1 f(x) 2^{J/2} \phi(2^J x - k) dx. \tag{2.12}$$

Scaling coefficients, $c_{J,k}$, can also be represented using the translation notation, so

$$\begin{aligned} c_{J,k} &= \int_0^1 f(x) \phi_{J,k}(x) dx & (2.13) \\ &= \langle f, \phi_{J,k} \rangle, & (2.14) \end{aligned}$$

where

$$\phi_{J,k}(x) = 2^{J/2} \phi(2^J x - k).$$

The notation which is given in Equation (2.14) is the inner product notation. To explain the $c_{J,k}$, we should first note that

$$\phi_{J,k}(x) = \begin{cases} 2^{J/2}, & x \in [2^{-J}k, 2^{-J}(k+1)], \\ 0, & \text{otherwise.} \end{cases} \tag{2.15}$$

It is seen that $\phi_{J,k}(x)$ are constant over the interval $I_{J,k} = [2^{-J}k, 2^{-J}(k+1)]$ and zero for other values. The interval of k changes when $f(x)$ is defined on interval $[0, 1]$, so in this case, k changes between 0 and $2^J - 1$. Hence, father wavelet coefficient, $c_{J,k}$, are equal to the integral of $f(x)$ over the interval $I_{J,k}$, and the projects of the function $f(x)$ on the Jth level can be written as

$$f_J(x) = \sum_{k=0}^{2^J-1} c_{J,k}\phi_{J,k}(x). \tag{2.16}$$

2.3.3 Computing coarser-scale c from finer-scale ones

We have discussed how we can find the father wavelet coefficients at the finest level, $c_{J,k}$, but we do not know how to compute the father wavelet coefficients for the next coarser level, $c_{J-1,k}$. Using Equation (2.12), we can compute $c_{J-1,k}$, and the interval for $c_{J-1,k}$ can be defined as $[2^{-(J-1)}k, 2^{-(J-1)}(k+1)]$ (see Equation (2.15)), so

$$\begin{aligned} c_{J-1,k} &= \int_{2^{-(J-1)}k}^{2^{-(J-1)}(k+1)} f(x)\phi_{J-1,k}(x)dx \qquad (2.17) \\ &= 2^{-1/2}(c_{J,2k} + c_{J,2k+1}). \qquad (2.18) \end{aligned}$$

After taking the integral in Equation (2.17), we can define $c_{J-1,k}$ using $c_{J,2k}$ and $c_{J,2k+1}$. Thus, the Haar wavelets can be written as

$$\phi(y) = \phi(2y) + \phi(2y-1). \tag{2.19}$$

Equation (2.19) illustrates that how we change the scales from $J-1$ to J. If we set $y = 2^{J-1}x - k$, Equation (2.19) will take the form

$$\phi(2^{J-1}x - k) = \phi(2^J x - 2k) + \phi(2^J x - (2k+1)).$$

Thus, to find $c_{J-1,k}$, we can use the simple form given in Equation (2.18) instead of computing the integral in Equation (2.13).

2.3.4 The difference between scale approximations-wavelets

Assume we have two different functions at two different levels, $f_0(x)$ and $f_1(x)$. Using Equation (2.16), $f_0(x) = c_{0,0}\phi(x)$, and $f_1(x)$ is

$$\begin{aligned} f_1(x) &= c_{1,0}\phi_{1,0}(x) + c_{1,1}\phi_{1,1}(x) \\ &= c_{1,0}2^{1/2}\phi(2x) + c_{1,1}2^{1/2}\phi(2x-1). \end{aligned}$$

While we go from finer-scale function $f_1(x)$ to coarser-scale function $f_0(x)$, we lose some details, which are

$$f_0(x) - f_1(x) = c_{0,0}\phi(x) - c_{1,0}2^{1/2}\phi(2x) - c_{1,1}2^{1/2}\phi(2x-1). \tag{2.20}$$

We can rewrite Equation (2.20) using $\phi(x)$ defined in Equation (2.19) and $c_{J,k}$ defined in Equation (2.18), so $c_{0,0} = (c_{1,0} + c_{1,1})/\sqrt{2}$. Thus, a simplified version of Equation (2.20) is

$$\begin{aligned} f_0(x) - f_1(x) &= d_{0,0}[\phi(2x) - \phi(2x-1)] \\ &= d_{0,0}\psi(x), \end{aligned} \tag{2.21}$$

where $d_{0,0} = (c_{1,1} - c_{1,0})/\sqrt{2}$, and $\psi(x)$ is the Haar mother wavelet defined by

$$\begin{aligned} \psi(x) &= \phi(2x) - \phi(2x-1) \\ &= \begin{cases} 1, & \text{if } x \in [0, 1/2)\,, \\ -1, & \text{if } x \in [1/2, 1)\,, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

We know f_0 is constant, so we can rewrite Equation (2.21) as

$$f_1(x) = c_{0,0}\phi(x) + d_{0,0}\psi(x). \tag{2.22}$$

If we generalize Equation (2.22), it will take the form

$$\begin{aligned} f_{j+1}(x) &= f_j(x) + \sum_{k=0}^{2^j-1} d_{j,k}\psi_{j,k}(x) \\ &= \sum_{k=0}^{2^j-1} c_{j,k}\phi_{j,k}(x) + \sum_{k=0}^{2^j-1} d_{j,k}\psi_{j,k}(x). \end{aligned} \tag{2.23}$$

2.4 Multiresolution analysis

2.4.1 Multiresolution analysis (MRA)

Let us define the space for functions with detail coefficients at level j on V_j. Functions in these spaces may have detail coefficients in coarser levels, but these spaces also contain detail coefficients in finer resolution levels. Thus, the parameter j represents the resolution level. While increasing j symbolizes finer resolution levels, decreasing j means coarser resolution levels. We can illustrate the relationship between the spaces as

$$\cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots\,. \tag{2.24}$$

When j positively increases, we include a greater number of functions in the space. Almost all functions are included if j goes to positive infinity. However, if j negatively decreases, the number of functions included decreases, so we lose some details. If j goes to negative infinity, we do not include any function.

If $f(x)$ is a member of V_j, $f(2x)$ is a member of the space V_{j+1} because the variation of function changes. This is called *interscale linkage*. However, shifting a function does not affect the space of the function; for example, if $f(x)$ is a member of V_0, $f(x-k)$ is also a member of the same space (V_0).

In Section 2.3, we discussed that we need father wavelet function $\phi(x)$ to construct functions at different levels of detail. Thus, we can say that $\phi(x)$ is an essential feature of V_0, so in general, $\{\phi(x-k)\}_k$ is an orthonormal basis for the space V_0. Due to *interscale linkage*,

$$\{\phi_{j,k}(x)\}_{k\in\mathbb{Z}} \text{ is an orthonormal basis for } V_j.$$

2.4.2 Projection notation

A function $f(x)$ is projected into the space V_j by the projection operator P_j, so the projection is

$$f_j(x) = \sum_{k\in\mathbb{Z}} c_{j,k}\phi_{j,k}(x) = P_j f,$$

where $\{\phi_{j,k}(x)\}_k \in V_j$, and the mother wavelet coefficients, $c_{j,k}$, can be written as

$$\begin{aligned} c_{j,k} &= \int_{-\infty}^{\infty} f(x)\phi_{j,k}(x)dx \\ &= \langle f, \phi_{j,k}\rangle. \end{aligned}$$

2.4.3 The dilation equation and wavelet construction

We presented function spaces in Equation (2.24). In this point, we can describe the *dilation equation* as

$$\phi(x) = \sum_{n\in\mathbb{Z}} h_n \phi_{1,n}(x), \tag{2.25}$$

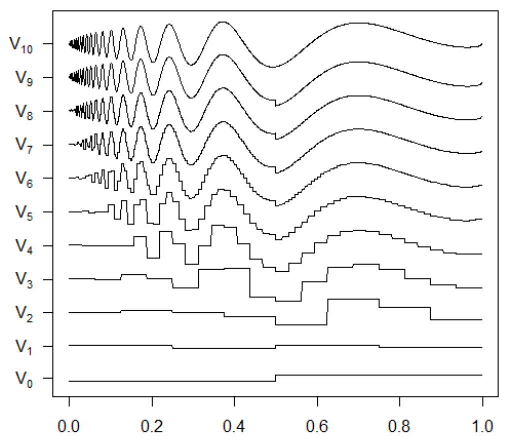
where $\{\phi_{1,k}(x)\} \in V_1$, and $\phi(x) \in V_0$. It is clearly seen that Equation (2.25) is the general form of Equation (2.19) (h_0 and h_1 are equal and $1/\sqrt{2}$ for Haar wavelets). To construct the general MRA, we need the *dilation equation*, so it is an essential equation in wavelets.

Theorem 2.4.1. *(Daubechies, 1992, p. 135) If $\{V_j\}_{j\in\mathbb{Z}}$ with ϕ form MRA of $L^2(\mathbb{R})$, then there exists an associated orthonormal wavelet basis $\{\psi_{j,k}(x) : j,k \in \mathbb{Z}\}$ for $L^2(\mathbb{R})$ such that for $j \in \mathbb{Z}$*

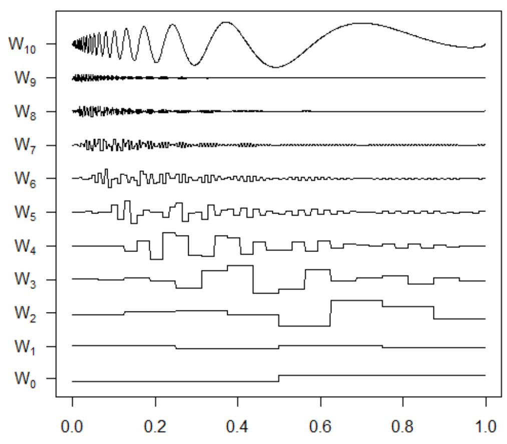
$$P_{j+1}f = P_j f + \sum_k \langle f, \psi_{j,k}\rangle \psi_{j,k}(x). \tag{2.26}$$

One possibility for the construction of the wavelet $\psi(x)$ is

$$\hat{\psi}(\omega) = e^{i\omega/2}\overline{m_0(\omega/2+\pi)}\hat{\phi}(\omega/2),$$



(a) MRA ladder for V_j subspaces.



(b) MRA ladder for W_j subspaces.

Figure 2.2: MRA ladder plot for Doppler function. The function is plotted by subspaces. (a) and (b) illustrate the projection of the Doppler function on $V_0, V_1, \ldots, V_9$ and $W_0, W_1, W_2, \ldots, W_9$ subspaces from bottom to top, respectively, and the top lines in both figures illustrate the original Doppler function.

where $\hat{\psi}$ and $\hat{\phi}$ are the Fourier transforms of ψ and ϕ, respectively, where

$$m_0(\omega) = \frac{1}{\sqrt{2}} \sum_n h_n e^{-in\omega}, \tag{2.27}$$

or equivalently

$$\psi(x) = \sum_n (-1)^{1-n} h_n \phi_{-1,n}(x),$$

where $\psi(x)$ is the mother wavelet. The coefficient $(-1)^{1-n}h_n$ can be represented by g_n.

The proof of Theorem 2.4.1 can be found in Daubechies (1992), and we need to note that for the Haar wavelets, $g_0 = h_1 = 1/\sqrt{2}$, and $g_1 = -h_0 = -1/\sqrt{2}$.

We can represent a function in finer scale by summarizing Equations (2.23) and (2.26) as

$$f(x) = \sum_{k\in\mathbb{Z}} c_{j_0,k}\phi_{j_0,k}(x) + \sum_{j=j_0}^{\infty}\sum_{k\in\mathbb{Z}} d_{j,k}\psi_{j,k}(x). \tag{2.28}$$

The first part of the function in Equation (2.28) is $P_j f$, the projection of the function, f at level j, and the second part illustrates the details at all finer levels.

We can illustrate the shape of a function after MRA. MRA ladder of Doppler function, generated by Donoho & Johnstone (1994), is plotted for each subspace and for the difference between subspaces and given in Figures 2.2a and 2.2b, respectively.

2.5 Vanishing moments

A mother wavelet function $\psi \in L^2(\mathbb{R})$ has m *vanishing moments* under the condition of

$$\int x^{\ell}\psi(x)dx = 0, \quad \ell = 0, \ldots, m-1.$$

If a function has m *vanishing moments*, coefficients of that function in those moments will be zero. This means wavelet coefficients are zero or close to zero for a smooth function. Other coefficients are non-zero, and these coefficients break the continuity.

If we have many zero wavelet coefficients, this means we have a sparse set of coefficients. Having a sparse set means we need to estimate just a few coefficients because just a few coefficients are non-zero.

2.6 Daubechies' compactly supported wavelets

Addition to the Haar wavelets, there are many other wavelets such as Shannon wavelets (Chui, 1997), Meyer wavelets and Daubechies' wavelets (Daubechies, 1992). In this section, we briefly discuss Daubechies' wavelets which is one of the well known ones.

The building of the orthogonal wavelets which are compactly supported is one of the significant accomplishments in the wavelet theory, and these wavelets are the outcome of the dilation in Equation (2.25). Daubechies (1992) extended the compactly support of Haar wavelets using more complicated filters. The assumption in these wavelets is that mother wavelet function ψ has N (> 2) vanishing moments ($\int x^{\ell}\psi(x)dx = 0$, $\ell = 0, \ldots, N-1$), and Daubechies' wavelets turn into Haar wavelets when $N = 1$ (see Figure 2.3). We provide the comparison plots of scaling and mother wavelet functions for different vanishing moments for "Extremal-Phase" family and "Least-Asymmetric" family in Figures 2.4 and 2.5, respectively. Both figures illustrate that increasing vanishing moments gives a better compression of signals.

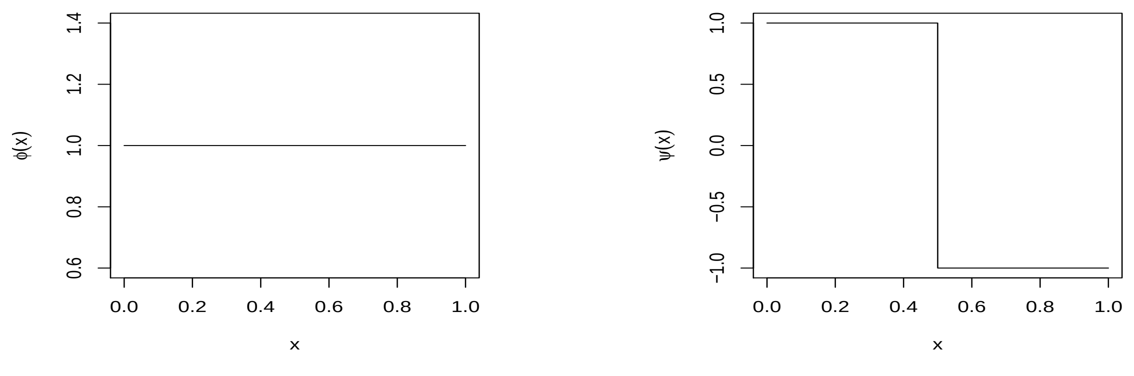


Figure 2.3: Haar wavelets. ϕ: scaling function and ψ: mother wavelet function.

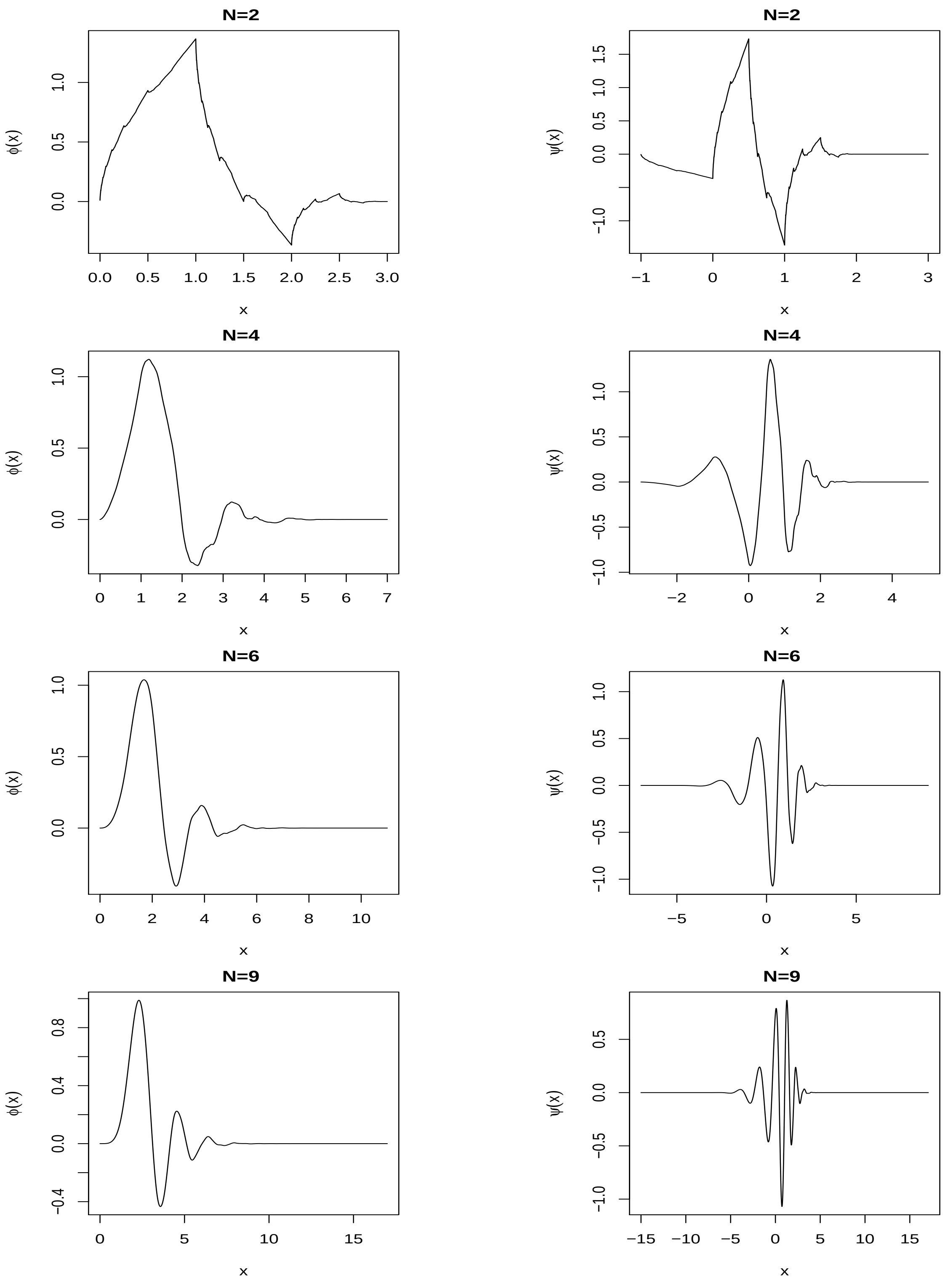


Figure 2.4: Plots of the scaling function ϕ and the mother wavelet function ψ for "Extremal-Phase" family. Each plot is labelled with the number of vanishing moments, N.

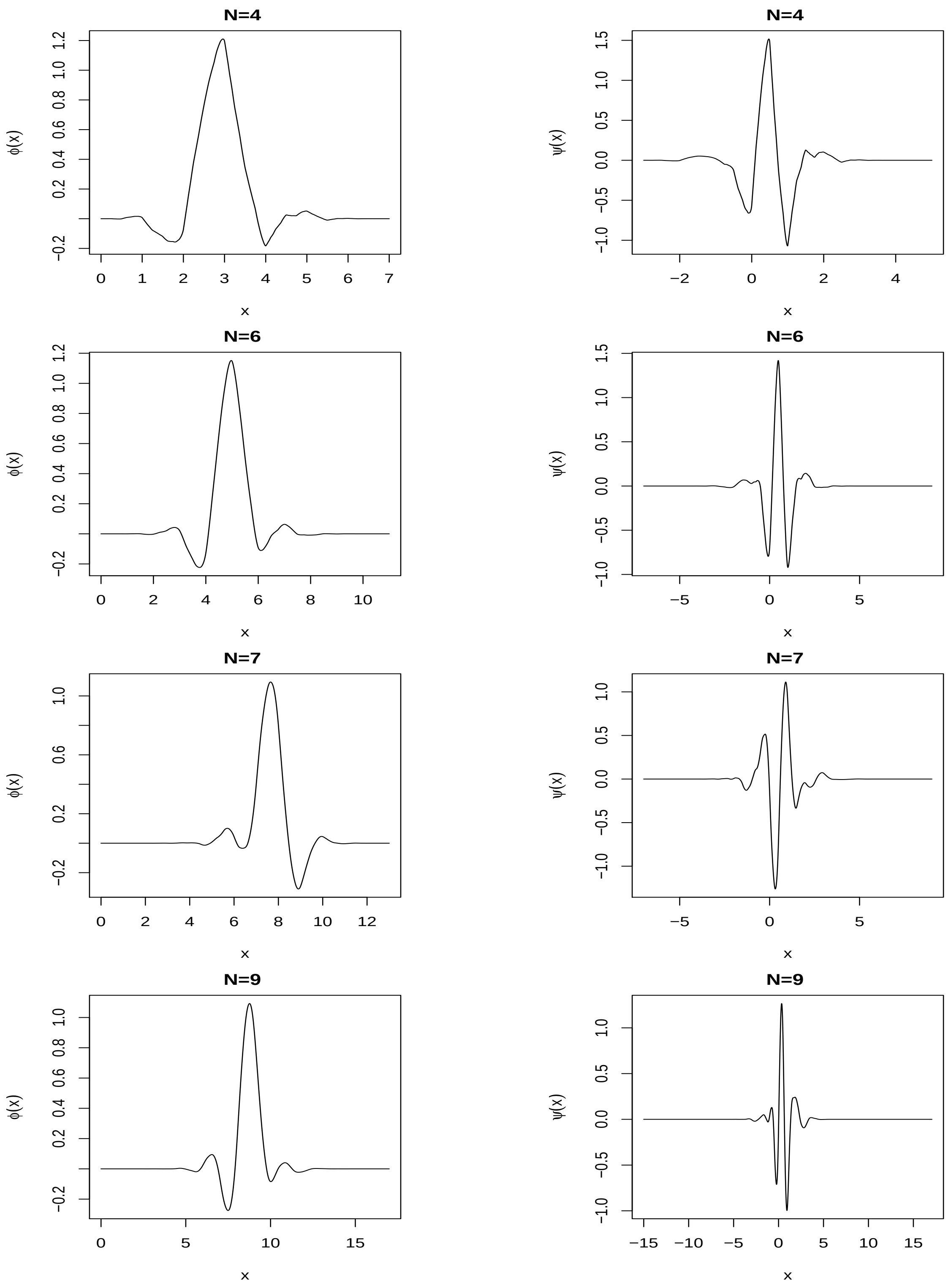


Figure 2.5: Plots of the scaling function ϕ and the mother wavelet function ψ for the "Least-Asymmetric" family. Each plot is labelled with the number of vanishing moments, N.

2.7 The general (fast) discrete wavelet transform

2.7.1 The forward discrete wavelet transform

In Section 2.3, we discussed how to find Haar wavelet coefficients for coarser levels. In this section, we work on computing wavelet coefficients at coarser levels in general form.

For a function $f(x)$, we can compute father wavelet coefficients at level $J-1$, using

$$c_{J-1,k} = \int_{\mathbb{R}} f(x)\phi_{J-1,k}(x)dx, \tag{2.29}$$

where $\{\phi_{J-1,k}(x)\}_k \in V_{J-1}$.

To find $\phi_{J-1,k}(x)$, we work with the dilation equation given in Equation (2.25). In this way, we have an expression for father wavelet coefficients at level $J-1$ in terms of father wavelet coefficients at the finest level J, so we can find the general form of Equation (2.25) as

$$\begin{aligned} \phi_{J-1,k}(x) =& 2^{(J-1)/2}\phi(2^{J-1}x-k) \\ =& \sum_n h_n \phi_{J,n+2k}(x). \end{aligned} \tag{2.30}$$

Now, let us find father wavelet coefficients by substituting Equation (2.30) into Equation (2.29), so

$$\begin{aligned} c_{J-1,k} &= \int_{\mathbb{R}} f(x)\sum_n h_n \phi_{J,n+2k}(x)dx \\ &= \sum_n h_n c_{J,n+2k}, \end{aligned}$$

or

$$c_{J-1,k} = \sum_n h_{n-2k} c_{J,n}.$$

To find wavelet coefficients, we also follow the same steps, but we use mother wavelet function instead of father wavelet function, so wavelet coefficients $d_{j,k}$ are

$$d_{j,k} = \int f(x)\psi_{j,k}(x)dx, \tag{2.31}$$

where $j = 1, \dots, J$. Thus, to find $d_{J-1,k}$, we use Equation (2.31) instead of using Equation (2.25), and after some calculations, we find

$$d_{J-1,k} = \sum_n g_{n-2k} c_{J,n}.$$

2.7.2 Filtering, dyadic decimation, downsampling

Another way to find mother wavelet coefficients at scale $J-1$ is

$$c^*_{J-1,k} = \sum_n h_{n-k} c_{J,n},$$

where c^* is the standard convolution operation. We can compute father wavelet coefficients using *dyadic decimation* or *downsampling* operation, so $c_{J-1,k} = c^*_{J-1,2k}$.

Dyadic decimation operators can be even or odd. In this point, we define even dyadic decimation, and odd dyadic decimation is defined later in Section 2.9.1. Hence, even dyadic decimation, $\mathcal{D}_0$, is

$$(\mathcal{D}_0 x)_\ell = x_{2\ell}.$$

Father wavelet and wavelet coefficients can be found using even dyadic decimation operator, so

$$c_{J-1} = \mathcal{D}_0 \mathcal{H} c_J, \tag{2.32}$$

$$d_{J-1} = \mathcal{D}_0 \mathcal{G} c_J, \tag{2.33}$$

where $\mathcal{H}$ and $\mathcal{G}$ are ordinary filtering operation. For simplicity, in Equations (2.32) and (2.33), vector notation of father wavelet and wavelet coefficients is used (c_J, c_{J-1}, d_{J-1}) instead of individual coefficients. We can find each vector of wavelet and father wavelet coefficients using

$$\begin{aligned} d_j &= \mathcal{D}_0 \mathcal{G} (\mathcal{D}_0 \mathcal{H})^{J-j-1} c_J, \\ c_j &= (\mathcal{D}_0 \mathcal{H})^{J-j} c_J, \end{aligned}$$

where $j = 1, \dots, J-1$.

2.7.3 Inverse discrete wavelet transform

After completing the forward wavelet transform, we continue with the inverse wavelet transform. The inverse transform is also discussed using Haar wavelets. In Section 2.3, we saw that $c_{0,0} = (c_{1,0} + c_{1,1})/\sqrt{2}$, and $d_{0,0} = (c_{1,0} - c_{1,1})/\sqrt{2}$ using the Haar wavelet transform. The general version of father and mother wavelet coefficients in coarser scales are

$$\begin{aligned} c_{j-1,k} &= (c_{j,2k} + c_{j,2k+1})/\sqrt{2}, \\ d_{j-1,k} &= (c_{j,2k+1} - c_{j,2k})/\sqrt{2}. \end{aligned} \tag{2.34}$$

Thus, we can write father and mother wavelet coefficients at coarser scales using father wavelet coefficients at the following finer scale, but the problem is how we can reach $c_{j,2k}$ and $c_{j,2k+1}$. If we solve Equation (2.34), we obtain

$$\begin{aligned} c_{j,2k} &= (c_{j-1,k} - d_{j-1,k})/\sqrt{2}, \\ c_{j,2k+1} &= (c_{j-1,k} + d_{j-1,k})/\sqrt{2}. \end{aligned}$$

The general form of the inverse relationship is

$$c_{j,n} = \sum_k h_{n-2k} c_{j-1,k} + \sum_k g_{n-2k} d_{j-1,k},$$

where h_n and g_n are the filters described in Equation (2.25) and in Theorem 2.4.1, respectively.

We saw the matrix representation for Haar wavelet transform in Section 2.2.2. We know that the matrix W is orthogonal, so $W^T W = I_n$ which means $W^{-1} = W^T$. Thus, W^T can be easily found using the matrix W defined in Equation (2.10).

2.8 Boundary condition

Coefficients near boundaries in general create problems, so we need to deal with those coefficients precisely. However, in Haar wavelets, we do not need to worry about this problem. When we have a dyadic sequence, Haar filters generate another pair of dyadic sequence using those dyadic sequences, so coefficients around boundaries do not make any problem.

Nason (2008) explained the boundary condition by giving a simple example: Assume that we have $x_0, \ldots, x_3$ which is a dyadic decimation vector, and the first element of this vector is $\sum_{k=0}^{3} g_k x_k$. The next coefficient is $\sum_{k=0}^{3} g_k x_{k+2}$ because of the even dyadic decimation. We see that while the first coefficient is the combination of the first four elements, the later one skips the first two elements from the right side each time. The same issue is also occurred when we look at left side, $\sum_{k=0}^{3} g_k x_{k-2}$. As it is seen while $k = 2$ and $k = 3$ cover observations x_0 and x_1, we skip observations x_{-2} and x_{-1} (when $k = 0$ and $k = 1$, respectively). In this example, we just miss two observations, but for greater vanishing moments, skipping some observations create problems.

There are some methods to deal with the boundary problem such as symmetric and periodic end effect. For symmetric end effects, we assume $f(-x) = f(x)$, and $f(1-x) = f(1+x)$, where $x \in [0, 1]$. For example, for the above example, x_{-2} and x_{-1} are equal to x_2 and x_1, respectively. Regarding the periodic end effect, we assume $f(-x) = f(1-x)$. Thus, $f(-0.2)$, and $f(1.2)$ are equal to $f(0.8)$ and $f(0.2)$, respectively.

Until now, we have described how to deal with the boundary problem by adjusting the data set, but there is another way to cope with this problem: changing wavelets without doing anything to data set. Boundary problems are generally occurred because of wavelet coefficients at coarser resolution levels which are big or big and close to boundaries of the defined data interval. One way to adapt wavelets which overlap the boundary of the data set is changing wavelets to satisfy the orthogonality condition (Nason, 2008).

2.9 Non-decimated wavelets

In previous sections, we have described the standard wavet transform, but there is also another wavelet transform called non-decimated wavelet transform (NDWT; Nason & Silverman, 1995). In this section, we provide a discussion on the NDWT using dyadic decimation operators.

2.9.1 The ε-decimated wavelet transform

We described the even dyadic decimation operator $\mathcal{D}_0$ in Section 2.7.2, but we can also use the odd dyadic decimation operator $\mathcal{D}_1$ which is defined as

$$(\mathcal{D}_1 x)_\ell = x_{2\ell+1}.$$

Thus, mother and father wavelet coefficients can be defined as

$$\begin{aligned} d_j &= \mathcal{D}_1 \mathcal{G} (\mathcal{D}_1 \mathcal{H})^{J-j-1} c_J, \\ c_j &= (\mathcal{D}_1 \mathcal{H})^{J-j} c_J, \end{aligned}$$

where $j = 0, \ldots, J-1$.

Either $\mathcal{D}_0$ or $\mathcal{D}_1$ can be used at each level, so if $\mathcal{D}_1$ is used to create the specific orthogonal basis, the basis will be defined by one. However, if $\mathcal{D}_0$ is used, the basis will be defined by zero. Each level can be represented by a sequence length of J, ε, to point which dyadic operation is used, so

$$\varepsilon = \varepsilon_{J-1}, \varepsilon_{J-2}, \ldots, \varepsilon_0,$$

where

$$\varepsilon_j = \begin{cases} 1, & \text{if } \mathcal{D}_1 \text{ is used in level } j, \\ 0, & \text{if } \mathcal{D}_0 \text{ is used in level } j, \end{cases}$$

where $j = 0, \ldots, J-1$. We call this transformation as *ε-decimated wavelet transform.*

In the finest resolution level, after the first regular twist (i.e., setting $x_{k+1} = x_k$ and $x_0 = x_{2^J-1}$), we see the impact of $\mathcal{D}_1$, and then we apply $\mathcal{D}_0$ (i.e., $\mathcal{D}_1 = \mathcal{D}_0 S$, where S is the shift operator). The shift operator S is defined as $(Sx)_j = x_{j+1}$. From this definition, we can write $S\mathcal{D}_0 = \mathcal{D}_0 S^2$, and we can change the shift operator with $\mathcal{H}$ and $\mathcal{G}$. To apply the ε-decimated wavelet transform, we take the DWT of the original data using a specific shifting operator. In addition, we should select an ε which matches with a specific selection of origin in respect of the described basis functions.

2.9.2 Non-decimated (stationary) wavelet transform (NDWT)

In some situations, we may need further information than the standard decimated DWT. In Example 2.2.2, while we compute $d_{2,1} = (y_2 - y_1)/\sqrt{2}$ and $d_{2,2} = (y_4 - y_3)/\sqrt{2}$, we do not have a chance to inspect the difference between y_3 and y_2. If these values are very different to each other, this will cause us to lose some information.

Using the steps from ε-decimated transformation, we shift the original sequence regularly, so our new sequence is $(y_8, y_1, \dots, y_7)$ which allows us to inspect the difference between y_3 and y_2. We can find all possible details by shifting the sequence regularly, but not to lose any information, we need to store wavelet coefficients which we obtain from the original sequence and also from the shifted sequence together.

Using this method, we are able to get missing detail coefficients, but keeping both information from the original sequence and the shifted sequence damages the orthogonality of the transform. In addition, it is unnecessary to store all these coefficients because we can rebuild the original sequence using either the original wavelet coefficients or the shifted coefficients.

Nason (2008) pointed out that the meaning of the NDWT is to hold both even and odd decimation at each resolution level. To the transform, we start with the original sequence, $y_1, \dots, y_n$, then we keep $\mathcal{D}_0\mathcal{G}y$ and $\mathcal{D}_1\mathcal{G}y$. The length of even and odd filtered wavelet coefficients are $n/2$, so the total number of wavelet coefficients at the finest resolution level is $2 \times n/2 = n$.

Father wavelet coefficients at the finest resolution level can be found in the same way, but we use $\mathcal{D}_0\mathcal{H}y$ and $\mathcal{D}_1\mathcal{H}y$ operations. The length of father wavelet coefficients at the finest resolution level is $n/2$, so in total, $2 \times n/2 = n$. We compute mother wavelet coefficients using $\mathcal{D}_0\mathcal{G}y$ and $\mathcal{D}_1\mathcal{G}y$, and father wavelet coefficients are computed using $\mathcal{D}_0\mathcal{H}y$ and $\mathcal{D}_1\mathcal{H}y$. If we continue in this way, at the resolution level j, we will have 2^j packets (groups of coefficients), and each packet includes $2^{-j}n$ number of coefficients. Thus, each resolution level includes $2^{-j}n \times 2^j = n$ wavelet coefficients ($j = 1, \dots, J$, where $n = 2^J$). In total, we have J resolution levels, so we have $J \times n$ coefficients after the NDWT.

In the NDWT, the ordering of coefficients is also important. There are two different ordering methods: time order and packet order. The non-decimated coefficients in time order are

$$(y_2 - y_1), (y_3 - y_2), (y_4 - y_3), \dots, (y_8 - y_7), (y_1 - y_8).$$

Time-ordered data is beneficial in time series analysis because we keep the data order same with the original data.

We can also use packet-ordered data in our analysis. In each resolution level, we have two different packets to produce wavelet coefficients: even decimation $\mathcal{D}_0\mathcal{G}$ and odd decimation $\mathcal{D}_1\mathcal{G}$ packets. Even decimation packets, $\mathcal{D}_0\mathcal{G}$ are

$$(y_2 - y_1), (y_4 - y_3), (y_6 - y_5), (y_8 - y_7),$$

and odd decimation packets, $\mathcal{D}_1\mathcal{G}$ are

$$(y_3 - y_2), (y_5 - y_4), (y_7 - y_6), (y_1 - y_8).$$

Thus, packet-ordered data is helpful in nonparametric regression analysis because every group of coefficients matches with a specific kind of basis element. In addition, it is appropriate to put into use qualifications to all packets. It is also appropriate to integrate packets easily to build estimators.

By this point, we have discussed how to find the wavelet transform of a function, and how to get the inverse wavelet transform. To denoise a function, the forward wavelet transform is followed by a denoising stage, where we apply one of the wavelet shrinkage methods. Then we find the inverse transform using the denoised detail coefficients. Thus, we discuss the idea of the wavelet shrinkage and some of the wavelet shrinkage methods in the following section.

2.10 Wavelet shrinkage

We can write the model, given in Equation (2.1), after wavelet transform as

$$d = d^* + e,$$

where $d = (d_{J-1,1}, \dots, d_{J-1,k}, d_{J-2,1}, \dots, d_0, c_0)^T = Wf(x)$, $d^* = (d^*_{J-1,1}, \dots, d^*_{J-1,k}, d^*_{J-2,1}, \dots, d^*_0, c_0)^T = Wg(x)$, $e = (e_1, \dots, e_n)^T = W\varepsilon$, and W is an orthogonal matrix, where the matrix for the Haar wavelet is defined in Equation (2.10).

Characteristics of the discrete wavelet-transformed model can be summarized in three points. Firstly, d^* is a sparse vector for some functions. Secondly, in terms of the Parseval's relationship defined in Equation (2.11), the energy ($\sum_i g(x_i)^2$) of the function g is equal to $\sum_{j,k} {d^*_{j,k}}^2$, where $\{d^*_{j,k}\}$ are wavelet coefficients. The energy of the function g can be represented with fewer coefficients without losing any information because of the sparsity of wavelet coefficients. After the wavelet transform, the noise is still white noise ($e \sim N(0, \sigma^2 I_n)$ independently) because the noise is expanded to all wavelet coefficients.

If the empirical wavelet coefficients, $\{d_{j,k}\}$, are large, these coefficients include original signals and some noise, but if $\{d_{j,k}\}$ are small enough, those coefficients only contain noise. Thus, to find an appropriate estimate for $\{d^*_{j,k}\}$, $\hat{d}^*$, we need to use one of the wavelet shrinkage methods which remove small coefficients in $\{d_{j,k}\}$ when they are smaller than a *threshold*, λ. Donoho & Johnstone (1994) introduced two different thresholding methods: hard (η_H) and soft (η_S) thresholdings, and these can be defined as

$$\begin{aligned}\hat{d}^* &= \eta_H(d,\lambda)\\ &= d\mathbb{I}\{|d|>\lambda\},\\ \hat{d}^* &= \eta_S(d,\lambda)\\ &= \text{sgn}(d)(|d|-\lambda)\mathbb{I}\{|d|>\lambda\},\end{aligned}$$

where d, $\mathbb{I}$ and λ represent the empirical wavelet coefficients, the indicator function and the threshold, respectively. There are many other wavelet shrinkage methods such as Bayesian wavelet shrinkage, discussed in Section 2.10.3.

To evaluate the accuracy of our estimate, we create an error measure between the estimate $\hat{g}(x)$ and the true function $g(x)$, and we choose the estimate $\hat{g}$ which minimizes this error measure. The most preferred error measure is the integrated squared error (ISE), defined as

$$\hat{M} = \frac{1}{n}\sum_{i=1}^{n}\left[\hat{g}(x_i) - g(x_i)\right]^2.$$

The error term between the true function g and its estimate $\hat{g}$ relies on the specific error vector ε, so our interest is the mean integrated squared error (MISE), $M = \mathbb{E}(\hat{M})$.

2.10.1 The Oracle

Mean integrated squared error (risk) from the function domain is equal to the MISE in wavelet domain (from the Perseval's relation). Thus,

$$\hat{M} = \sum_{j,k}\left[\hat{d}^*_{j,k} - d^*_{j,k}\right]^2. \tag{2.35}$$

Equation (2.35) tells that we can obtain MISE for each coefficient.

We decide which coefficients we keep and which ones we delete using hard thresholding. As we discussed earlier, we delete noisy coefficients, $\{d_{j,k}\}$, less than a threshold, λ. The MISE for one coefficient is

$$\begin{aligned} M(\hat{d}^*, d^*) &= \mathbb{E}\left[(\hat{d}^* - d^*)^2\right]\\ &= \begin{cases} \mathbb{E}(e)^2 & = \sigma^2 \quad \text{if } |d|>\lambda\\ & = d^{*2} \quad \text{if } |d|<\lambda. \end{cases}\end{aligned} \tag{2.36}$$

Equation (2.36) shows that if $d \gg \sigma$, we will expect the first choice is true ($|d|>\lambda$), and this can be obtained by picking a small λ. However, if $d \ll \sigma$, we will expect that the second choice is true ($|d^*|<\lambda$), and in this case, we need to pick a large λ. The threshold is chosen as σ to minimize the MISE, but we generally do not know σ which means the optimal risk is unreachable.

Let suppose that there is an oracle, and it explains which of the d_i are around zero. In this case, the oracle tells us whether to delete the ith coefficient or keep it. Thus,

$d_i^* = d_i\partial_i$, where $\partial_i = 0$ or 1 (from the oracle). If we accept the oracle, it makes us choose the smallest d^{*2} or σ^2 for each coefficient. When we use the oracle, Donoho & Johnstone (1994) formed the ideal risk to be

$$M_{ideal} = \sum_{j,k} \min(|d_{j,k}^*|^2, \sigma^2).$$

We usually do not know $\{\partial_i\}$, so we do not have an oracle. In this case, Donoho & Johnstone (1994) showed that if we do wavelet shrinkage using soft thresholding, and if we choose the threshold as $\sigma\sqrt{2\log(n)}$ (which is called *universal threshold*), then the resulting risk $M_{universal}$ is

$$M_{universal} \leq (2\log(n) + 1)(\sigma^2 + M_{ideal}). \tag{2.37}$$

2.10.2 Universal thresholding

Donoho & Johnstone (1994) defined the universal threshold as

$$\lambda^u = \sigma\sqrt{2\log(n)}.$$

In real studies, σ (noise level) is estimated by $\hat{\sigma}$ because the magnitude of the noise is not known.

The universal thresholding risk is given in Equation (2.37), but the usage of the universal threshold, λ^u, has another explanation. Vidakovic (1999) stated this interpretation by recalling a theorem by Pickands (1967).

Theorem 2.10.1. *Let $X_1, X_2, \ldots, X_n$ be a stationary Gaussian process such that $\mathbb{E}[X_i] = 0$, $\mathbb{E}[X_i^2] = 1$, and $\mathbb{E}[X_i X_{i+k}] = \gamma(k)$. Let $X_{(n)} = \max_{i\in\{1,\ldots,n\}}\{X_i\}$. If $\lim_{k\to\infty}\gamma(k) = 0$, then $X_{(n)}/\sqrt{2\log(n)} \to 1$, almost surely, when $n \to \infty$.*

In addition, Vidakovic (1999) pointed out that if random variables $X_i \sim N(0,1)$ independently ($i = 1, \ldots, n$), then for large n, it can be shown that

$$\mathbb{P}(|X_{(n)}| > \sqrt{c\log(n)}) \approx \frac{\sqrt{2}}{n^{c/2-1}\sqrt{c\pi\log(n)}}. \tag{2.38}$$

Thus, the universal threshold is

$$\lambda^u = \hat{\sigma}\sqrt{2\log(n)}. \tag{2.39}$$

The number 2 is specifically chosen in Equation (2.39). If $c \leq 0$ in Equation (2.38), the right hand side of that equation is likely to be zero. Nason (2008) summarized its meaning in wavelet shrinkage terminology as the largest wavelet coefficients, containing only Gaussian noise, is not greater than the threshold with a high probability.

There are different methods to estimate σ, and these estimations are generally based on wavelet coefficients at the finest resolution level. The finest resolution level involves the 50% of the coefficients which mainly just holds noise, so the signal is not present in that level. This means that *signal-to-noise ratio* (SNR) is low in that level, where SNR is defined as the ratio of standard deviation of signals and noises.

A regular estimator of σ is

$$s = \sqrt{\frac{1}{n/2-1}\sum_{i=1}^{n/2}\left[d_{J-1,i} - \bar{d}_{J-1}\right]^2}, \tag{2.40}$$

where $d_{J-1,i}$ and $\bar{d}_{J-1}$ are the detail coefficients at the finest resolution level and their mean, respectively. Donoho & Johnstone (1994) offered another estimation method for σ. In terms of their method, the estimate of σ, $\hat{\sigma}$, can be computed using the median absolute deviation (MAD) of wavelet coefficients at the finest resolution level. Hence, the MAD can be computed by

$$MAD[d_{J-1}] = b \times MEDIAN\left[\left|\underset{\sim}{d}_{J-1} - MEDIAN\left(\underset{\sim}{d}_{J-1}\right)\right|\right], \tag{2.41}$$

where $\underset{\sim}{d}_{J-1}$ is the vector of detail coefficients at the finest resolution level connected to the multiresolution subspace W_{J-1}, and b is a correction term which reduces the bias on $\hat{\sigma}$. If a data set comes from Gaussian distribution, the correction term, b, is set at 1.4826.

Nason (2008) commented on the suggestion of Donoho & Johnstone (1994) as at the finest resolution level, functions which create problems in practice do not have much signal, so many coefficients just include noise in that level. Thus, we need to carefully estimate σ using the coefficients at the finest resolution level. Since the MAD estimator is robust, having some large signal coefficients at the finest resolution level does not make a great difference to the estimator $\hat{\sigma}$.

Usually, the wavelet shrinkage *oversmooths* as a result of using the universal thresholding. This is a property of *VisuShrink*, which is a combination of the universal threshold and soft thresholding policy. Nason (2008) explained the results of *oversmoothing*: it implies that significant number of true wavelet coefficients are removed or modified, and just a few basis functions are used to build the estimate. However, *oversmoothing* does not always mean that the estimate is very smooth.

2.10.3 Bayesian wavelet shrinkage

In this section, we focus on empirical Bayesian thresholding (EBayes) which was introduced by Johnstone & Silverman (2005). Assume that

$$D_i = d_i^* + e_i,$$

where $i = 1, \ldots, n$ and $e_i \sim N(0,1)$, and the unknown coefficients, d_i^*, are generally zeroes. Thus, one possible D_i are the wavelet coefficients in a certain resolution level, and we continue the discussion on this shrinkage method assuming D_i to be wavelet coefficients. We do all derivations under the condition of $D_i \sim N(d_i^*, 1)$, and if the variance is different than one ($\sigma^2 \neq 1$), we normalize the data by σ. After finding the estimates, we multiply the results by σ.

In Bayesian thresholding, the sparsity is formed by choosing an appropriate prior distribution of d_i^*, and a common choice for the prior distribution of independent d_i^* is

$$f_{prior}(d^*) = (1-\omega)\delta_0(d^*) + \omega\gamma(d^*), \tag{2.42}$$

where ω and γ are the mixing weight and a fixed symmetric density, respectively.

In Bayesian thresholding, one method to find the estimate of d^*, $\hat{d}^*(d;\omega)$, is defined to be the median of the posterior distribution. Thus, we need to find the posterior distribution. Assume that the prior distribution of d^* is the one given in Equation (2.42) and $D \sim N(d^*, 1)$, so we can find the posterior distribution of d^* conditional on D, $f(d^*|D = d)$, where $\hat{d}^*(d;\omega)$ is a monotonic function of d, and $\hat{d}^*(d;\omega) = 0$, if and only if $|d| \leq t(\omega)$, where threshold $t(\omega) > 0$.

The posterior median estimation method is an exact Bayesian process if D_i are independent, but if there is dependence between D_i, we can miss some information. Then we can find the estimate of d^*, $\tilde{d}^*(d;\omega)$, using the mean of the posterior distribution.

An important feature of the empirical Bayesian approach is the selection of the mixing weight ω or in other words, threshold $t(\omega)$. The estimation of ω can be obtained using the marginal maximum likelihood method under the assumption of independence of D_i, and the marginal density of D_i can be defined as

$$(1-\omega)\varphi(d) + \omega g(d),$$

where $g = \gamma \star \varphi$ ($\star$ indicates convolution), and φ stands for the standard normal distribution. By maximizing the marginal log-likelihood, we can obtain the marginal maximum likelihood estimator, $\hat{\omega}$, of ω. Thus, the marginal log-likelihood is

$$\ell(\omega) = \sum_{i=1}^{n} \log\{(1-\omega)\varphi(D_i) + \omega g(D_i)\}.$$

We restrict ω to satisfy the threshold inequality, $t(\omega) \leq \sqrt{2\log(n)}$. We know that $\sqrt{2\log(n)}$ is the universal threshold, defined in Equation (2.39), so the empirical Bayesian threshold can not be greater than the universal threshold.

Since we define the threshold as the posterior median, we first need to define the posterior distribution. Let

$$\theta(d) = \frac{g(d)}{\varphi(d)} - 1.$$

The posterior probability $\omega_{post}(d) = P(d^* \neq 0|D = d)$ is defined as

$$\begin{aligned}\omega_{post}(d) &= \frac{\omega g(d)}{\omega g(d) + (1-\omega)\varphi(d)} \\ &= \frac{1+\theta(d)}{\omega^{-1}+\theta(d)}.\end{aligned}$$

We now know the posterior probability, so we can define the posterior mean and posterior median easily. Let

$$f_1(d^*|D = d) = f(d^*|D = d, d^* \neq 0),$$

so the posterior density is

$$f_{post}(d^*|D = d) = (1-\omega_{post})\delta_0(d^*) + \omega_{post} f_1(d^*|d).$$

Then the posterior mean $\tilde{d}^*(d;\omega)$ is $\omega_{post} d_1^*(d)$, where $d_1^*(d)$ is the mean of the density $f_1(\cdot|d)$.

Lastly, we need to define the posterior median. Let

$$\tilde{F}_1(d^*|d) = \int_{d^*}^{\infty} f_1(v|d)dv.$$

If $d > 0$, $\hat{d}^*(d;\omega)$ is

$$\begin{aligned} \hat{d}^*(d;\omega) &= 0, && \text{if } \omega_{post}(d)\tilde{F}_1(0|d) \leq 1/2, \\ \tilde{F}_1(\hat{d}^*(d;\omega)|d) &= \{2\omega_{post}(d)\}^{-1}, && \text{otherwise.}\end{aligned}$$

The median is zero when $\omega_{post}(d) \leq 1/2$, and under this condition, there is no need to compute $\tilde{F}_1(0|d)$.
If $d < 0$, $\hat{d}^*(-d;\omega) = -\hat{d}^*(d;\omega)$ which is called antisymmetry property.

2.10.4 Non-decimated wavelet shrinkage

We discussed another wavelet transformation, non-decimated wavelet transform (NDWT) in Section 2.9, so we discuss how to apply wavelet shrinkage to non-decimated wavelet transformed data sets in this section.

Translation invariant wavelet shrinkage

The non-decimated wavelet transform (NDWT) is also known as *Translation-invariant* (TI) transform, where the wavelet transform of n cyclic shifts of the original data is taken. Coifman & Donoho (1995) introduced the TI smoothing algorithm which denoises a few cyclic shifts of the data instead of all cyclic shifts and obtains a result close to the one achieved from all cyclic shifts. Nason (2008) explained this algorithm using a small example. Let's say 50 shifts are done to the wavelet basis, and for each shift, wavelet shrinkage is accomplished, then the data are shifted back. Lastly, the mean of the whole shift-denoise-unshift estimates is taken. This method is called ‘cycle spinning’, and the example includes 50 cycle spins. If the number of the cycle spin is n (length of the data), this is called full cycle spinning which is the basis of the TI denoising.

The TI is not a wavelet shrinkage technique, but it can be considered as a *model averaging* method. For each cycle spin, there is a wavelet shrinkage, but there is just one model which explains the data at a certain shift. For every spin, there is one model, so we take the average of all models to reach the result.

Using the NDWT, we increase the possibility of getting a basis which represents the data sparsely. If we use a single basis (if we use just one packet), we might not obtain the best representation of the signal. This is because of the arrangement problem between the signal quality and the wavelet basis function.

In the NDWT, we have n wavelet coefficients in all resolution levels as we explained before in Section 2.9. To compute σ, we use the coefficients at the finest resolution level. The estimate of σ depends more on wavelet coefficients in the NDWT; however, there is correlation between coefficients in this transformation method (it does not matter if the noise is independent). Here, we also work with the universal threshold, $\sqrt{2\log(n)}$ because non-decimated coefficients can be thought as n different bases, and each base is supposed to be thresholded using the universal thresholding method. Nevertheless, this is not an optimal way because we do a sort of multiple hypothesis testing here.

Basis selection

In TI denoising with basis averaging, we get the average of all shifted wavelets, but instead of doing this, we can try to select one basis which works well from shifted wavelets. To understand the importance of this choice, every estimate can be compared with the true function, then the results of different basis choices can be seen. Instead of choosing the best one, we can also choose several which perform well, and we can average them (Nason, 2008).

2.11 Simulation study for wavelets

In this section, we carry out a simulation study where we compare the performance of the DWT and NDWT with different shrinkage methods (universal thresholding and EBayes) using the average mean squared error (AMSE). We also provide box plots of mean squared error (MSE) which help us to see how much variation found by different methods.

To test the performance of wavelet shrinkage methods, Donoho & Johnstone (1994) introduced four different functions: Bumps, Blocks, HeaviSine and Doppler functions, illustrated in Figure 2.6. In this simulation study, we use Blocks function. The estimate

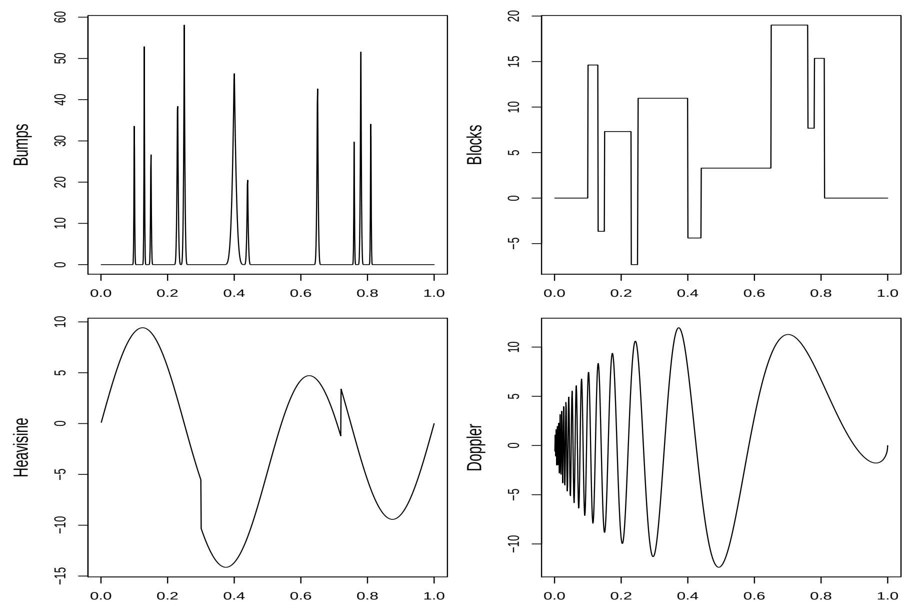


Figure 2.6: Bumps, Blocks, Heavisine and Doppler test functions.

		Vanishing moments					
		1		5		10	
Wavelet	Thresholding	Thresholding type		Thresholding type		Thresholding type	
Transform	method	Hard	Soft	Hard	Soft	Hard	Soft
DWT	Universal	84	665	446	1405	520	1736
	EBayes	92		280		311	
NDWT	Universal	35	418	204	1161	281	1457
	EBayes	54		199		244	

Table 2.1: Different wavelet transformations of Blocks function using "Extremal-Phase" family. AMSE of hard and soft thresholded wavelet and non-decimated wavelet coefficients when we use universal thresholding, and wavelet coefficients after EBayes are given for different vanishing moments. Results are multiplied by 1000.

of variance is found by MAD using the wavelet coefficients at the finest resolution level, and the signal-noise-ratio (SNR) is fixed at 8. In addition, we apply basis averaging in the NDWT. For each function, we apply 1000 replicates.

In Table 2.1, AMSE are given for the DWT and NDWT using Daubechies' family sets as "Extremal-Phase" family of wavelets. We compare the DWT and NDWT using two different thresholding methods: universal (both soft and hard) and EBayes (with posterior median) thresholdings, and we also check their behaviours for different vanishing moments. The DWT and NDWT results show that AMSE for hard-thresholding is always lower than for soft-thresholding for all vanishing moments. Hard or soft thresholding does not have any effect on EBayes approach. If we compare results for different vanishing moments, AMSE of the NDWT is always lower than AMSE of the DWT in both universal and EBayes thresholdings. Overall, the smallest AMSE is found by the NDWT

		Vanishing moments					
		4		7		10	
Wavelet	Thresholding	Thresholding type		Thresholding type		Thresholding type	
Transform	method	Hard	Soft	Hard	Soft	Hard	Soft
DWT	Universal	358	1256	344	1223	450	1371
	EBayes	222		239		275	
NDWT	Universal	126	899	166	1036	196	1106
	EBayes	143		169		188	

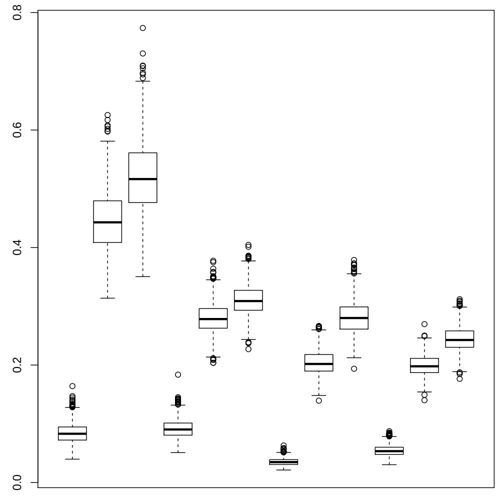
Table 2.2: Different wavelet transformations of Blocks function using "Least-Asymmetric" family. AMSE of hard and soft thresholded wavelet and non-decimated wavelet coefficients when we use universal thresholding, and wavelet coefficients after EBayes thresholding are given for different vanishing moments. Results are multiplied by 1000.

with hard-thresholding when the number of vanishing moments is 1.

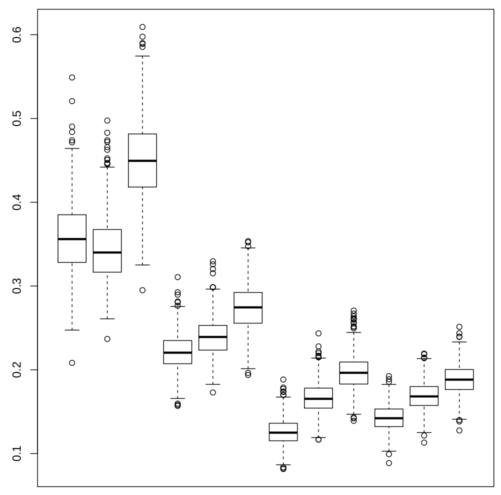
We do the equivalent comparison as in Table 2.1 using Daubechies' "Least-Asymmetric" family of wavelets; results are summarized in Table 2.2. In this case, we also notice that AMSE for hard-thresholding for both the DWT and NDWT using the universal thresholding are always lower than the soft-thresholding ones for all vanishing moments. While EBayes thresholding gives the lowest AMSE for the DWT, AMSE of EBayes and hard universal thresholdings are close to each other for the NDWT. For 4 and 7 vanishing moments in NDWT, hard universal thresholding gives slightly lower AMSE than EBayes threholding. Overall, the smallest AMSE is found by the NDWT with hard-thresholding when the number of vanishing moments is 4.

As an illustration, box plots are drawn, and given in Figure 2.7. For each Daubechies' family, we present two groups of box plots. One group presents the MSE of the DWT and NDWT with hard universal and EBayes thresholding for different vanishing moments, and the same comparison study with soft universal thresolding is illustrated in the second group of box plots. These box plots clearly show the difference between different wavelet transformation methods with different wavelet shrinkage methods.

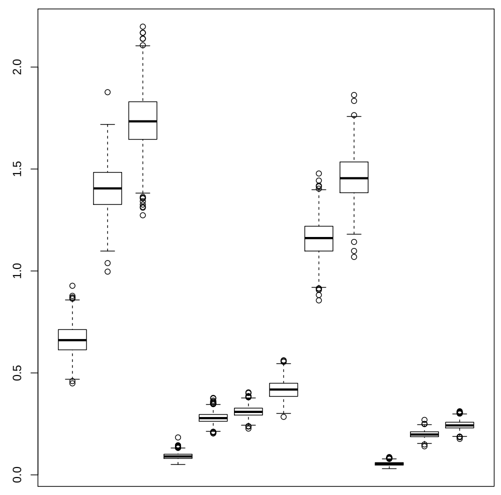
Overall, when vanishing moments increase, we generally obtain higher AMSE in both the DWT and the NDWT with any shrinkage methods and types. Soft-thresholding always finds higher AMSE than hard-thresholding, and in almost all cases, the NDWT with EBayes thresholding returns the smallest AMSE. Thus, the performance of the NDWT with EBayes thresholding is generally higher than the DWT with any thresholding method or the NDWT with universal thresholding.



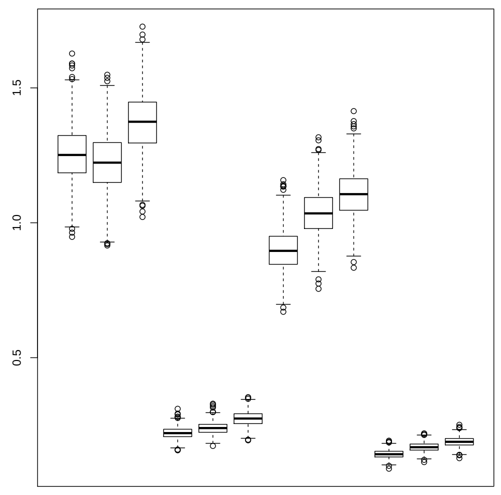
(a) "Extremal-Phase" family with hard-thresholding.



(b) "Least-Asymmetric" family with hard-thresholding.



(c) "Extremal-Phase" family with soft-thresholding.



(d) "Least-Asymmetric" family with soft-thresholding.

Figure 2.7: Box plot comparison for the wavelet simulation study using Blocks function. The vertical axis shows MSE. First 3, second 3, third 3 and last 3 plots are for the following approaches: DWT with universal thresholding, DWT with EBayes thresholding, NDWT with universal thresholding and NDWT with EBayes thresholding, respectively. (a) and (c): each box plot in each group is plotted with vanishing moments 1, 5 and 10, respectively. (b) and (d): each box plot in each group is plotted with vanishing moments 4, 7 and 10, respectively.

Chapter 3

Second generation wavelets: lifting

3.1 Introduction

Wavelet shrinkage is a method to estimate a function from noise-corrupted data. The general form of the model used in wavelet-based function estimation is

$$f(x_i) = g(x_i) + \varepsilon_i, \tag{3.1}$$

where our interest is in $g(x)$, and we assume that $x_i = i/n$, $\varepsilon_i \sim N(0, \sigma^2)$, and $i = 1, \ldots, n$. The general idea in wavelet shrinkage is to first get the wavelet transform of the observed data, and then the wavelet coefficients of $g(x)$ are estimated using a shrinkage method. Finally, the inverse wavelet transformation is used to get the estimate of $g(x)$. Details of each step were described in Chapter 2.

Lifting (second generation wavelets) is an extension of the standard wavelet transform. Lifting deals with irregularly spaced data, and there is no limitation on the number of data points. One feature of lifting is that it is carried out "in-place" which means during the transformation, we can change the old data to the new coefficients. However, in the discrete wavelet transformation, we store new coefficients at each stage. Another difference between lifting and the wavelet transform is the split process. While the wavelet transform splits the data into two sets (scaling and wavelet coefficients) at each level, the researcher decides the number of scaling and wavelet coefficients in lifting. A well known lifting method is "one coefficient at a time" (LOCAAT) proposed by Jansen et al. (2001). In this method, while one group includes just one coefficient which will be lifted, another group includes the rest of the coefficients. Hence, the lifting algorithm finds some coefficients which behave in a similar fashion to wavelet coefficients, so we can easily denoise the lifted coefficients using one of the wavelet shrinkage methods.

Later in Chapter 5, we will propose a new algorithm based on lifting on multidimensional data which finds the clustering pattern in a dendrogram, so it is important to understand the working structure of lifting. Thus, in this chapter, we provide a detailed discussion on the second generation wavelets (lifting). We start our discussion on lifting by describing the general idea behind it in Section 3.2. Then in Section 3.3, the LOCAAT algorithm is summarized in detail. First, the algorithm is explained, and then possible modifications of the algorithm if there is more than one signal in the same grid point are discussed. Finally, an application of the algorithm to a toy example is given. Then we provide a brief discussion on other lifting transformation methods called adaptive and non-decimated lifting transforms in Sections 3.5 and 3.6, respectively. Note that we will not use adaptive lifting in our proposed methods in later chapters, but to see the difference between different lifting methods, it is important to briefly describe this method. This chapter continues with a simulation study in Section 3.7. In this simulation, a comparison between discrete wavelet transform (DWT), non-decimated wavelet transform (NDWT), LOCAAT, adaptive lifting and non-decimated lifting is done in terms of denoising performance on the piecewise polynomial and Donoho and Johnstone (DJ) test functions. Sections 3.2 to 3.7 include details of lifting in one dimension, and we discuss lifting in multiple dimensions in Section 3.8, including lifting on networks which is the base of our proposed algorithm in Chapter 5. Section 3.9 closes the chapter with a detailed example to show how the lifting algorithm works on networks.

3.2 Lifting

Lifting is a recent mathematical method which allows us to apply the wavelet transform to more general data sets. One of the most important properties of lifting is its application to *irregularly spaced* data sets. The lifting algorithm was first introduced by Sweldens (1998) and includes three steps: split, predict and update.

1. *Split*: The observed data $f(x_i)$ are split into two groups: evenly and odd indexed sets.

2. *Predict*: Odd-indexed data are estimated using the evenly-indexed data. Then the detail vector (prediction error) is created. This vector is the difference between the odd-indexed data (observed function values) and estimated values for the same positions.

3. *Update*: The evenly-indexed data values are updated using a linear combination of the observed values for the evenly-indexed data points and the detail vector.

This procedure is repeated, but the user defines how many repetitions are done, effectively choosing the number of non-lifted points. In many applications, the number of non-lifted points is chosen as 2 which is a recommended choice by Nunes & Nason (2004), and we denote the number of non-lifted points by r. Then the signal f is obtained using the rest of the updated observations (scaling coefficients) and detail coefficients which are found during the process. Thus, first $n - r$ entries of signal f represent detail coefficients, and last r entries of signal f represent scaling coefficients.

While the evenly and odd-indexed split works in one dimension, this method does not work in more than one dimension. For two dimensional data sets, Jansen et al. (2001) proposed the LOCAAT algorithm. In terms of the one coefficient at a time process, observations are still split into two groups, but one group includes just one coefficient which is predicted using its neighbours.

3.3 LOCAAT

In this section, first the LOCAAT algorithm which was proposed by Jansen et al. (2001) is described, and then the modification method for multiple observations at one value of x introduced by Nunes et al. (2006) is summarized.

3.3.1 Forward transform of the LOCAAT

In this section, we describe the LOCAAT algorithm in one dimension. Assume that we have a function, $f(x)$, which is observed at n irregularly spaced data points, $x_i \in \mathbb{R}$, where $i = 1, \ldots, n$, and let $f_i = f(x_i)$. Hence, we try to estimate the function g at location x_i using the function values f_i which are corrupted by noise. To do this, we use the LOCAAT algorithm.

Analogously to the wavelet transform, we suppose that the initial function has the form of

$$f(x) = \sum_{k=1}^{n} c_{n,k}\phi_{n,k}(x),$$

where $\phi_{n,k}$ represent the scaling functions which are defined as

$$\phi_{n,k}(x_i) = \delta_{i,k},$$

where $i, k = 1, \ldots, n$. Thus, $f(x_i) = \sum_{k=1}^{n} c_{n,k}\delta_{i,k} = c_{n,i}$ which means that the initial scaling coefficients are taken to be the observed function values.

We discussed in Chapter 2 that the signal f can be written using a combination of wavelet functions and scaling functions, so using this idea, the assumption in lifting is

$$f(x) = \sum_{k=r+1}^{n} d_{j_k}\psi_{j_k}(x) + \sum_{k' \in S_r} c_{r,k'}\phi_{r,k'}(x),$$

where d_{j_k} is the detail coefficient for the point j_k which is defined later in Equation (3.6), r is the number of non-lifted data points, ψ_{j_k} are wavelet functions with zero integral, and S_r is the space for non-lifted data points, where $S_r \subset \{1, \ldots, n\}\backslash\{j_k\}$, and $k = r+1, \ldots, n$. Thus, we form the signal f using the LOCAAT algorithm.

Before starting the lifting transformation, the data points are sorted into increasing order ($x_i < x_{i+1}$). Then we define intervals with widths I (integrated initial scaling functions) by taking the endpoints as the midpoints between initial data points:

$$\begin{aligned} I_{n,1} &= \frac{x_2 - x_1}{2} \times 2 = x_2 - x_1, \\ I_{n,j} &= \frac{x_{j+1} + x_j}{2} - \frac{x_j + x_{j-1}}{2} = \frac{x_{j+1} - x_{j-1}}{2}, \\ I_{n,n} &= \frac{x_n - x_{n-1}}{2} \times 2 = x_n - x_{n-1}, \end{aligned} \tag{3.2}$$

where $j \in \{2, \ldots, n-1\}$, and $I_{n,i}$ is the interval width for ith location in resolution level n.

As shown in Equation (3.2), the first and last interval widths are defined differently than other interval widths because we do not have information for the previous and later points for the first and last points, respectively. Thus, we multiply the width of the first and the last intervals by two.

After defining I, we can start the lifting transformation. The first lifting step is the choice of lifted point, j_n. The point which has the minimum scaling function integral at the finest level is the one which is lifted:

$$j_n = \text{arg } \min_{\ell \in \{1,\ldots,n\}} I_{n,\ell}. \tag{3.3}$$

We use interval widths in Equation (3.2) which represent our integrated scaling functions, so we lift the point which has the narrowest interval. By choosing the smaller integral values, we pick the region which is the most densely sampled, so we lose little information when we remove the point after the update stage. The first coefficient to be lifted is the one in the finest level, then in later stages, we eliminate details in coarser levels.

After choosing the lifted point, we determine its set of neighbours $J_n = \{j_n - 1, j_n + 1\}$ (the first order neighbours). Neighbours are used to predict the value of the function at the point j_n using simple regression techniques, so

$$y_{n,j_n} = \sum_{i \in J_n} a_i^n c_{n,i}, \tag{3.4}$$

where a_i^n are prediction weights obtained from the regression process such as linear interpolation, and $\sum_{i \in J_n} a_i^n = 1$. If the lifted point has one neighbour, then the prediction weight is equal to one ($a_i^n = 1$). However, if the lifted point has two neighbours, then we define

$$a_i^n = \begin{cases} \dfrac{x_{j_n} - x_{j_n - 1}}{x_{j_n + 1} - x_{j_n - 1}}, & \text{where } i = j_n - 1 \\ 1 - \dfrac{x_{j_n} - x_{j_n - 1}}{x_{j_n + 1} - x_{j_n - 1}}, & \text{where } i = j_n + 1. \end{cases} \tag{3.5}$$

After choosing the lifted point and defining its neighbours, we can find the detail coefficient for the point j_n, d_{j_n}, which is

$$d_{j_n} = c_{n,j_n} - y_{n,j_n}, \tag{3.6}$$

or if there is one neighbour,

$$d_{j_n} = c_{n,j_n} - c_{n,i}. \tag{3.7}$$

The update stage only has an effect on scaling coefficients for the neighbouring points. However, we also need to update the interval widths of the neighbouring points. These updates can be done using

$$I_{n-1,i} = I_{n,i} + a_i^n I_{n,j_n}, \tag{3.8}$$

where $i \in J_n$, so we update the interval widths for each neighbouring point.

After updating the interval widths, we can update the scaling coefficients of the neighbours:

$$c_{n-1,i} = c_{n,i} + b_i^n d_{j_n}, \tag{3.9}$$

where $i \in J_n$, and we can find weights b_i^n using the formula

$$b_i^n = \frac{I_{n,j_n} I_{n-1,i}}{\sum_{\ell \in J_n} I_{n-1,\ell}^2}. \tag{3.10}$$

After prediction and update stages, we remove the lifted point, j_n, and the process is repeated. At stage k, $k = n-1, n-2, \dots, r+1$, we

- choose the lifted point, j_k (minimum interval width),
- predict the lifted point,
- update interval widths for neighbours,
- update scaling coefficients for neighbours (predicted neighbours can not be updated, so the next closer neighbour should be updated instead of the predicted one).

We also present pseudo code for the forward LOCAAT transform in Algorithm 1.

3.3.2 Reconstruction of the LOCAAT

The forward lifting transformation is followed by the inverse transformation, so the inverse transformation is described in this section. In forward transform, first we do prediction, and then we update neighbours:

$$\begin{aligned} \text{Prediction step :} \quad d_{j_k} &= c_{k,j_k} - \textstyle\sum_{i \in J_k} a_i^k c_{k,i}, \\ \text{Update step :} \quad c_{k-1,i} &= c_{k,i} + b_i^k d_{j_k}, \end{aligned} \tag{3.11}$$

where $k = n, n-1, \dots, r+1$. For inverse transformation, we go backward, so first we update neighbours, and then we do the prediction. For $k = r+1, r+2, \dots, n$,

$$\begin{aligned} \text{Update step :} \quad c_{k,i} &= c_{k-1,i} - b_i^k d_{j_k}, \\ \text{Prediction step :} \quad c_{k,j_k} &= d_{j_k} + \textstyle\sum_{i \in J_k} a_i^k c_{k,i}. \end{aligned} \tag{3.12}$$

Thus, we invert the transform by reversing the order of the forward transform. We also provide a pseudo code for the inverse LOCAAT transform in Algorithm 2.

Algorithm 1 Forward transform of the LOCAAT algorithm.

```
1: Input: Function values, $f(x)$ and grid points, $x$.
2: Sort the data: $x_i < x_{i+1}$
3: Decide number of non-lifted points, $r$
4: Find the interval widths, $I_n$
5: Forward transform:
6:   for $k = n$ to $r + 1$ do
7:     Choose the lifted point, $j_k = \arg\ \min_{k \in \{1,\ldots,k\}} I_k$
8:     Set neighbourhood space, $J_k$
9:     Prediction step: $d_{j_k} = c_{k,j_k} - \sum_{i \in J_k} a_i^k c_{k,i}$
10:    Update interval widths: $I_{k-1,i} = I_{k,i} + a_i^k I_{k,j_k}$, where $i \in J_k$
11:    Find weights, $b_i^k = \frac{I_{k,j_k} I_{k-1,i}}{\sum_{\ell \in J_k} I^2_{k-1,\ell}}$, where $i \in J_k$
12:    Update neighbours' function values: $c_{k-1,i} = c_{k,i} + b_i^k d_{j_k}$, where $i \in J_k$
13:    Remove $x_{j_k}$
14:  end for
15: Output: $r$, $n$, and list of $j_k$, $J_k$, $a^k$, $b^k$ $d_{j_k}$, and $c_{k-1}$, where $k = n, n-1, \ldots, r+1$.
```

Algorithm 2 Inverse transform of the LOCAAT algorithm.

```
1: Input: Output values of forward LOCAAT transform in Algorithm 1.
2: Reconstruction:
3:   for $k = r + 1$ to $n$ do
4:     Update step: $c_{k,i} = c_{k-1,i} - b_i^k d_{j_k}$, where $i \in J_k$
5:     Prediction step: $c_{k,j_k} = d_{j_k} + \sum_{i \in J_k} a_i^k c_{k,i}$
6:   end for
7: Output: $c$ which is the estimate of $g(x)$.
```

3.3.3 The variance definition of lifting coefficients

We assume that function values $f(x_s)$ are independent random variables with variance Z_s. The variance of the detail coefficients in kth level, d_{j_k}, shown in Equation (3.6) can be defined as

$$\begin{aligned} \operatorname{var}(d_{j_k}) &= Z_{j_k} + \textstyle\sum_{i \in J_k} \left(a_i^k\right)^2 Z_i, \\ \operatorname{cov}(d_{j_k}, c_{k,i}) &= -a_i^k Z_i, \end{aligned} \tag{3.13}$$

where $i \in J_k$, and the variance term for the updated scaling coefficients of the neighbours, $c_{k-1,i}$, shown in Equation (3.9) can be defined as

$$\operatorname{var}(c_{k-1,i}) = Z_i + (b_i^k)^2 \operatorname{var}(d_{j_k}) + 2b_i^k \operatorname{cov}(c_{k,i}, d_{j_k}) = (1 - 2a_i^k b_i^k) Z_i + (b_i^k)^2 \operatorname{var}(d_{j_k}). \tag{3.14}$$

Since $\operatorname{var}(d_{j_k})$ and $\operatorname{var}(c_{k-1,i})$ are applicable for a single lifting step, we update initial variance Z_s for the lifted point j_k and its ith neighbours ($i \in J_k$) after each lifting stage.

Thus, $Z_{j_k} = \mathrm{var}(d_{j_k})$ and $Z_i = \mathrm{var}(c_{k-1,i})$.

3.3.4 Modification for multiple values at a single grid point

If a data set has multiple observations at a single grid point, the lifting algorithm needs some modifications. If we do not modify the algorithm, we will have some zero integrals for scaling functions. Thus, we first remove the point with zero integral, but in reality, that point does not have zero integral. Nunes et al. (2006) introduced a method to deal with multiple values for a single grid point, so we summarize their method in this section. If the removed point has multiple observations, they treat them as coming from different grid points, and they calculate detail coefficients for each point by taking the difference between the observation and the prediction curve. To create one detail coefficient from these multiple detail coefficients, they take the mean of the detail coefficients. They also suggested that we can take the minimum of detail coefficients. If neighbours of the removed point have repeated observations, they estimate the prediction curve using all the observations for neighbours which can be easily done using polynomial regression. To update neighbours, they also use Equation (3.9). After the update stage, the number of multiple observations stays stable. After completing the lifting stage, if there are multiple scaling coefficients for non-lifted points, the mean of the scaling coefficients can be used in the inverse transform. Thus, we have the mean of the detail and scaling coefficients for the grid points which have multiple data points at the end of the forward transformation. Hence, when we apply the inverse transformation, we have one observation (mean of the original function values) at each grid point.

3.4 Example: LOCAAT on one dimensional data

3.4.1 Forward transform

In this section, a toy example in one-dimensional space is used to illustrate the mechanics of the lifting algorithm. Let $x = (0.1, 0.1, 0.3, 0.75, 0.5, 0.9)^T$ and $y = (1, 3, 2, 4, 5, 6)^T$, where T represents the transpose: an illustrative plot of this small data set is given in Figure 3.1. There are five different x values, so the length of the data set is $n = 5$. We take the initial function as our initial scaling coefficients, so $c_n = c_5 = ((1, 3), 2, 4, 5, 6)^T$. Note that grid points, x, are irregularly spaced, and there are two observations at $x = 0.1$ which are illustrated as $(1, 3)$. Before starting the forward transform, we sort the data increasingly in terms of x. So our data points and scaling coefficients become $x = (0.1, 0.3, 0.5, 0.75, 0.9)^T$ and $c_5 = ((1, 3), 2, 5, 4, 6)^T$, respectively. Thus, we start with level $k = n = 5$, and we repeat the algorithm $n - r$ times, where r is the number of non-lifted points which is set to be $r = 2$. We can start the algorithm by calculating interval

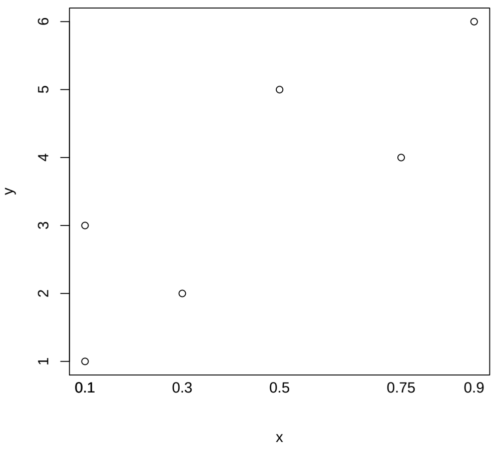


Figure 3.1: Scatter plot of one dimensional toy data.

widths at level $k = 5$:

$$I_{5,1} = x_2 - x_1 = 0.3 - 0.1 = 0.2, \qquad I_{5,2} = (x_3 - x_1)/2 = (0.5 - 0.1)/2 = 0.2,$$
$$I_{5,3} = (0.75 - 0.3)/2 = 0.225, \qquad I_{5,4} = (0.9 - 0.5)/2 = 0.2,$$
$$I_{5,5} = x_5 - x_4 = 0.9 - 0.75 = 0.15.$$

Thus, we find the areas for the different points to be $I_5 = (0.200, 0.200, 0.225, 0.200, 0.150)^T$.

We start our lifting transform from stage 5 ($k = 5$), and lift the point which has the minimum area, so we lift point 5 ($j_5 = 5$), $x_5 = 0.9$, with the area of $I_{5,5} = 0.15$. It has just one neighbour, $c_{5,4} = 4$, so our neighbourhood space at level 5 is $J_5 = \{4\}$, and the prediction weight is $a_4^5 = 1$.

Hence, our prediction for the function value $f(x_5)$ is, from Equation (3.4),

$$y_{5,5} = \sum_{i \in J_5} a_i^5 c_{5,i} = 1 \times 4 = 4,$$

so we can find the detail coefficient from Equation (3.6),

$$d_5 = c_{5,5} - y_{5,5} = 6 - 4 = 2.$$

Thus, our first predicted value at x_5 is 2, so we can remove the point x_5. We should update interval widths for the neighbours using Equation (3.8), so interval widths at level $n - 1 = 4$ are

$$I_4 = (0.200, 0.200, 0.225, 0.350, 0.000)^T.$$

To discriminate the lifted point, we replace the interval width for the lifted point at stage 4 with 0 ($I_{4,5}$). After updating the interval widths, we should calculate the weight (b_i^k) for the update stage using Equation (3.10):

$$b_4^5 = \frac{0.150 \times 0.350}{0.350^2} = 0.43,$$

and we can update the neighbours using Equation (3.9):

$$c_{4,4} = c_{5,4} + b_4^5 d_5 = 4 + 0.43 \times 2 = 4.86.$$

Thus, our updated data set is $c_4 = ((1,3), 2, 5, 4.86, 2)^T$.

We repeat the process for the next points to be lifted (next narrowest interval). The minimum interval width is 0.2 at stage $k = 4$, but there are two points with the same interval width, x_1 and x_2. We take the one on the left side, so we lift $x_1 = 0.1$ ($j_4 = 1$). Another problem in this stage is we have two observations $c_{4,1} = (1,3)$ for the point $x_1 = 0.1$, so we need to do some modifications. The mean of the scaling coefficients at $c_{4,1}$ is 2; the neighbourhood space at level 4 is $J_4 = \{2\}$, and the prediction weight is $a_2^4 = 1$. Then

$$y_{4,1} = \sum_{i \in J_4} a_i^4 c_{4,i} = 1 \times 2 = 2,$$

so our prediction for the point x_1 is 2. We can find detail coefficients for both observations using

$$d_1 = c_{4,1} - y_{4,1} = (1,3) - 2 = (-1,1).$$

Note that this is not a vector subtraction; we take the difference of each observation from the predicted curve. In the update stage, the mean of detail coefficients at x_1 (which is 0) is used.

We next update the interval widths at stage $n - 2 = 3$:

$$I_3 = (0.000, 0.400, 0.225, 0.350, 0.000)^T,$$

and $b_2^4 = 0.5$, so updated scaling coefficients are $c_3 = ((-1,1), 2, 5, 4.86, 2)^T$.

For the stage $k = 3$, we lift $x_3 = 0.5$ ($j_3 = 3$) with the interval width $I_{3,3} = 0.225$. The neighbourhood space at level 3 is $J_3 = \{2,4\}$, and the prediction weights are $a^3 = \{0.444, 0.556\}$. The prediction weights, a^3, come from the regression estimate, so we can simply find our weights by

$$\begin{aligned} a_2^3 &= \frac{x_{j_k} - x_{j_k - 1}}{x_{j_k + 1} - x_{j_k - 1}} = \frac{0.5 - 0.3}{0.75 - 0.3} \approx 0.444, \\ a_4^3 &= 1 - a_2^3 \approx 0.556. \end{aligned}$$

Then

$$y_{3,3} = \sum_{i \in J_3} a_i^3 c_{3,i} = 0.444 \times 2 + 0.556 \times 4.86 \approx 3.59,$$

and

$$d_3 = c_{3,3} - y_{3,3} = 5 - 3.59 \approx 1.41.$$

The updated interval widths are

$$I_2 = (0.000, 0.500, 0.000, 0.475, 0.000)^T,$$

and $b_2^3 \approx 0.237$, and $b_4^3 \approx 0.225$, so our updated scaling coefficients are $c_2 = ((-1,1), 2.33, 1.41, 5.18, 2)^T$.

Thus, we find our final output. For the inverse transform, we use the mean of the repeated observations, so our initial vector is $c_2 \approx (0, 2.33, 1.41, 5.18, 2)^T$.

3.4.2 Reconstruction

The forward transform is followed by the inverse transform. The final lifted point was x_3 in the forward transform, so we first update neighbours for this point, and then we predict the scaling coefficient for x_3. Thus, using Equation (3.12), the update stage for the inverse transform is

$$\begin{aligned} c_{3,2} &= c_{2,2} - b_2^3 d_3 = 2.33 - 0.237 \times 1.41 = 2 \\ c_{3,4} &= c_{2,4} - b_4^3 d_3 = 5.18 - 0.225 \times 1.41 = 4.86, \end{aligned}$$

and prediction stage for the inverse transform is

$$c_{3,3} = d_3 + \sum_{i \in J_3} a_i^3 c_{3,i} = 1.41 + 0.444 \times 2 + 0.556 \times 4.86 = 5.$$

After the first inverse step, our output is $c_3 = (0, 2, 5, 4.86, 2)^T$. We repeat the same process for the next inverse step. The previous lifted point in the forward transform was x_1. Since we had two observations for the first grid point, we would take the mean of the detail coefficients at this stage, so $d_1 = 0$. Thus, the updated function value for the neighbour is

$$c_{4,2} = c_{3,2} - b_2^4 d_1 = 2 - 0.5 \times 0 = 2,$$

and prediction stage for the inverse transform is

$$c_{4,1} = d_1 + \sum_{i \in J_4} a_i^4 c_{4,i} = 0 + 1 \times 2 = 2.$$

Our new output is $c_4 = (2, 2, 5, 4.86, 2)$, and the last inverse step is

$$c_{5,4} = c_{4,4} - b_4^5 d_5 = 4.86 - 0.43 \times 2 = 4,$$

and prediction stage for the inverse transform is

$$c_{5,5} = d_5 + \sum_{i \in J_5} a_i^5 c_{5,i} = 2 + 1 \times 4 = 6.$$

Thus, our new output is $c_5 = (2, 2, 5, 4, 6)^T$. As can be seen, we get the original ordered data set with a slight difference. While the first data point had two different observations in the forward transform, we now have one function value for the first data point which is the mean of function values for the first observation in the original data set.

3.5 Adaptive lifting

3.5.1 Introduction

In this thesis, we propose an algorithm to find where exactly clustering occurs in a dendrogram in Chapters 5 , 6 and 7 based on the LOCAAT algorithm described in Section 3.3. Another available lifting method is adaptive lifting which we briefly discuss in this section. The benefits of the adaptive lifting method is that many different prediction methods are tried at each step, and the one which gives the smallest absolute value of the wavelet coefficient is chosen to improve the sparsity. Other advantages are that the adaptive lifting algorithm is computationally efficient, and the algorithm can easily work with multiple observations at one value of x.

The idea behind the adaptive lifting is finding the efficient representative of a signal by setting the 'wavelet functions'. Thus, a few different adaptive lifting methods based on even/odd splits of the data were proposed by Claypoole et al. (2003), Piella & Heijmans (2002), and Trappe & Liu (2000). However, in this section, we discuss the later work depending on the LOCAAT algorithm by Nunes et al. (2006). First, we briefly describe the previous adaptive lifting algorithms.

Claypoole et al. (2003) offered an adaptive lifting algorithm in image compression. In terms of the proposed algorithm, adaptation is done in the prediction step, but they reverse the algorithm steps. First, they do the update stage, so they get scaling coefficients and quantize them. Then using the quantized scaling coefficients, they do prediction using linear predictors from the $(1, N)$ branch of the Cohen-Daubechies-Feauveau family. Hence, detail coefficients are obtained.

Piella & Heijmans (2002) proposed an adaptive lifting algorithm, and they also start the algorithm with the update stage. However, their method does the adaptation in the update stage which is different from Claypoole et al. (2003). The prediction step remains stable.

Trappe & Liu (2000) added adaptiveness into the prediction step. They use Wiener filtering to minimize the l_2-norm of the signal. When used on an AR(2) process which had correlation, this adaptive method was used to remove the correlation. In addition, if the AR(2) process was corrupted by Gaussian noise, this method was used to denoise the process.

3.5.2 Adaptive LOCAAT algorithm

The previous adaptive algorithms explained above were based on general lifting algorithm. In this section, we focus on the adaptive lifting algorithm proposed by Nunes et al. (2006) which is based on LOCAAT. They start the algorithm by ordering the grid points.

Then they set the intervals, and they can decide the point to be lifted. Details of these steps can be found in Section 3.3.

The next step is the prediction step which works differently than LOCAAT. In adaptive lifting, the prediction step is based on three different regression methods (linear, quadratic and cubic regression) and two different neighbour configurations (closest neighbours and symmetrical neighbours). Thus, there are $3 \times 2 = 6$ different choices for the prediction step. They choose the combination of the sort of regression and neighbourhood configuration with the smallest detail coefficient in absolute value.

The final step in the forward lifting transform is the update stage. In this step, they update the intervals and scaling coefficients for neighbours using the same procedure with LOCAAT algorithm. The rest of the scaling coefficients remain the same. Then they replicate these steps for other points to be lifted (choosing which point to lift, compute detail coefficients, update interval widths and scaling coefficients for neighbours).

To invert the lifting transform, it is not enough to know the detail and scaling coefficients. Some information used in the forward transformation should be stored: the type of regression, the order of lifting the points, the location of scaling coefficients which are not lifted, the vector of updated interval lengths, the vector of interval lengths for lifted points and list of neighbours used in each step. Using the stored information, update and prediction stage for the last lifted point can be undone. After updating detail and scaling coefficients, interval lengths and the list of lifted points, they continue taking the inverse transform for the second last lifted point. Inversion continues step-by-step until they undo all points in the list of lifted points.

The adaptive lifting algorithm introduced by Nunes et al. (2006) can also deal with data sets which include multiple 'y' signals at a certain 'x' value. If we have this kind of data set, we need to modify the algorithm. Details of modifications were given in Section 3.3.4. Further information about adaptive lifting can be found in Nunes et al. (2006).

3.6 Non-decimated lifting

3.6.1 Introduction

Another available lifting algorithm is 'non-decimated lifting' (NLT). In Chapter 6, we will propose an algorithm based on NLT which finds the probability of being clustered for each possible cluster in a dendrogram. Thus, it is important to understand the idea behind NLT.

The non-decimated wavelet transform (NDWT) was described in Section 2.9.2, but the wavelet transform has limitations: the length of the data set is supposed to be 2^J, and data grids should be equally spaced. However, real data sets do not generally satisfy these assumptions. In this case, we can use NLT proposed by Knight & Nason (2009). NLT also depends on LOCAAT (Jansen et al., 2001) details of which can be found in Section 3.3.

In NLT, there is no limitation on either the length of the data set or spacing of grid points, and it creates detail coefficients at each location and scale. Even though there are no formal scales in lifting, Jansen et al. (2009) discussed how to set artificial resolution levels in lifting. Since we start the lifting algorithm with sorted data sets, we can allocate the first half of the coefficients with the finest level to the highest level, the first half of the remaining coefficients to the next level and so on. Details of generating artificial levels can be found in Jansen et al. (2009).

3.6.2 The non-decimated lifting algorithm

In this section, we discuss the details of NLT algorithm which is also based on LOCAAT. The difference is in the split step. As we discussed in Section 3.3, the lifted point is the one which has the narrowest interval at each step, but in NLT, the order of lifted points is chosen differently.

Before starting the algorithm, we define paths/trajectories. A path is an ordered list of signals to be lifted. For example, the first path is $T_1 = (x_{o_1}, x_{o_2}, \dots, x_{o_n})$, where n is the number of signals, and $(o_1, o_2, \dots o_n)$ is the permutation of the indices of the signals (permutation of $(1, \dots n)$).

The split step is followed by the prediction step which is exactly the same as in LOCAAT: predict the lifted point using its neighbours and find detail coefficients. The last step is the update step. In this step, we update the neighbours which are used in the prediction step, and update the corresponding interval widths. The inversion step is also the same as in LOCAAT: we invert the algorithm using the reverse order of lifted points.

Setting the path is not the only characteristic of NLT. Another important feature of the algorithm is that we choose P different paths, $T_1, \dots, T_P$. It means we repeat the algorithm P times, and each time we use the path T_p in the split step ($p \in \{1, \dots, P\}$).

How should we choose the paths? We have n signals, and these signals can be ordered $n!$ ways, so we have $n!$ possible of paths. However, it is hard to use all possibilities; because of that, we choose a computationally efficient number of paths (P paths). For each path, we apply the algorithm, and we obtain detail coefficients. It means we obtain several different detail coefficients for each location. If we set artificial levels, we have several wavelet coefficients for each location in each level. On the contrary, we obtain just one wavelet coefficient for each location in NDWT. This is the main difference between NDWT and NLT.

3.6.3 Risk estimation of averaged estimator of g

When we use NLT to solve the nonparametric regression problem, we should apply one shrinkage method to each of the P sets of coefficients obtained from lifting with different paths. Then we need to invert the transform to obtain the estimate of the function g, defined in Equation (3.1) separately for each path, T_p. Knight & Nason (2009) proposed the estimate of g at location $\underline{x}$ by $\hat{g}^{(p)}(\underline{x})$, where $p \in \{1, \dots, P\}$ and the average estimator of g as $\hat{\hat{g}}(\underline{x})$, where they take the mean of $\hat{g}^{(p)}(\underline{x})$ to find the average estimate of g:

$$\hat{\hat{g}}(x_i) = \frac{1}{P} \sum_{p=1}^{P} \hat{g}^{(p)}(x_i), \tag{3.15}$$

where $i \in \{1, \dots, n\}$, and $p \in \{1, \dots, P\}$.

To see the performance of $\hat{\hat{g}}$, we should find the $AMSE$ of $\hat{\hat{g}}$:

$$\begin{aligned} AMSE(\hat{\hat{g}}, g) &= E\left[\frac{1}{n}\sum_{i=1}^{n}\left\{\hat{\hat{g}}(x_i) - g(x_i)\right\}^2\right], \\ &= \frac{1}{n}\sum_{i=1}^{n} E\{\hat{\hat{g}}(x_i) - g(x_i)\}^2. \end{aligned} \tag{3.16}$$

Using Equation (3.15), we can find

$$\hat{\hat{g}}(x_i) - g(x_i) = \frac{1}{P}\sum_{p=1}^{P}\{\hat{g}^{(p)}(x_i) - g(x_i)\}, \tag{3.17}$$

and by combining Equations (3.16) and (3.17), we find

$$\begin{aligned} AMSE(\hat{\hat{g}}, g) = &\frac{1}{P^2}\sum_{p=1}^{P}\frac{1}{n}\sum_{i=1}^{n} E\{\hat{g}^{(p)}(x_i) - g(x_i)\}^2 \\ &+ \frac{1}{P^2}\sum_{t=1}^{P}\sum_{\substack{k=1\\k\neq t}}^{P}\frac{1}{n}\sum_{i=1}^{n} E\left[\left\{\hat{g}^{(t)}(x_i) - g(x_i)\right\}\left\{\hat{g}^{(k)}(x_i) - g(x_i)\right\}\right]. \end{aligned} \tag{3.18}$$

The second part of Equation (3.18) can be written as

$$ACovE(g^{(t)}, g^{(k)}, g) = \frac{1}{n}\sum_{i=1}^{n} E\left[\left\{\hat{g}^{(t)}(x_i) - g(x_i)\right\}\left\{\hat{g}^{(k)}(x_i) - g(x_i)\right\}\right], \tag{3.19}$$

where $t \neq k \in \{1, \dots, P\}$, and $ACovE$ represents the average covariance error. Estimators $\hat{g}^{(t)}$ and $\hat{g}^{(k)}$ should be unbiased estimators of g, so we can show $ACovE(g^{(t)}, g^{(k)}, g) = ACovE(g^{(k)}, g^{(t)}, g)$ for all $t \neq k$. By combining Equations (3.16), (3.18) and (3.19), we can rewrite the formula for $AMSE$ as

$$AMSE(\hat{\hat{g}}, g) = \frac{1}{P^2}\sum_{p=1}^{P} AMSE(\hat{g}^{(p)}, g) + \frac{1}{P^2}\sum_{t=1}^{P}\sum_{\substack{k=1\\k\neq t}}^{P} ACovE(g^{(t)}, g^{(k)}, g). \tag{3.20}$$

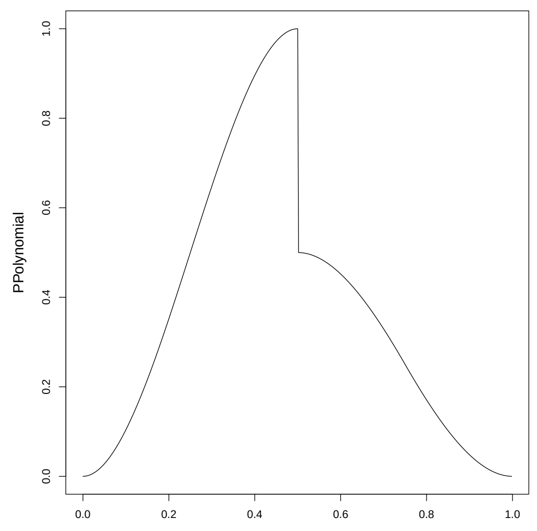


Figure 3.2: Piecewise polynomial (PPolynomial) test function.

While the first term in the overall risk given in Equation (3.20) represents the risks of the separate estimators $\hat{g}^{(p)}$, where $p \in \{1, \dots, P\}$, the second term represents the covariance structure of estimators. Thus, the covariance structure of the estimators does have a place in finding the overall risk.

So far we have summarized three different lifting methods: LOCAAT, adaptive lifting and NLT. We can check the performance of each method using some artificial data sets, so we carry out a simulation study in the following section to compare the behaviour of various lifting methods and wavelet transforms.

3.7 Simulation study

In this section, we carry out a simulation study using different wavelet transforms (discrete wavelet transform (DWT) and non-decimated wavelet transform (NDWT)) and lifting methods (LOCAAT, adaptive lifting and NLT). DWT, NDWT, adaptive lifting and NLT are available in **R**, so we use **wavethresh** package (Nason, 2016) for DWT and NDWT, **adlift** package (Nunes & Knight, 2017) for adaptive lifting and **nlt** package (Knight & Nunes, 2012) for NLT, and LOCAAT algorithm is implemented by ourselves.

In this study, we compare wavelets and lifting methods using DJ functions (Blocks, Bumps, Heavisine and Doppler) generated by Donoho & Johnstone (1994) and the piecewise polynomial function (PPolynomial) generated by Nason & Silverman (1994) in terms of their average mean squared error (AMSE). DJ functions were illustrated in Section 2.10.2 (Figure 2.6), and PPolynomial function is shown in Figure 3.2. We use the Haar wavelet transform for both DWT and NDWT and empirical Bayesian thresholding given in Section 2.10.3 (available in package **EbayesThresh**; Silverman, 2012) for each method. The signal-noise-ratio (SNR) is fixed at 8, and the number of paths, P, for NLT is

		Test functions				
		Blocks	Bumps	Heavisine	Doppler	PPolynomial
Wavelets	DWT	92	266	166	452	0.291
	NDWT	54	170	70	211	0.120
LOCAAT		229	294	113	238	0.176
Adaptive lifting		160	261	80	222	0.121
NLT		117	157	67	138	0.088

Table 3.1: Comparison of wavelet transforms and lifting methods. Different wavelet transforms (DWT and NDWT) and various lifting methods (LOCAAT, Adaptive lifting and NLT) are compared in terms of AMSE. Results are multiplied by 1000.

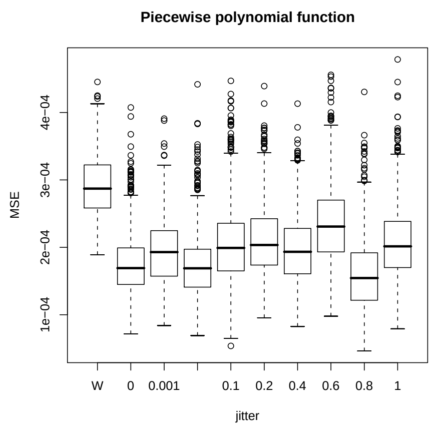


Figure 3.3: The comparison of PPolynomial function after adding different amount of jitter. Jitter axis shows how much jitter is added. The Haar wavelet transformation with empirical Bayesian thresholding is labelled with jitter=W. When jitter= 0*, there is no jitter added to the LOCAAT algorithm. From left to right, jitters are* $0.001, 0.01, 0.1, 0.2, 0.4, 0.6, 0.8$*, and* 1.

set at 10. For each function, we have applied 1000 replicates, and results are summarized in Table 3.1.

The results in Table 3.1 depict that while the smallest AMSE for Blocks function is found by NDWT, NLT finds the smallest AMSE for other functions. Within the lifting methods, NLT always finds the smallest AMSE, and adaptive lifting follows the NLT, and NDWT gives lower AMSE values than DWT for all test functions.

Another simulation compares results from the LOCAAT algorithm with empirical Bayesian thresholding after adding some jitter to the grid points, creating 1000 replicate data sets. Different amount of jitter is added, and the results are compared by box plots in Figures 3.3 and 3.4. DWT results for each function is also added to the box plot comparison. In this comparison study, DWT also finds much smaller AMSE for the Bumps function and much higher AMSE for the Doppler function. Each box plot for each func-

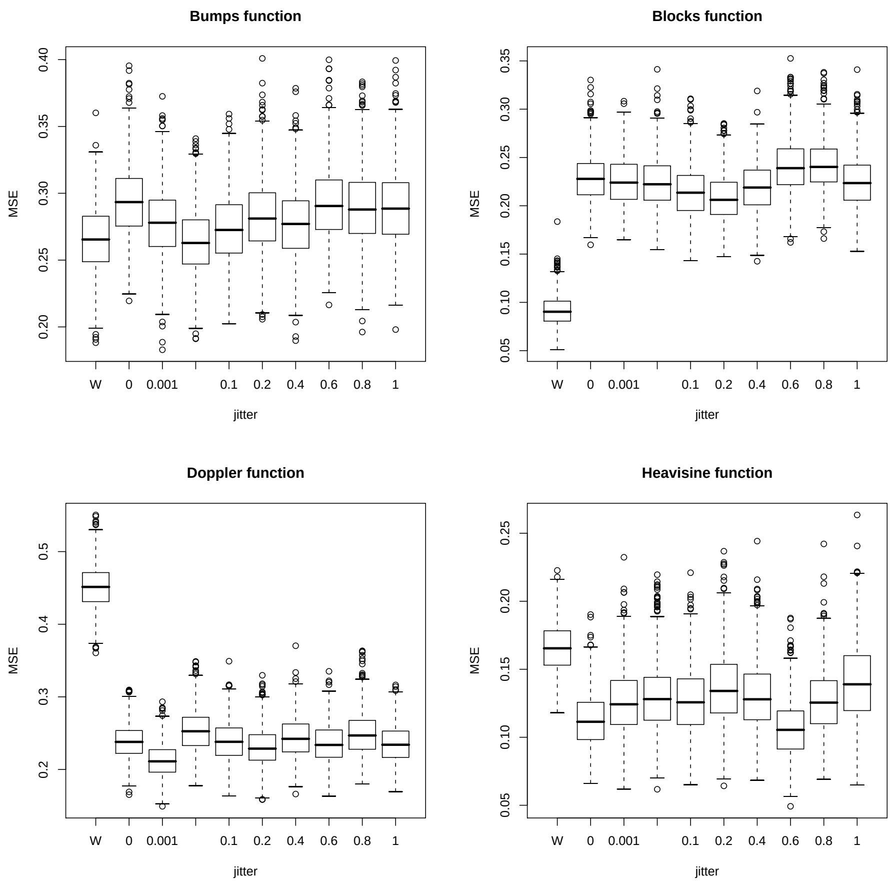


Figure 3.4: The comparison of DJ functions after adding different amount of jitter. Jitter axis shows how much jitter is added. The Haar wavelet transformation with empirical Bayesian thresholding is labelled with jitter=W. When jitter= 0*, there is no jitter added to the LOCAAT algorithm. From left to right, jitters are* $0.001, 0.01, 0.1, 0.2, 0.4, 0.6, 0.8$*, and* 1.

tion illustrates that AMSE slightly changes as the amount of jitter changes, but there is no clear pattern. It is also seen from figures that jittering does not have any effect on the accuracy of the estimates obtained from denoising the lifted data.

3.8 Lifting on multidimensional data

In previous sections, we discussed lifting methods if we have data sets with one-dimensional locations, $x \in \mathbb{R}$. However, the lifting method can be used for any "spatial-like" data, $x \in \mathbb{R}^d$. The only thing we need is to define the neighbourhood structure. Thus, in this section, we discuss the lifting method based on Voronoi-polygons (for two dimensional data) and the lifting method depending on trees and graphs proposed by Jansen et al. (2009).

3.8.1 Lifting in two dimensions

The method of lifting in two dimensions proposed by Jansen et al. (2009) is also based on the LOCAAT algorithm. Before explaining the algorithm, we need to give a brief explanation of the terms which we use.

Assume that we have a set of data sites in the plane, and the appropriate region of the plane is shown by Ω. A set of points in Ω which are closer to a specific data site than any other create the Voronoi cell for that specific data site. The boundary of Voronoi cells is created by sketching a perpendicular line from all midpoint of lines bordering two data sites. Two data sites can be neighbours if their Voronoi cells share an edge, and all neighbours of the site create the Delaunay triangulation. In this method, the aim is to find the Delaunay triangulation at each step.

Algorithm steps

If we have two-dimensional data, the application of the LOCAAT algorithm given in Section 3.3 changes slightly. Thus, these differences are listed and discussed in detail:

1. The integral of the initial scaling function $\phi_{k,i}$, $I_{k,i}$, is the area of the Voronoi cell of the data site x_i, where $i \in \{1, \ldots, n\}$, and $k \in \{n, n-1, \ldots, r+1\}$, where r is the number of non-lifted Voronoi cells, defined by the researcher.

2. Choosing the point to lift. We lift the site i whose Voronoi cell has the smallest area ($j_k = i$). Thus, we choose
$$j_k = \arg \min_{i \in \{1,\ldots,n\}} I_{k,i}. \tag{3.21}$$

3. Setting neighbours, J_k, of the lifted site, by choosing all sites whose Voronoi cells share an edge with that of site x_{j_k}.

4. Prediction step using neighbour interpolation. When we lift the site j_k with neighbours J_k, the prediction weights are
$$a_i^k = \frac{|W_{j_k,i}|}{|W_{j_k}|},$$
where W_{j_k} represents the area of the j_kth cell, and $W_{j_k,i}$ is the piece of W_{j_k} which is created by points whose closest site is site i after site j_k, where $i \in J_k$. To predict the value of the site j_k and the detail coefficient, we use Equations (3.4) and (3.6), respectively.

5. Update stage. In this stage we update integrated initial scaling function values and scaling coefficients for neighbours using Equations (3.8) and (3.9), respectively. Update weights are also given by Equation (3.10).

6. Remove lifted site j_k, and update Voronoi polygons. After site j_k is removed, we update site i, where $i \in J_k$. The part of the Voronoi cell of site i, $W_{j_k,i}$, is the piece of new Voronoi cell of the site i.

7. Go to step 2, and repeat the algorithm $n - r$ times.

3.8.2 Lifting in three or more dimensions

Assume that we have a data set with multidimensional locations, $x \in \mathbb{R}^d$, where $d > 2$. In this case, Voronoi polygons are hard to use because the number of neighbours is high for each point, and this is computationally infeasible. That is why we should use another approach proposed by Jansen et al. (2009). They suggest lifting based on trees and graphs. As a basis of their approach, they use minimal spanning tree (MST) because of computational efficiency, but any tree basis can be used.

Algorithm steps

The lifting scheme on trees and graphs also depend on the LOCAAT algorithm. In this section, we discuss how we can arrange the algorithm if we have tree or graph based data sets:

1. Defining the initial scaling function. We start by defining scaling function, ϕ_{ki}, and its integral. The scaling function is defined as

$$\phi_{k,i} = \begin{cases} 1, & \text{at node } i \\ 0, & \text{at other nodes,} \end{cases}$$

   where $i \in \{1, \dots, n\}$, $k \in \{n, n-1, \dots, r+1\}$, where r is the number of non-lifted data points which is defined by the researcher, and n is the number of nodes (the sum of the number of leaves, internal nodes and the root) on the tree. We define the initial integrated function as sum of the weighted function value at node i. Our weights in tree based data set are based on the lengths of the edges between node i and its immediate neighbours, and the initial integrated function, $I_{k,i}$, is the sum of the edge lengths between node i and its immediate neighbours.

2. Determining lifting point, j_k. At each stage k, we lift node i with the smallest $I_{k,i}$.

3. Setting neighbours. We can set neighbours at stage k, J_k, as immediate neighbours of node i, or we can even include second-order or higher-order neighbours to the J_k.

4. Prediction of wavelet coefficients. To predict the wavelet coefficient at node j_k, we need to find prediction weights, a_i^k, where $i \in J_k$. Jansen et al. (2009) used

inverse distance prediction weights to calculate prediction weights, so they defined $a_i^k = s\delta_{j_k,i}^{-1}$, where $\delta_{j_k,i}$ is the distance between node j_k to its ith neighbour, and s is a scalar which makes $\sum_{i \in J_k} a_i^k = 1$. Equations (3.4) and (3.6) are used to find the estimate of function value and detail coefficient at node j_k, respectively. If node j_k has just one neighbour, i, the value at node i is taken as the prediction value at node j_k.

5. Update stage. We update initial integrated scaling function, $I_{k-1,i}$, and update weights, b_i^k, using Equations (3.8) and (3.10), respectively.

6. Update the neighbourhood form. We exclude the lifted point, j_k, and we adjust the spanning tree. To do this, we change the link between nodes which are straight link to node j_k. Let's say that lifted node j_k has neighbours which are indexed by $i_1, \ldots, i_m$. After removing node j_k, we renew the link between node j_k and neighbours i_ℓ ($\ell \in \{1, \ldots, m\}$) by the connection of the minimum spanning tree of neighbours indicated by i_ℓ ($\ell \in \{1, \ldots, m\}$).

7. Go to step 2, and repeat the algorithm $n - r$ times.

To illustrate, we use a tree structured toy data set and apply each step of the forward and inverse transform in detail in Section 3.9.

3.8.3 Modification for multiple values at a single node

We described in Section 3.3.4 how the LOCAAT algorithm deals with a single grid point when it has multiple function values. In tree structured data sets, the grid points are both leaves and internal nodes from the tree. Thus, we need to carefully discuss the meaning of having multiple function values for a single node. In some data sets, the edge length between some nodes might be zero which creates the problem of having multiple function values for a single node. In this case, we need to do some modifications.

If the point to be lifted has some neighbours with zero edge length, we still treat the nodes zero distance away as neighbours, so each time we just lift one node. To be able to do this, we set the prediction weights as $a_i^k = s$, where the distance between the lifted node j_k to its ith neighbour is $\delta_{j_k,i} = 0$. Another case is that the initial integrated function values, $I_{k,i}$, might be zero too. This means that the lifted point, x_{j_k}, has multiple data points. In this case, we set the update weights as $b_i^k = 1$, so we add/remove the estimation error (detail coefficient) from the function value for the neighbour which has $I_{k,i} = 0$. If a neighbour has multiple data points, the edge lengths between that node and some of its neighbours are zero. Assume that we set the neighbourhood space from the first order neighbours. Thus, we do not interest in the second or higher degree order neighbours even though they have zero edge lengths from the node in our neighbourhood space. Thus, we

x_i	J_k	e_k	c_k
1	$\{7\}$	$\{0.118\}$	0
2	$\{7\}$	$\{0.118\}$	0
3	$\{9\}$	$\{0.632\}$	0
4	$\{8\}$	$\{0.401\}$	0
5	$\{8\}$	$\{0.401\}$	0
6	$\{10\}$	$\{0.855\}$	0
7	$\{1, 2, 9\}$	$\{0.118, 0.118, 0.514\}$	0.059
8	$\{4, 5, 10\}$	$\{0.401, 0.401, 0.454\}$	0.201
9	$\{7, 3, 11\}$	$\{0.514, 0.632, 2.521\}$	0.255
10	$\{8, 6, 11\}$	$\{0.454, 0.855, 2.298\}$	0.339
11	$\{10, 9\}$	$\{2.298, 2.521\}$	1.254

Table 3.2: The summary of the tree structured toy data given in Table 5.1.

update the neighbours using the same method with the LOCAAT algorithm. At the end of the lifting transformation, we have $n - r$ detail coefficients and r scaling coefficients.

3.9 Example: LOCAAT on tree structured data

3.9.1 Forward transform

For illustrative purposes, we use the toy data which will be given later in Section 5.2 (see Table 5.1), and the joined pairs for each agglomeration step and the corresponding edge lengths will be summarized in Table 5.6. The function values for each node are found as $y_n = (0, 0, 0, 0, 0, 0, 0.059, 0.201, 0.255, 0.339, 1.254)^T$, where the first six entries are for the leaves which are zero. The details of building the tree and how we can find the function values will be explained in Example 5.4.1.

Since there are six leaves ($m = 6$), there are $n = 2m - 1$ nodes, so $n = 11$. We take the initial function values y_n as our initial scaling coefficients, so $c_n = c_{11} = (0, 0, 0, 0, 0, 0, 0.059, 0.201, 0.255, 0.339, 1.254)^T$, and we set the number of non-lifted points to be $r = 2$. We label the nodes with x_i and edge lengths with $e_{k,p}$, where $i \in \{1, 2, \dots, n\}$, $p \in \{1, \dots, h\}$, where h is the length of J_k, and J_k is the neighbourhood space at level k. Thus, we start with level $k = n = 11$, and we repeat the algorithm $n - r$ times. To be able to apply the lifting algorithm, the summary of this tree structured data is tabulated in Table 3.2, and its dendrogram is illustrated in Figure 3.5.

We can start the algorithm by calculating the initial integrated function values for each node, $I_{k,i}$, for level $k = 11$:

$$I_{k,i} = \sum_{p=1}^{h} e_{k,p},$$

where h is the number of neighbours at level k. Thus,

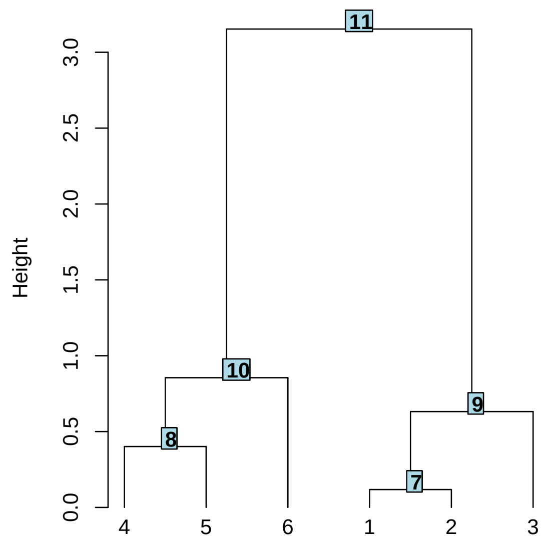


Figure 3.5: The dendrogram of the tree structured toy data given in Table 5.1. Internal nodes are labelled with the agglomeration order starting from $m+1$.

$I_{11} = (0.118, 0.118, 0.632, 0.401, 0.401, 0.855, 0.750, 1.256, 3.667, 3.607, 4.819)^T$. We can next choose the point to be lifted, $j_k = j_{11}$, which is the $\min\{I_{11,i}\}$, so we lift node 1, $j_{11} = 1$, with $I_{11,1} = 0.118$ and the function value, $c_{11,1} = 0$. It has just one neighbour, node 7, with the function value for the node 7, $c_{11,7} = 0.059$. Our neighbourhood space at level 11 is $J_{11} = \{7\}$, and the prediction weight is $a_1^{11} = 1$.

Hence, our prediction for the function value $f(x_1)$ is from Equation (3.4),

$$y_{11,1} = a_1^{11} c_{11,7} = 1 \times 0.059 = 0.059,$$

so we can find the detail coefficient from Equation (3.6),

$$d_1 = c_{11,1} - y_{11,1} = 0 - 0.059 = -0.059.$$

Thus, our first predicted value and detail coefficient at x_1 are 0.059 and -0.059, respectively. We can remove the lifted node 1 and update the linkage between neighbours. Thus, the neighbourhood space for the node 7 includes node $\{2, 9\}$ after updating the linkage since we removed the node 1 from the space. We should update the initial integrated function values for the neighbours using Equation (3.8), so $I_{10,7} = I_{11,7} + 1 \times I_{11,1} = 0.750 + 1 \times 0.118 = 0.868$. Thus, the initial integrated function values at level $n-1 = 10$ are

$$I_{10} = (0.000, 0.118, 0.632, 0.401, 0.401, 0.855, 0.868, 1.256, 3.667, 3.607, 4.819)^T.$$

To discriminate the lifted point, we replace the initial integrated function value for the lifted node at stage 10 with 0 ($I_{10,1}$). Note that we removed the lifted point, so the next node to lift is $j_{10} =$ arg $\min\{I_{10}\} \neq 0$. We need to calculate the weight (b_p^k) for the

k	j_k	J_k	a^k	d_{j_k}	b^k	c_{k-1,J_k}
11	1	$\{7\}$	$\{1\}$	-0.059	$\{0.136\}$	$\{0.051\}$
10	2	$\{7\}$	$\{1\}$	-0.051	$\{0.120\}$	$\{0.045\}$
9	4	$\{8\}$	$\{1\}$	-0.201	$\{0.242\}$	$\{0.152\}$
8	5	$\{8\}$	$\{1\}$	-0.152	$\{0.195\}$	$\{0.122\}$
7	3	$\{9\}$	$\{1\}$	-0.255	$\{0.147\}$	$\{0.218\}$
6	6	$\{10\}$	$\{1\}$	-0.339	$\{0.192\}$	$\{0.274\}$
5	7	$\{9\}$	$\{1\}$	-0.173	$\{0.187\}$	$\{0.186\}$
4	8	$\{10\}$	$\{1\}$	-0.152	$\{0.316\}$	$\{0.226\}$
3	11	$\{10, 9\}$	$\{0.523, 0.477\}$	1.047	$\{0.317, 0.262\}$	$\{0.558, 0.460\}$

Table 3.3: Lifting results for the tree structured toy data given in Table 5.1.

update stage using Equation (3.10):

$$b_1^{11} = \frac{0.118 \times 0.868}{0.868^2} = 0.136,$$

and we can update the neighbours using Equation (3.9):

$$c_{10,7} = c_{11,7} + b_1^{11} d_1 = 0.059 + 0.136 \times (-0.059) = 0.051.$$

Thus, our updated data set is $c_{10} = (-0.059, 0, 0, 0, 0, 0, 0.051, 0.201, 0.255, 0.339, 1.254)^T$.

We repeat the process $n - r$ times. The numerical results for the following stages are calculated and presented in Table 3.3. Thus, we find our final output. For the inverse transform, our initial vector is

$$c_2 = (-0.059, -0.051, -0.255, -0.201, -0.152, -0.339, -0.173, -0.152, 0.460, 0.262, 1.047)^T.$$

3.9.2 Reconstruction

After completing the forward transform, we can follow by our inverse transform. The final lifted node, j_k, was the node 11 in the forward transform, so we first update neighbours for this node, and then we predict the scaling coefficient for the node 11. Thus, using Equation (3.12), the update stage for the inverse transform is

$$\begin{aligned} c_{3,10} &= c_{2,10} - b_1^3 d_{11} = 0.558 - 0.317 \times 1.047 = 0.226 \\ c_{3,9} &= c_{2,9} - b_2^3 d_{11} = 0.460 - 0.262 \times 1.047 = 0.186, \end{aligned}$$

and prediction stage for the inverse transform is

$$c_{3,11} = d_{11} + \sum_{p=1}^{h=2} a_p^3 c_{3,J_{3,p}} = 1.047 + 0.523 \times 0.226 + 0.477 \times 0.186 = 1.254.$$

After the first inverse step, our output is

$$c_3 = (-0.059, -0.051, -0.255, -0.201, -0.152, -0.339, -0.173, -0.152, 0.186, 0.226, 1.254)^T.$$

We repeat the same process for the next inverse step. The previous lifted point in the forward transform was the node 8. Thus, the updated function value for the neighbour is

$$c_{4,10} = c_{3,10} - b_1^4 d_8 = 0.226 - 0.316 \times (-0.152) = 0.274,$$

and prediction stage for the inverse transform is

$$c_{4,8} = d_8 + a_1^4 c_{4,10} = -0.152 + 1 \times 0.274 = 0.122.$$

The updated scaling coefficients at level 4 is

$$c_4 = (-0.059, -0.051, -0.255, -0.201, -0.152, -0.339, -0.173, 0.122, 0.186, 0.274, 1.254)^T.$$

We repeat the inverse transform using the output of the forward transform summarized in Table 3.3 until we reach the finest resolution level $n - 1 = 11$. Thus, our new output at level 11 is $c_{11} = (0, 0, 0, 0, 0, 0, 0.059, 0.201, 0.255, 0.339, 1.254)^T$. As can be seen, we exactly get the same original node values.

Chapter 4

Phylogenetic tree reconstruction

4.1 Introduction

In Chapter 5, we shall develop an algorithm based on the lifting algorithm described in Chapter 3 to decide where we need to cut a tree to find the classification scheme, and we shall introduce a different version of the proposed method, which is applicable to phylogenetic trees, in Chapter 7. Thus, we discuss different phylogenetic tree reconstruction methods available in the literature in this chapter which guide us in Chapter 7. There are some available sources which discuss each step of phylogenetic tree reconstruction in detail such as Durbin (1998) and Isaev (2006).

Phylogenetic trees describe the relationship between species since any set of species are related in terms of evolutionary theory. Evolutionary theory explains the reason of sharing similar DNA sequences by various organisms: common ancestors of these organisms had evolutionary mutations. These mutations can occur in two ways: insertion or deletion of nucleotides from DNA sequences.

Phylogenetic trees can be in two forms: rooted and unrooted trees. Rooted trees illustrate that evolution starts from a single node named the root (ancestor of all current leaves) and continues to the leaves (tips/terminal nodes/operational taxonomic units (OTUs)) via internal nodes (ancestors of specific group of leaves), where a node is the endpoint of an edge. Leaves are labelled using the name of species. While rooted trees give the direction of the evolution, unrooted trees do not. Unrooted trees just illustrate the evolutionary connection among the OTUs.

In phylogenetic trees, if an edge branches, this edge is called a parent edge, and it splits into two daughter edges. The length of the edge is obtained using the product of the length of the time interval and a specific evolutionary rate which indicates how fast the species or genes evolve. The length of the edge symbolizes the dissimilarity between species or sequences. If a tree has a branching pattern, this pattern is called a labelled tree topology. However, such a labelled tree topology does not include the lengths of the branches. In a rooted tree, if there are n leaves, there are $n - 1$ internal nodes, $2n - 1$ nodes and $2n - 2$ edges excluding the root edge.

Phylogenetic trees are usually assumed to be binary while early ones were not. Early trees were constructed using the morphological similarities between OTUs; in the last few decades, however, phylogenetic trees have been constructed using gene and protein sequences. One of the characteristics of gene divergence is that it can occur either because of speciation (a new species occurs at the end of the evolutionary stage) or gene duplication (a part of the DNA, which has a gene, is duplicated). With cases of speciation and gene duplication, diverged genes are named orthologues and paralogues, respectively.

In this chapter, phylogenetic tree construction methods are described. Trees can be built with different methods, so the chapter starts with the main construction steps in Section 4.2. This section is followed by different phylogenetic reconstruction methods, parsimony and distance methods, in Section 4.3. In Section 4.4, evolutionary models are explained which help us to understand a final phylogenetic tree construction method, called probabilistic methods, described in Section 4.5.

4.2 Phylogenetic reconstruction

The main steps to reconstruct phylogenetic trees are summarized by Isaev (2006):

1. Choice of a family of homologous sequences as OTUs. Phylogenetic tree reconstruction starts with protein or gene sequences as we discussed in Section 4.1. How should we choose these sequences? The choice of sequences is important to find an informative tree. Species tend to look like each other if they have strong phylogenetic signals, so these kind of sequences should be chosen. If sequences do not have the strong phylogenetic signals, we may end up with an uninformative tree.

2. Arranging sequences. Sequences need to be aligned, and there are many multiple sequence alignment (MSA) methods and available software tools for them. The most popular progressive alignment software tool is ClustalW (Thompson et al., 1994); some other progressive alignment methods were proposed by Hogeweg & Hesper (1984), Feng & Doolittle (1987), Taylor (1988) and Notredame et al. (2000). There are also iteration based methods. The most commonly referenced software in the literature is MUSCLE (Edgar, 2004), which is similar to MAFFT (Katoh et al., 2002) and PRRP (Gotoh, 1996). Sequences can be aligned using one of these MSA procedures, and reduced aligned sequences are found to start the tree reconstruction. For example, assume that we have DNA sequences for four different species labelled as α, β, γ and δ given in Table 4.1a, and the multiple aligned

α :	ATCATG
β :	ATCAG
γ :	ACTT
δ :	ATCTTT

(a) DNA sequences.

α :	A	T	C	A	T	G
β :	A	T	C	A	-	G
γ :	A	-	C	T	T	-
δ :	A	T	C	T	T	T

(b) Aligned sequences.

α :	ACA
β :	ACA
γ :	ACT
δ :	ACT

(c) Reduced aligned sequences.

Table 4.1: Toy DNA data set with alignment process.

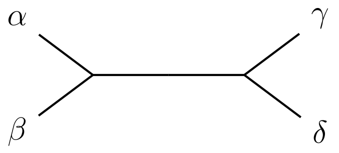


Figure 4.1: Unrooted tree of the toy DNA data set.

sequences in Table 4.1b are obtained. A simple way to reach the reduced multiple alignments is removing gaps from sequence alignments (Table 4.1c).

3. Construction of the tree topology. This step is the most challenging one. From the toy example, it is obvious that we will cluster α and β together and γ and δ together. Thus, we have the topology of the tree given in Figure 4.1, but to find the length of the edge and the place of the root, we need extra information which we discuss in the following sections.

4.3 Phylogenetic reconstruction methods

The number of OTUs in real data sets is large, so analysing these data sets is not easy. Therefore, there are some methods designed to make it easier to analyze reduced aligned sequences. These methods can be categorized in three groups:

1. Parsimony methods,
2. Distance methods,
3. Probabilistic methods (relying on maximum likelihood).

4.3.1 Parsimony methods

Parsimony methods find rooted tree topologies, but they do not find branch lengths. In these methods, the ancestral sequences for the root and internal nodes are found. Basically, the total cost for each possible topology is computed, and the optimal topology to construct the tree is chosen. The optimal topology (called the parsimonious topology) is the one which has the smallest cost.

The simplest way to define the cost function is counting the number of substitutions between root, internal nodes and leaves. Assume that we have some number of sequences. Possible sequences are allocated to the root. Using the minimal number of substitutions between root and internal nodes, new sequences for internal nodes are found whose lengths are the same as the ancestral sequence. This process is repeated until the desired number of leaves is obtained. For each possible topology, the cost function can be found in this way.

Other methods of defining cost functions are using *Fitch's algorithm* (Fitch, 1971) or the *branch and bound algorithm.* If the number of leaves (N) is reasonable, all possible topologies can be found, and the cost function can be calculated using Fitch's algorithm. However, if N is large, finding all possible topologies will not be computationally feasible. In this case, a sample from the topology space is used to discover a nearer optimal topology. Finding cost functions just for the sample space may cause us to miss the optimal topology, so instead of using Fitch's algorithm, the optimal topology can be detected using the branch and bound algorithm when N is large. Details of both Fitch's algorithm and the branch and bound algorithm can be found in Durbin (1998).

4.3.2 Distance methods

The next method in phylogenetic reconstruction is distance methods. The idea is that phylogenetic trees can be constructed by distances between sequences in a reduced multiple alignment, and depending on the method used, either rooted or unrooted trees can be obtained. A well-known way to create distances is to use the "pseudodistances" which are defined later in this section. The distance between ith and jth sequences in the dataset is represented by d_{ij}, and the values $\{d_{ij}\}$ satisfy the conditions

- $d_{ij} > 0,$ for all $i \neq j,$
- $d_{ij} = 0,$ where $i = j,$
- $d_{ij} = d_{ji},$ where $i \neq j,$
- the triangular inequality exists: $d_{ij} \leq d_{ik} + d_{kj},$ where $i \neq j \neq k.$

(4.1)

Using distance methods, branch lengths are also calculated. Popular methods to generate rooted and unrooted trees are the clustering method UPGMA and the neighbour joining algorithm, respectively. These two distance based phylogenetic tree reconstruction methods are summarized in the following subsections.

Clustering method: UPGMA — Hierarchical clustering with average linkage

The clustering method UPGMA (the unweighted pair group method using arithmetic averages) was presented by Sokal & Michener (1958), and UPGMA is the well known statistical clustering method: hierarchical clustering with average linkage. This method is easy to apply. The construction of the tree starts from leaves and continues to the root via internal nodes. Leaves are set at height zero, and UPGMA combines sequences in two clusters in each stage. Hence, a new internal node is added to the tree, and the distance between clusters can be found using

$$d_{ij} = \frac{1}{N(C^i)N(C^j)} \sum_{x \in C^i, y \in C^j} d_{xy}, \tag{4.2}$$

where $N(C^i)$ and $N(C^j)$ are the number of OTUs in clusters C^i and C^j, respectively.

In the first stage, each OTU (x^i, $i \in N$) is appointed to a separate cluster (C^i, $i \in N$), and then two clusters are chosen with the minimum distance, $d(C^i, C^j)$, where $i \neq j$. A new cluster, C^{N+1} is created which includes C^i and C^j, and distances between C^{N+1} and remaining clusters are computed using Equation (4.2). Also a new OTU, x^{N+1} is appointed to the new internal node which is the parent node of x^i and x^j, and distances between x^{N+1} and other OTUs are calculated. At the end of stage one, there are $N - 1$ OTUs. This process is repeated until just two clusters remain. Denote the last two clusters C^k and C^n. Then these two clusters, C^k and C^n, are connected with the root of the tree, and the edges between the root and these two clusters have length $d(C^k, C^n)/2$.

Example 4.3.1. *Assume that we have the distance matrix for four different sequences, $x^1, \ldots, x^4$, and it is given as*

$$\begin{array}{c c} & \begin{array}{cccc} x^1 & x^2 & x^3 & x^4 \end{array} \\ \begin{array}{c} x^1 \\ x^2 \\ x^3 \\ x^4 \end{array} & \begin{bmatrix} 0 & 32 & 12 & 32 \\ 32 & 0 & 32 & 4 \\ 12 & 32 & 0 & 32 \\ 32 & 4 & 32 & 0 \end{bmatrix} \end{array}.$$

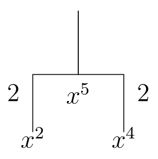
The minimum distance between clusters is $d(C^2, C^4) = 4$, and the new cluster is $C^5 = \{C^2, C^4\}$. The distances between C^5 and other clusters (C^1 and C^3) are

$$d(C^1, C^5) = \frac{1}{1 \times 2}(d(C^1, C^2) + d(C^1, C^4)) = \frac{32 + 32}{2} = 32,$$

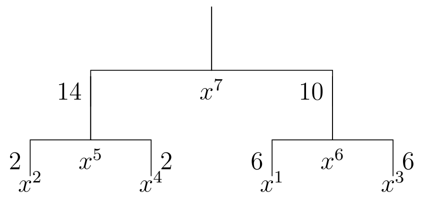
$$d(C^3, C^5) = \frac{1}{1 \times 2}(d(C^3, C^2) + d(C^3, C^4)) = \frac{32 + 32}{2} = 32,$$

so the new OTU x^5 is placed above the OTUs x^2 and x^4 by the height of $d(x^2, x^4)/2 = 2$. The sub-tree is given in Figure 4.2a. We can continue to construct the rest of the tree. The updated distance matrix is

$$\begin{array}{c c} & \begin{array}{ccc} x^1 & x^3 & x^5 \end{array} \\ \begin{array}{c} x^1 \\ x^3 \\ x^5 \end{array} & \begin{bmatrix} 0 & 12 & 32 \\ 12 & 0 & 32 \\ 32 & 32 & 0 \end{bmatrix} \end{array},$$



(a) First stage of UPGMA.



(b) Last stage of UPGMA.

Figure 4.2: Phylogenetic tree reconstruction by UPGMA. The algorithm is based on the distance matrix in Example 4.3.1.

and the minimum distance between clusters is $d(C^1, C^3) = 12$*. The new cluster is* $C^6 = \{C^1, C^3\}$*, and the distances between* C^6 *and* C^5 *is*

$$\begin{aligned} d(C^5, C^6) =& \frac{1}{2 \times 2}(d(C^1, C^2) + d(C^1, C^4) + d(C^3, C^2) + d(C^3, C^4)) \\ =& \frac{32 + 32 + 32 + 32}{4} = 32, \end{aligned}$$

so the new OTU x^6 *is placed above the OTUs* x^1 *and* x^3 *by the height of* $d(x^1, x^3)/2 = 6$*. There are two clusters left:* C^5 *and* C^6*. These two clusters are combined under the cluster* C^7*, and the root (OTU* x^7*) is placed above the OTUs* x^5 *and* x^6 *by the height of* $d(x^5, x^6)/2 = 16$*. Thus, the rooted tree for this data set is given in Figure 4.2b.*

Ultrametric property and molecular clock assumption of distances The UPGMA algorithm constructs a rooted tree, but it may not build a correct tree if the ultrametricity condition does not hold. The distances d_{ij} are called ultrametric for three sequences from the data space, x^i, x^j, x^k if distances d_{ij}, d_{ik} and d_{jk} are either all equal or two of them are equal and the third one is smaller.

The evolution of species or sequences with constant evolution rates through times is explained by rooted trees holding the ultrametric property. This property is named the molecular clock assumption, and if a tree holds this condition, this tree is called a molecular clock tree. Hence, these kind of trees imply that the total time between any node in the tree and leaves does not vary depending on the choice of path. If the molecular clock assumption holds, a correct tree will be obtained by the UPGMA.

Neighbour-joining algorithm

Another assumption of the UPGMA method is additivity. During the discussion of the molecular clock assumption, additivity is also discussed implicitly. Additivity of the distance function d holds if and only if two of the distances $d_{ij} + d_{kl}$, $d_{ik} + d_{jl}$, $d_{il} + d_{jk}$ are equal and greater than the third one for each set of four OTUs x^i, x^j, x^k and x^l. This condition is called the *four-point condition*. In Figure 4.3, the four-point condition is illustrated. The top-left plot illustrates the topology we are interested in. The sum of the distances between species are equal in top-right and bottom-left figures, and they are greater than the final figure (bottom-right one).

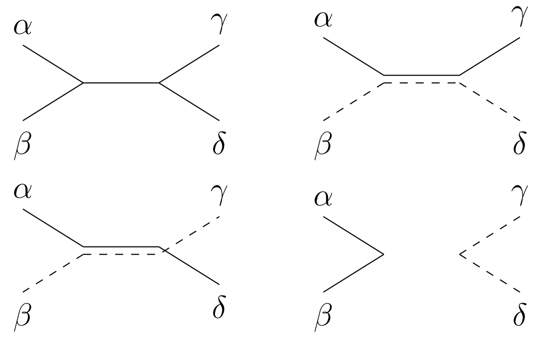


Figure 4.3: Four-point condition. Top-left: The topology we are interested in. Other topologies given in this figure show how many different way we can find the total length of the tree.

In some cases, the molecular clock assumption can fail, but the additivity property can hold. In these cases, instead of reconstructing the tree by the UPGMA method, the neighbour-joining algorithm introduced by Saitou & Nei (1987) and clarified by Studier & Keppler (1988) should be used. The neighbour-joining algorithm works iteratively. In each step, a pair of OTUs is replaced by a new OTU (parent node), and the distance between nodes is computed. This process is repeated until $N = 3$ OTUs remain because there is only one unrooted tree topology for the final three OTUs, and branch lengths for the final tree topology can be found using

$$d_{k\ell} = \frac{1}{2}\left(d_{i\ell} + d_{j\ell} - d_{ij}\right), \tag{4.3}$$

where i and j are OTUs which have the same parent node k, and ℓ represents any other node in the tree. Thus, in each iteration step, the linkage between nodes is stored, and as a final step, the tree is built by linking nodes.

To apply the algorithm, an estimated tree-length is calculated for each possible topology, and the estimated tree-length is defined as

$$D_{ij} = d_{ij} - (r_i + r_j), \tag{4.4}$$

where $i, j = 1 \dots N, i < j$, and

$$r_i = \frac{1}{N-2}\sum_{k=1}^{N} d_{ik}.$$

The nodes x^i and x^j with minimal D_{ij} are connected by a parent node, x^{N+1}. Then the distances between OTUs x^i and x^j and internal node x^{N+1} are computed using

$$\begin{aligned} d_{N+1\,i} &= \frac{1}{2}(d_{ij} + r_i - r_j), \\ d_{N+1\,j} &= \frac{1}{2}(d_{ij} + r_j - r_i), \end{aligned}$$

and the distances between the new node, x^{N+1} and other nodes x^ℓ (where $\ell \neq i, j$) are computed using

$$d_{N+1\,\ell} = \frac{1}{2}(d_{i\ell} + d_{j\ell} - d_{ij}).$$

At the end of the first iteration, the new OTU list is $\{x^\ell, x^{N+1} : \ell \neq i, j\}$. This procedure is repeated until three OTUs are left. The distances between the final three OTUs are calculated using Equation (4.3), and the tree is built by linking each iteration step.

Example 4.3.2. *Assume that we have the following distance matrix for five different sequences:*

$$\begin{array}{c|ccccc} & x^1 & x^2 & x^3 & x^4 & x^5 \\ \hline x^1 & 0 & 4 & 7 & 5 & 6 \\ x^2 & 4 & 0 & 5 & 3 & 4 \\ x^3 & 7 & 5 & 0 & 5 & 4 \\ x^4 & 5 & 3 & 5 & 0 & 4 \\ x^5 & 6 & 4 & 4 & 4 & 0 \end{array}.$$

This distance matrix satisfies the four-point condition, so we can apply the neighbour-joining algorithm to build the tree. First, $r_i, i \in \{1, \ldots, N = 5\}$ are calculated as

$$r_1 = \frac{22}{3}, \quad r_2 = \frac{16}{3}, \quad r_3 = 7, \quad r_4 = \frac{17}{3}, \quad r_5 = 6.$$

Then the following matrix D (estimated tree length) is obtained using Equation (4.4)*:*

$$\begin{array}{c|ccccc} D & x^1 & x^2 & x^3 & x^4 & x^5 \\ \hline x^1 & & -26/3 & -22/3 & -8 & -22/3 \\ x^2 & & & -22/3 & -8 & -22/3 \\ x^3 & & & & -23/3 & -9 \\ x^4 & & & & & -23/3 \end{array}.$$

From the matrix D, the minimum value is $D_{35} = -9$, so OTUs x^3 and x^5 are replaced by the new OTU x^6. The OTU x^6 is located by the following distances from OTUs x^3 and x^5 as

$$\begin{aligned} d_{63} &= \frac{1}{2}(d_{35} + r_3 - r_5) = \frac{5}{2}, \\ d_{65} &= \frac{1}{2}(d_{35} + r_5 - r_3) = \frac{3}{2}. \end{aligned}$$

The distance between x^6 and other nodes, x^1, x^2, x^4 are calculated. These distances are

$$\begin{aligned} d_{61} &= \frac{1}{2}(d_{31} + d_{51} - d_{35}) = \frac{9}{2}, \\ d_{62} &= \frac{1}{2}(d_{32} + d_{52} - d_{35}) = \frac{5}{2}, \\ d_{64} &= \frac{1}{2}(d_{34} + d_{54} - d_{35}) = \frac{5}{2}, \end{aligned}$$

so the new distance matrix is for OTUs x^1, x^2, x^4, x^6 is

$$\begin{array}{c|cccc} & x^1 & x^2 & x^4 & x^6 \\ \hline x^1 & 0 & 4 & 5 & 9/2 \\ x^2 & 4 & 0 & 3 & 5/2 \\ x^4 & 5 & 3 & 0 & 5/2 \\ x^6 & 9/2 & 5/2 & 5/2 & 0 \end{array}.$$

The process is repeated using the new distance matrix, so

$$r_1 = \frac{27}{4}, \quad r_2 = \frac{19}{4}, \quad r_4 = \frac{21}{4}, \quad r_6 = \frac{19}{4},$$

and

$$\begin{array}{c|cccc} D & x^1 & x^2 & x^4 & x^6 \\ \hline x^1 & & -15/2 & -7 & -7 \\ x^2 & & & -7 & -7 \\ x^4 & & & & -15/2 \end{array}.$$

The minimum $D = -15/2$, so we can group either x^1 and x^2 or x^4 and x^6. We group x^4 and x^6, and the new OTU x^7 is located at the distance $3/2$ from x^4 and at the distance 1 from x^6. We also calculate the distance d_{71} and d_{72}, and we set the distance matrix for OTUs x^1, x^2, x^7 as

$$\begin{array}{c|ccc} & x^1 & x^2 & x^7 \\ \hline x^1 & 0 & 4 & 7/2 \\ x^2 & 4 & 0 & 3/2 \\ x^7 & 7/2 & 3/2 & 0 \end{array}.$$

As a final step, a new OTU, x^8 is set, and distances d_{81}, d_{82}, d_{87} are calculated using Equation (4.3)*. Thus, distances d_{81}, d_{82} and d_{87} are found as* 3, 1 *and* 1/2*, respectively, and the reconstructed tree is given in Figure 4.4.*

The generated tree clearly shows that the four-point condition is satisfied by the set of distances $\{d_{ij}\}$. However, in reality, neither the four-point condition nor the triangular equality is easy to verify. Instead of using these set of distances, a “pseudodistance”

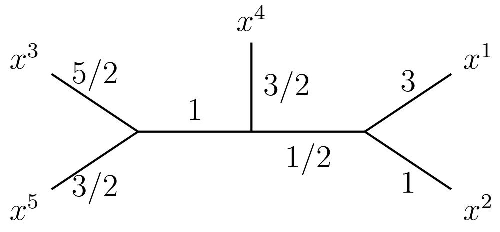


Figure 4.4: Phylogenetic tree reconstruction using neighbour-joining algorithm. The algorithm is based on the distance matrix in Example 4.3.2.

function is used to reconstruct the tree. A “pseudodistance” function can be defined using the assumptions on distances, given in Equation (4.1). If all assumptions hold except the triangle inequality, this distance function is a “pseudodistance” function. Hence, the neighbour-joining algorithm reconstructs the tree using a “pseudodistance” function. However, if a tree is constructed using a “pseudodistance” matrix which does not satisfy the four-point condition, some problems can be observed such as obtaining more than one tree, having negative branch lengths, or obtaining a different distance matrix from the one in the beginning.

Another point to be raised here is how to construct a rooted tree using the neighbour-joining algorithm. Note that the four-point condition is satisfied directly if any distance function is ultrametric. When the neighbour-joining algorithm is applied to an ultrametric distance function, after the tree is constructed, the root of the tree can be placed such that the total length of the branches from the root to each leaf should be equal.

The next phylogenetic reconstruction method is a probabilistic method which is based on evolutionary models, so we continue with a brief summary of some main evolutionary models in the following section before we discuss the probabilistic method in Section 4.5.

4.4 Evolutionary Models

In this section, evolutionary models are described briefly. Details of these models can be found in Isaev (2006) and Durbin (1998). Evolutionary models are required to understand the substitution process in DNA, RNA and amino acid sequences. If DNA sequences are considered, almost all evolutionary models assume that nucleotide sites are independent. DNA sequences are built by four different nucleotides: adenine (A), thymine (T), guanine (G) and cytosine (C), and we illustrate the nucleotide space as $\mathcal{Q} = \{A, C, G, T\}$. Evolutionary models are constructed via a transition probability matrix. These matrices denote the probability of state change in time, so the structure of the transition probability matrices is

$$P(t) = \begin{bmatrix} p_{AA}(t) & p_{AC}(t) & p_{AG}(t) & p_{AT}(t) \\ p_{CA}(t) & p_{CC}(t) & p_{CG}(t) & p_{CT}(t) \\ p_{GA}(t) & p_{GC}(t) & p_{GG}(t) & p_{GT}(t) \\ p_{TA}(t) & p_{TC}(t) & p_{TG}(t) & p_{TT}(t) \end{bmatrix}_{4\times 4}, \tag{4.5}$$

where $p_{ij} > 0$, $i, j \in \mathcal{Q}$, and $\sum_{j\in\mathcal{Q}} p_{ij}(t) = 1$ for each $i \in \mathcal{Q}$. Any $p_{ij}(t)$ from $P(t)$ denotes the probability of state change from site i to j in time t.

One of the assumptions on evolutionary models is that when a site has nucleotide i at time t, the probability of change from nucleotide i to nucleotide j at time $t + \tau$ only depends on i, j and τ ($\tau \geq 0$), so it can be written as

$$p_{ij}(t+\tau) = \sum_{k\in\mathcal{Q}} p_{ik}(t) p_{kj}(\tau),$$

where $i, j \in \mathcal{Q}$, and this can be written in matrix notation as

$$P(t+\tau) = P(t)P(\tau),$$

and assume that $P(0) = I_4$, and I_4 is the identity matrix. These assumptions lead us the following theorem.

Theorem 4.4.1. *(Isaev, 2006, p. 124) The transition probabilities matrix, $P(t)$ can be defined as*

$$P(t) = \exp(tQ), \tag{4.6}$$

where Q is a 4×4 matrix and has a specific form for each different evolutionary model.

Isaev (2006) defined the exponential of a matrix as

$$\exp(H) = I_n + H + \frac{H^2}{2!} + \frac{H^3}{3!} + \cdots = \sum_{k=0}^{\infty} \frac{H^k}{k!},$$

where H is a $n \times n$ matrix. The proof of Theorem 4.4.1 can be found in Isaev (2006).

A number of different evolutionary models have been proposed each with its own matrix Q. We now describe the main ones.

4.4.1 The Jukes-Cantor (JC) model

One of the earliest evolutionary models was introduced by Jukes & Cantor (1969), and this model is given as

$$Q = \begin{bmatrix} -3\alpha/4 & \alpha/4 & \alpha/4 & \alpha/4 \\ \alpha/4 & -3\alpha/4 & \alpha/4 & \alpha/4 \\ \alpha/4 & \alpha/4 & -3\alpha/4 & \alpha/4 \\ \alpha/4 & \alpha/4 & \alpha/4 & -3\alpha/4 \end{bmatrix},$$

where $\alpha > 0$ is the evolutionary rate. Since Q is known, $P(t)$ can be found via Equation (4.6). Thus,

$$\begin{aligned} p_{ii}(t) &= \frac{1}{4} + \frac{3}{4}\exp(-t\alpha), \quad i \in \mathcal{Q} = \{A, C, G, T\}, \\ p_{ij}(t) &= \frac{1}{4} - \frac{1}{4}\exp(-t\alpha), \quad i \neq j,\ i, j \in \mathcal{Q} = \{A, C, G, T\}. \end{aligned}$$

JC model assumes that the rate of transitions and transversions are equal to each other, where transitions stand for the nucleotide substitutions from purine to purine (A and G) or pyrimidine to pyrimidine (C and T), and transversions are the nucleotide substitutions between purine and pyrimidine. This model also assumes that the nucleotide equilibrium frequencies equal to each other ($\pi_A = \pi_C = \pi_G = \pi_T = 1/4$). However, in reality, transitions occur more than transversions.

4.4.2 The Kimura model

The Kimura model (Kimura, 1980) generalizes the JC model by giving different rates to transitions and transversions. This model is given as

$$Q = \begin{bmatrix} -(2\beta+\alpha)/4 & \beta/4 & \alpha/4 & \beta/4 \\ \beta/4 & -(2\beta+\alpha)/4 & \beta/4 & \alpha/4 \\ \alpha/4 & \beta/4 & -(2\beta+\alpha)/4 & \beta/4 \\ \beta/4 & \alpha/4 & \beta/4 & -(2\beta+\alpha)/4 \end{bmatrix},$$

where $\alpha > 0$ is the evolutionary rate, and $\beta > 0$. Using Equation (4.6), $P(t)$ is given by

$$p_{ii}(t) = \frac{1}{4} + \frac{1}{4}\exp(-t\beta) + \frac{1}{2}\exp\left(-t\frac{(\beta+\alpha)}{2}\right), \text{ where } i \in \mathcal{Q} = \{A, C, G, T\},$$

$$\begin{aligned} &p_{AC}(t) = p_{CA}(t) = p_{AT}(t) = p_{TA}(t) = p_{CG}(t) = p_{GC}(t) \\ &= p_{GT}(t) = p_{TG}(t) = \frac{1}{4} - \frac{1}{4}\exp(-t\beta), \\ &p_{AG}(t) = p_{GA}(t) = p_{CT}(t) = p_{TC}(t) = \frac{1}{4} + \frac{1}{4}\exp(-t\beta) - \frac{1}{2}\exp\left(-t\frac{(\beta+\alpha)}{2}\right). \end{aligned}$$

Even though Kimura model is a popular model, it is not realistic either because it shares one of the assumption done by JC model: equilibrium frequencies are equal ($\pi_A = \pi_C = \pi_G = \pi_T = 1/4$).

4.4.3 Felsenstein model

Felsenstein (1981) introduced another model which is also a general version of the JC model. This model is

$$Q = \begin{bmatrix} -\alpha(\pi_C + \pi_G + \pi_T) & \alpha\pi_C & \alpha\pi_G & \alpha\pi_T \\ \alpha\pi_A & -\alpha(\pi_A + \pi_G + \pi_T) & \alpha\pi_G & \alpha\pi_T \\ \alpha\pi_A & \alpha\pi_C & -\alpha(\pi_A + \pi_C + \pi_T) & \alpha\pi_T \\ \alpha\pi_A & \alpha\pi_C & \alpha\pi_G & -\alpha(\pi_A + \pi_C + \pi_G) \end{bmatrix},$$

where $\alpha > 0$ is the evolutionary rate, and $\pi_i > 0$, $i \in \mathcal{Q} = \{A, C, G, T\}$ are parameters with $\sum_{k\in\mathcal{Q}} \pi_k = 1$. Using Equation (4.6), $P(t)$ is found to be

$$\begin{aligned} p_{ii}(t) &= \pi_i + (1-\pi_i)\exp(-t\alpha), && \text{for all } i, \text{ where } i \in \mathcal{Q} = \{A, C, G, T\}, \\ p_{ij}(t) &= \pi_j - \exp(-t\alpha)\pi_j, && \text{for all } i \neq j. \end{aligned}$$

While Kimura model brings a solution to equality assumption on transition and transversion rates, Felsenstein model suggests a solution to equality assumption on equilibrium frequencies. Thus, Felsenstein model is not realistic either.

4.4.4 The Hasegawa-Kishino-Yano (HKY) model

Hasegawa et al. (1985) generalized the Felsenstein model. The HKY model defines different transitions and transversions rates along with different equilibrium rates, so the HKY model is a generalized version of all other three models (JC, Kimura and Felsenstein) which we described earlier. The HKY model is

$$Q = \begin{bmatrix} -(\beta\pi_C + \alpha\pi_G + \beta\pi_T) & \beta\pi_C & \alpha\pi_G & \beta\pi_T \\ \beta\pi_A & -(\beta\pi_A + \alpha\pi_G + \beta\pi_T) & \beta\pi_G & \alpha\pi_T \\ \alpha\pi_A & \beta\pi_C & -(\alpha\pi_A + \beta\pi_C + \beta\pi_T) & \beta\pi_T \\ \beta\pi_A & \alpha\pi_C & \beta\pi_G & -(\beta\pi_A + \alpha\pi_C + \beta\pi_G) \end{bmatrix},$$

where $\alpha > 0$ is the evolutionary rate, $\beta > 0$, and $\pi_i > 0$, $i \in \mathcal{Q} = \{A, C, G, T\}$ are parameters with $\sum_{k\in\mathcal{Q}} \pi_k = 1$. Using Equation (4.6), $P(t)$ can be found, but this probability matrix is not as simple as others to write down. Thus, we do not include it here. The full version of the matrix can be found in Hasegawa et al. (1985).

4.5 Probabilistic methods

In Section 4.3, parsimony and distance methods were described to construct the phylogenetic tree using reduced multiple alignment of DNA sequences. In this section, we present maximum likelihood method (probabilistic method), based on evolutionary models described in Section 4.4, to build phylogenetic trees. The earliest OTUs and the reduced multiple aligned sequences are represented by $M = \{x^1, \ldots, x^N\}$ and $D = \{\hat{x}^1, \ldots, \hat{x}^N\}$, respectively. To start the method, one of the evolutionary methods is chosen, and this method assumes that the molecular clock assumption holds for DNA sequences, D, and the time tree is constructed, where branch lengths are time intervals; evolution of DNA sequences occurred via substitutions, so there were no deletions or insertions, and evolution in each site is independent and identical to other sites; the substitution process is described via a selected evolutionary model; and the parameters α and β in evolutionary models can vary from branch to branch.

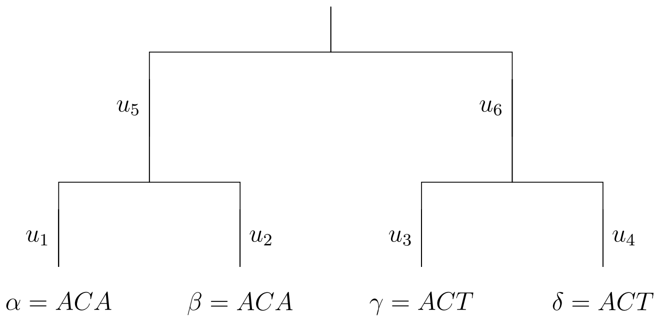


Figure 4.5: Constructed tree via probabilistic methods. Rooted tree for $D = (\hat{x}^1, \ldots, \hat{x}^4) = (\alpha, \beta, \gamma, \delta)$. $u_.$*: time interval between two nodes.*

To build the time tree via maximum likelihood method, the likelihood function of the aligned data gives the topology of the molecular clock tree, T. For each possible molecular clock tree, the likelihood function, $L(D|T)$ is calculated, and the optimal tree is the one having the maximal likelihood.

Branch lengths of the time tree can be computed using maximum likelihood method. We assume that the time tree satisfies the molecular clock assumption, but evolution can be faster in some branches than others. Since there is no information on evolutionary speed, it is not easy to calculate branch lengths. To be able to find branch lengths, time intervals can be scaled by evolutionary rates. Hence, distance matrices between aligned OTUs can be set via probabilistic methods.

Setting the likelihood function and distance matrices are explained using the toy data given in Table 4.1a and its reduced form as given in Table 4.1c. The rooted tree, T for this data is given in Figure 4.5. The likelihood function, $L(D|T)$ can be written via site specific likelihood functions, $L_i(D|T)$, where $i \in \{1, \ldots, n\}$, and n is the length of the reduced alignment of sequences. Thus,

$$\begin{aligned}
L_1(D|T) &= \sum_{i,j,k \in \mathcal{Q}} \varphi_i p_{ij}(u_5) p_{ik}(u_6) p_{jA}(u_1) p_{jA}(u_2) p_{kA}(u_3) p_{kA}(u_4),\\
L_2(D|T) &= \sum_{i,j,k \in \mathcal{Q}} \varphi_i p_{ij}(u_5) p_{ik}(u_6) p_{jC}(u_1) p_{jC}(u_2) p_{kC}(u_3) p_{kC}(u_4),\\
L_3(D|T) &= \sum_{i,j,k \in \mathcal{Q}} \varphi_i p_{ij}(u_5) p_{ik}(u_6) p_{jA}(u_1) p_{jA}(u_2) p_{kT}(u_3) p_{kT}(u_4),
\end{aligned}$$

where φ denotes the stationary distribution of chosen evolutionary model, u is for the time interval, and $L(D|T)$ is

$$L(D|T) = \prod_{i \in \{1, \ldots, n\}} L_i(D|T).$$

Since the likelihood function is known, distance between OTUs, $d(\hat{x}^\ell, \hat{x}^j)$, can be calculated by maximizing $L(D|T)$, where $\ell, j \in \{1, \ldots, N\}$. Derivation of distances can be found in Isaev (2006).

4.6 Discussion

To construct phylogenetic trees, one of the parsimony, distance and probabilistic methods can be used. Each one has some advantages or disadvantages. Parsimony methods are computationally efficient since they do not need to find the branch lengths, but they are not model based methods. Thus, there are some concerns on not having assumptions behind them. Distance methods are also computationally efficient methods, and they can even deal with large data sets easily. They are model based methods, so assumptions behind these methods are clear. The probabilistic methods are also based on models, so there is no confusion on their assumptions. However, they are not computationally efficient because of the process of finding branch lengths. Since distance methods are computationally efficient and have clear assumptions, we will use one of the distance based methods later in Chapter 7.

Chapter 5

Automatic cluster detection by lifting

5.1 Introduction

Using clustering methods, related objects are grouped in the same cluster. One of the well known clustering algorithms is called hierarchical clustering. The aim of introducing hierarchical clustering is to consider how the lifting algorithm can be applied to a tree built by the hierarchical clustering algorithm. Hierarchical clustering is a common methodology in statistics, so more details can be found in, for example, Mardia et al. (1979) and Manly (2004).

One of the open questions in hierarchical clustering is how many clusters exist, or where we will "cut the tree". Even though many cluster validity indices are proposed in the literature, this topic still catches the interest of researchers. There are also some studies offering a comparison of these indices; for example, Arbelaitz et al. (2013) recently compared 30 different cluster validity indices. All the available indices find the number of clusters, but we would like to explore if the number of clusers can be decided automatically. This automatic decision allows us to examine where exactly clustering happens in a dendrogram. To reach this goal, a new method is proposed which is created using the lifting algorithm for tree-structured data introduced by Jansen et al. (2009).

This chapter starts with a brief summary of hierarchical clustering in Section 5.2, then some of the recent cluster validity indices or the ones mostly referenced in the literature are described in Section 5.3. After that, how we can apply lifting method to a tree produced by hierarchical clustering is discussed in Section 5.4. We next compare the performance of our lifting algorithm and the partitioning found by other cluster validity indices using four different simulated data structures and a real data set in Sections 5.5 and 5.6, respectively. Finally, we discuss our findings briefly in Section 5.7.

index	component	1st dimension	2nd dimension
1	1	-1.033	1.085
2	1	-0.963	0.990
3	1	-0.645	0.587
4	2	1.010	-0.808
5	2	1.160	-1.180
6	2	0.471	-0.674

Table 5.1: Tree structured toy data.

5.2 Agglomerative hierarchical clustering

There are two different hierarchical clustering methods: agglomerative and divisive hierarchical methods. In this research, our main interest is the agglomerative hierarchical method. The algorithm starts with the distance matrix and each object being in a separate cluster. Then in each agglomeration step, the closest clusters are merged.

In hierarchical clustering, there are three main possible linkage methods: closest-neighbour (single linkage), furthest-neighbour (complete linkage) and average linkage. When two clusters are merged, the distance matrix is updated in terms of the choice of linkage method. Thus, the distances between the new cluster and the others are found by choosing the smallest distance between clusters when closest-neighbour linkage method is used. For example, assume that there is a data set, $x_1, \ldots, x_5$, and the distance between x_1 and x_3, d_{13} is the smallest. Thus, x_1 and x_3 are merged in the first agglomeration step, and we denote this new cluster with x_6. The distance between x_6 and x_2, d_{62}, is $\min\{d_{12}, d_{32}\}$, and distances between the new cluster and other data points, d_{64} and d_{65}, are updated in the same way. If the furthest-neighbour linkage or average linkage is used, the distance between x_6 and x_2, d_{62}, is $\max\{d_{12}, d_{32}\}$ or $\frac{1}{2}\{d_{12} + d_{32}\}$, respectively.

Toy Data: For illustrative purposes, a tree structured (multidimensional) toy data set is created and shown in Table 5.1. Two-component normally distributed data set in $\mathbb{R}^2$ is generated, and each component includes three observations.

We build the tree for the toy data hierarchically using Euclidean distances and complete linkage. The Euclidean distance matrix is computed and is given in Table 5.2 (it can be easily computed using the **dist()** function in **R**), and objects can be clustered hierarchically using the **hclust()** function in the **stats** package (R Core Team, 2017) in **R**. The corresponding dendrogram is given in Figure 5.1. Blue labels for the internal nodes represent the agglomeration order, and internal nodes are labelled starting from $n + 1$, where n is the number of objects in the data, so $n = 6$, and the possible clustering schemes for this dendrogram are given in Table 5.3.

After building a dendrogram of a data set, it is not always easy to know where to

	1	2	3	4	5
2	0.118				
3	0.632	0.514			
4	2.785	2.669	2.164		
5	3.153	3.036	2.526	0.401	
6	2.314	2.197	1.684	0.555	0.855

Table 5.2: Euclidean distance matrix for the toy data in Table 5.1.

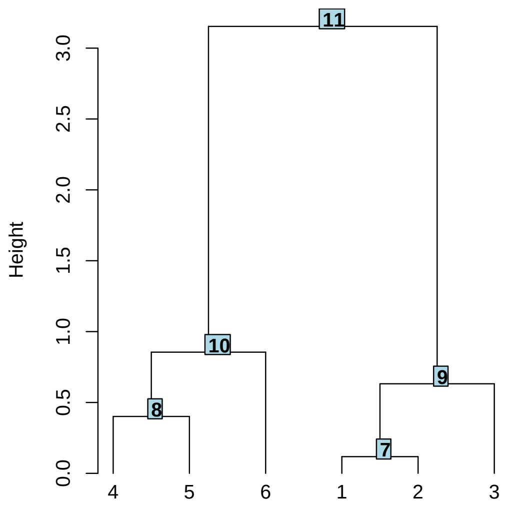


Figure 5.1: The dendrogram of the toy data in Table 5.1. Nodes are labelled by the agglomeration order starting from $n+1$.

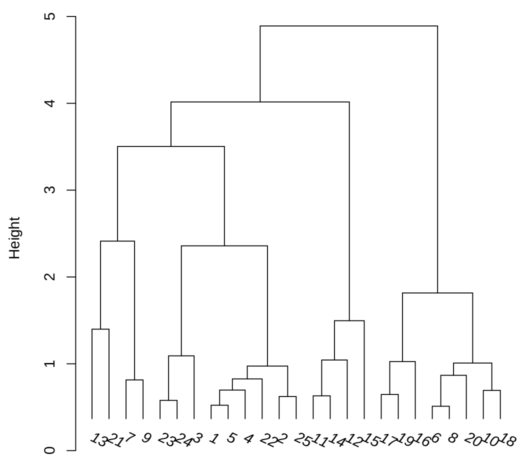


Figure 5.2: An example of a complicated dendrogram.

k	D_j where $j \in \{1, \ldots, k\}$
1	{1,2,3,4,5,6}
2	{1,2,3}, {4,5,6}
3	{1,2,3}, {4,5}, {6}
4	{1,2}, {3}, {4,5}, {6}
5	{1,2}, {3}, {4}, {5}, {6}
6	{1}, {2}, {3}, {4}, {5}, {6}.

Table 5.3: Clustering scheme for the toy data in Table 5.1.

cut the tree. For example, we can say where to cut the tree just seeing the dendrogram in Figure 5.1, but if we have a more complicated dendrogram such as the one given in Figure 5.2, it will not be easy to find how many clusters we have. Thus, we need some methods which tell us which partition represents the data better. To address this question, some methods called internal cluster validity indices are proposed in the literature. Since each of these indices returns one of the possible partitioning, we may also need to evaluate how well this partition captures true clusters when we have a labelled data. Thus, this evaluation can be done using one of the external cluster validity scores from the literature. In Section 5.3, we discuss both internal indices and external scores in detail.

5.3 Cluster validity indices

Cluster validity indices can be divided into two categories: internal indices and external scores. Internal indices are used to find the best partitioning after applying the clustering algorithm. Then external scores are used to measure how well the true components are captured by clustering if the true partition of the data is known.

In the literature, many internal and external scores are available, so we pick five different internal indices from the recently developed and the most commonly referenced indices: Calinski and Harabasz index (Calinski & Harabasz, 1974), Hartigan index (Hartigan, 1975), Silhouette statistic (Rousseeuw, 1987), Krzanowski and Lai index (Krzanowski & Lai, 1988) and Gap statistic (Tibshirani et al., 2001). All these internal indices are available in the **NbClust** package (Charrad et al., 2014) in **R**. Even though we are interested in hierarchically built trees, to see the performance of a different clustering method, we include the model-based clustering method (Fraley & Raftery, 2002) which uses mixture of normal distributions in our study. It is available in the **mclust** package (Fraley et al., 2012) in **R**. We compare the performance of these internal indices and model-based clustering in terms of six external scores: Wallace indices (Wallace, 1983) and the Fowlkes and Mallows index (Fowlkes & Mallows, 1983) which are available in the **profdpm** package (Shotwell, 2013), the adjusted Rand index (Hubert & Arabie, 1985) which is available in the **mclust** package, purity index (Rendón et al., 2011) which is available in the **IntNMF** package (Chalise et al., 2016) and adjusted variation information (Vinh et al., 2010). These internal indices and external scores are discussed in detail later in this section.

5.3.1 Internal indices

Internal index calculations are based on between-cluster sum of squares (BSS) and within-cluster sum of squares (WSS), so we need to define BSS and WSS with some notation which are used in the discussion of various indices. We define

k	: number of clusters,
n	: number of objects,
p	: number of variables,
D_j	: cluster j which includes indices of data points in it, where $j \in \{1, \ldots, k\}$,
x_i	: ith data point in data x, $x \in \mathbb{R}^p$ and $i \in \{1, \ldots, n\}$,

$$\begin{aligned}
n_j &: \text{ number of elements in cluster } D_j,\\
d(x_i, x_\ell) &: \text{ the distance between } i\text{th and } \ell\text{th data points, } i, \ell \in \{1, \dots, n\},\\
c_j &: \text{ the centroid of data points in the cluster } D_j\text{, so}\\
&\quad c_j = n_j^{-1} \textstyle\sum_{i \in D_j} x_i,\\
\bar{x} &: \text{ mean of all elements,}\\
WSS(k) &= \textstyle\sum_{j=1}^{k} \sum_{i \in D_j} (x_i - c_j)(x_i - c_j)^T,\\
BSS(k) &= \textstyle\sum_{j=1}^{k} n_j (c_j - \bar{x})(c_j - \bar{x})^T.
\end{aligned}$$

We refer internal cluster validity indices as CVIs in the later part of this chapter and the following chapters.

Calinski and Harabasz index (CH)

The Calinski and Harabasz index (CH) was proposed by Calinski & Harabasz (1974) and is defined as

$$\mathrm{CH}(k) = \frac{\mathrm{tr}(BSS(k))/(k-1)}{\mathrm{tr}(WSS(k))/(n-k)}, \tag{5.1}$$

where $k > 1$.

If the similar objects are clustered together, $WSS(k)$ will be small, and $BSS(k)$ will be high. If we scale $BSS(k)$ and $WSS(k)$ in terms of their degrees of freedom, we can take the proportion of BSS and WSS, and $\mathrm{CH}(k)$ takes its maximum value when large distances occur between clusters. Thus, the optimal number of clusters is the k which maximizes $\mathrm{CH}(k)$.

We illustrate how we can decide the optimal number of clusters using the CH index for the toy data, given in Table 5.1. So for each possible clustering scheme in Table 5.3, we calculate the CH index, then the clustering scheme with the maximum CH index is the "best" partitioning.

The CH index calculation is shown in detail for $k = 2$:

$$\begin{aligned}
n &= 6,\\
D_1 &= \{1, 2, 3\}, \quad D_2 = \{4, 5, 6\},\\
n_1 &= 3, \quad n_2 = 3,\\
c_1 &= (-0.881, 0.887), \quad c_2 = (0.881, -0.887),\\
\bar{x} &= (0, 0),\\
WSS(k=2) &= \textstyle\sum_{j=1}^{2} \sum_{i \in D_j} (x_i - c_j)(x_i - c_j)^T\\
&= \begin{bmatrix} 0.348 & -0.269 \\ -0.269 & 0.278 \end{bmatrix},\\
\mathrm{tr}(WSS(k=2)) &= 0.626,
\end{aligned}$$

$$\begin{aligned}
BSS(k=2) &= \textstyle\sum_{j=1}^{2} n_j(c_j-\bar{x})(c_j-\bar{x})^T \\
&= \begin{bmatrix} 4.652 & -4.687 \\ -4.687 & 4.722 \end{bmatrix}, \\
\mathrm{tr}(BSS(k=2)) &= 9.374, \\
\mathrm{CH}(k=2) &= \frac{\mathrm{tr}(BSS(k=2))/(2-1)}{\mathrm{tr}(WSS(k=2))/(6-2)} = 59.918.
\end{aligned}$$

For each possible clustering scheme, $k \in \{2,\dots,5\}$, the CH index is calculated and found to be $\{59.918, 47.472, 75.519, 359.549\}$, respectively. Hence, the maximum CH index is computed for $k = 5$. The optimal number of clusters for the toy data is five where one of the clusters includes $\{x_1, x_2\}$, and all the other data points are clustered separately.

Hartigan index (H)

The Hartigan index (H) was proposed by Hartigan (1975). The index is defined as

$$\mathrm{H}(k) = \left\{ \frac{\mathrm{tr}(WSS(k))}{\mathrm{tr}(WSS(k+1))} - 1 \right\} \times (n-k-1), \tag{5.2}$$

where $k \in \{2,\dots,(n-2)\}$.

If objects in cluster k are similar, $WSS(k)$ will be small. Thus, we need to start from one cluster and add more clusters if $\mathrm{H}(k+1)$ is large enough. This occurs if and only if $WSS(k+1)$ is small enough. Hartigan (1975) suggested that the optimal number of clusters is the smallest k which makes $\mathrm{H}(k) \le 10$, but Milligan & Cooper (1985) proposed another stopping rule which increases the performance of the index: the optimal number of clusters is the k which maximizes $\mathrm{H}(k)$.

Using the toy data in Table 5.1, we compute the H index for each possible clustering scheme, given in Table 5.3. We already calculated $WSS(k=2)$ in the illustration of the CH index. To find $\mathrm{H}(k=2)$, we need to find $WSS(k=3)$ using the clustering scheme for $k=3$. Thus,

$$\begin{aligned}
D_1 &= \{1,2,3\}, \quad D_2 = \{4,5\}, \quad D_3 = \{6\}, \\
n_1 &= 3, \quad n_2 = 2, \quad n_3 = 1, \\
c_1 &= (-0.881, 0.887), \quad c_2 = (1.085, -0.994), \quad c_3 = (0.471, -0.674), \\
\mathrm{tr}(WSS(k=3)) &= \mathrm{tr}\left(\textstyle\sum_{j=1}^{3}\sum_{i\in D_j}(x_i-c_j)(x_i-c_j)^T\right) = 0.306, \\
\mathrm{H}(k=2) &= \left\{ \frac{\mathrm{tr}(WSS(k=2))}{\mathrm{tr}(WSS(k=3))} - 1 \right\} \times (6-2-1) \\
&= \left\{ \frac{0.626}{0.306} - 1 \right\} \times 3 = 3.129.
\end{aligned}$$

The same procedure is repeated for other clustering schemes, $k \in \{2,\dots,4\}$, so H indices can be found as $\{3.129, 5.001, 11.594\}$, respectively. The best partition found by the H index is for $k = 4$ since the k which maximizes $\mathrm{H}(k)$ is four.

Krzanowski and Lai index (KL)

The Kranowski and Lai index (KL) (Krzanowski & Lai, 1988) is defined as

$$\mathrm{KL}(k) = \left| \frac{\mathrm{DIFF}(k)}{\mathrm{DIFF}(k+1)} \right|, \tag{5.3}$$

where $k \geq 2$, and

$$\mathrm{DIFF}(k) = (k-1)^{2/p}\,\mathrm{tr}(WSS(k-1)) - k^{2/p}\,\mathrm{tr}(WSS(k)).$$

The KL index is similar to the CH index. Krzanowski & Lai (1988) discussed that $WSS(k)$ will be reduced by $k^{2/p}$ if x_is are independently uniformly distributed. If k^* is the optimal number of clusters, $\mathrm{DIFF}(k)$ will be positive large numbers for $k < k^*$, and $\mathrm{DIFF}(k)$ will be smaller (it can even take negative values) for $k > k^*$. Thus, the optimal number of clusters, k, is the one which maximizes $\mathrm{KL}(k)$.

Using the toy data in Table 5.1, we show how we can find the KL index for $k = 2$. To compute the KL index, we need to calculate $WSS(k = 1)$, $WSS(k = 2)$ and $WSS(k = 3)$. Since the last two WSS are calculated in previous indices, we only need to compute the $WSS(k = 1)$, so

$$\begin{aligned}
p &= 2,\\
D_1 &= \{1, 2, 3, 4, 5, 6\},\\
c_1 &= (0, 0),\\
\mathrm{tr}(WSS(k=1)) &= \mathrm{tr}\left(\textstyle\sum_{j=1}^{1} \sum_{i \in D_j} (x_i - c_j)(x_i - c_j)^T\right) = 10.000,\\
\mathrm{DIFF}(k=2) &= \mathrm{tr}(WSS(1)) - 2(\mathrm{tr}(WSS(2))) = 8.748,\\
\mathrm{DIFF}(k=3) &= 2(\mathrm{tr}(WSS(2))) - 3(\mathrm{tr}(WSS(3))) = 0.333,\\
\mathrm{KL}(k=2) &= \left| \frac{8.748}{-0.919} \right| = 26.294.
\end{aligned}$$

Similarly, we compute the KL index for each clustering scheme, $k \in \{2, \ldots, 5\}$, and find as $\{26.294, 0.585, 1.804, 9.075\}$, respectively. The maximum KL index is found for $k = 2$, so the KL index finds the best partition of the toy data as two clusters.

Silhouette statistic

The silhouette statistic (Rousseeuw, 1987) is defined as

$$\mathrm{Sil}(k) = \frac{1}{n} \sum_{i=1}^{n} s(i), \tag{5.4}$$

where

$$\begin{aligned} s(i) &= \frac{b(i)-a(i)}{\max\{a(i),b(i)\}}, \\ a(i) &= \frac{1}{n_j-1}\sum\nolimits_{\ell\in D_j} \mathrm{d}(x_i,x_\ell), \\ b(i) &= \min\nolimits_{D_s\in D\setminus D_j}\left\{\frac{1}{n_s}\sum\nolimits_{\ell\in D_s}\mathrm{d}(x_i,x_\ell)\right\}. \end{aligned}$$

Here, $a(i)$ is the average distance between the ith point and the other points in its cluster, and $b(i)$ is the average distance between the ith point and the points from the nearest cluster. The silhouette statistic computes how well the ith object is clustered, so high $s(i)$ shows how strong the clustering is.

The silhouette statistic is defined in the range $[-1,1]$. The maximum index means the best partition, and also the silhouette statistic is not defined for $k=1$.

To be able to work on the toy data in Table 5.1, we need Euclidean distance matrix of the toy data in Table 5.2.

If $k=2$, and $i=1$,

$$\begin{aligned} a(1) &= \frac{1}{2}\sum\nolimits_{\ell\in D_1}\mathrm{d}(x_1,x_\ell) = \frac{1}{2}(0.118+0.632) = 0.375, \\ b(1) &= \min\nolimits_{D_2\in D\setminus D_1}\left\{\frac{1}{3}\sum\nolimits_{\ell\in D_2}\mathrm{d}(x_1,x_\ell)\right\} = \frac{1}{3}(2.785+3.153+2.314) = 2.751, \\ s(1) &= \frac{b(1)-a(1)}{\max\{a(1),b(1)\}} = \frac{2.751-0.375}{2.751} = 0.864. \end{aligned}$$

When we repeat these calculations for other data points in the data set, we find $a = (0.375, 0.316, 0.573, 0.478, 0.628, 0.705)$, $b = (2.751, 2.634, 2.124, 2.540, 2.905, 2.065)$ and $s = (0.864, 0.880, 0.730, 0.812, 0.784, 0.659)$. The silhouette statistic for two clusters is the average of $s(i)$, so $\mathrm{Sil}(k=2) = 0.788$. We need to find the silhouette statistic for each possible partition to investigate the optimal number of clusters, so for $k \in \{2,\ldots,5\}$, $\mathrm{Sil} = \{0.788, 0.694, 0.732, 0.931\}$, respectively. Hence, the silhouette statistic suggests that the best partition will be obtained if the toy data set is partitioned into five clusters ($\max(\mathrm{Sil}) = \mathrm{Sil}(k=5) = 0.931$).

Gap statistic

The Gap statistic was proposed by Tibshirani et al. (2001). It computes the amount of change between the expectation of the $WSS(k)$ of the data set coming from a suitable reference distribution and $WSS(k)$ of the original data set as k increases. The optimal number of clusters is the k which maximizes the Gap statistic, so the Gap statistic is defined as

$$\mathrm{Gap}(k) = \mathbb{E}\left\{\log(WSS_B^*(k))\right\} - \log(WSS(k)), \tag{5.5}$$

where $WSS_B^*(k)$ is the within sum of squares for B reference data sets coming from the reference distribution, and

$$\mathbb{E}\left\{\log(WSS_B^*(k))\right\} \approx \frac{1}{B} \sum_{b\in\{1,\ldots,B\}} \log(WSS_b^*(k)).$$

Tibshirani et al. (2001) suggested two methods for creating reference data sets. The simplest way is generating B data sets from a uniform distribution in the range of the observed data. The second way is generating data from a uniform distribution but over a box defined by the principal components of the data.

Computation of the Gap statistic can be summarized as follows:

- For each possible clustering scheme, $WSS(k)$ is computed, where $k \in \{1, \ldots, (n-1)\}$.
- We generate B reference data sets. Each data set is clustered, then for each possible partitioning, WSS_b^* is calculated where $b \in \{1, \ldots, B\}$. After that, we can calculate the Gap statistic, given in Equation (5.5).
- We need to also compute the standard deviation of $\log(WSS_b^*)$ which is defined as

$$sd_k = \left\{ \frac{1}{B} \sum_{b\in\{1,\ldots,B\}} \left[\log(WSS_b^*(k)) - \frac{1}{B} \sum_{b\in\{1,\ldots,B\}} \log(WSS_b^*(k)) \right]^2 \right\}^{1/2}.$$

- The optimal number of clusters is the smallest k such that

$$\text{Gap}(k) \geq \text{Gap}(k+1) - s_{k+1},$$

  where $s_k = sd_k\sqrt{1 + 1/B}$.

For the toy data in Table 5.1, the Gap statistic offers that the best partition of the data set is found when the data set is partitioned into two clusters. The Gap statistics for each possible cluster are $\text{Gap} = \{-0.414, 0.387, 0.296, 0.517, 1.003\}$, where $s_k = \{0.177, 0.131, 0.153, 0.174, 0.385\}$ and $k \in \{1, \ldots, 5\}$.

Discussion

All the indices reported in this section show high performances to find the best partitioning in clustering. However, their performance differs depending on the data structure and the clustering method. In the literature, there are some studies which compare different internal indices. One of the commonly referenced studies was carried out by Milligan & Cooper (1985). In their study, they reported that the CH index captured the true components better within 30 different indices. Arbelaitz et al. (2013) recently published a new comparison study, where they also compared 30 different indices including the CH index and the Sil statistic. They found that the Sil statistic performed better than others, and discussed that while the CH index performed better for the noiseless data and for the k-means clustering algorithm, the performance of the Sil statistic was better with the hierarchical clustering algorithm.

Another index is the H index. The stopping rule for the H index suggested by Hartigan (1975) includes an arbitrary number, and it decreases the performance of the index. In this study, we use the stopping rule offered by Milligan & Cooper (1985), so the estimated number of clusters (k) is the one which maximizes $\mathrm{H}(k)$. Another concern on the H index is we can not calculate the index for $\mathrm{H}(k = n - 1)$. There are also some concerns on the KL index. Krzanowski & Lai (1988) discussed that the efficiency of the index was lower for data sets having unequally sized true components or for data sets in high dimensions. The final index discussed in this section is the Gap statistic. Tibshirani et al. (2001) noted that the performance of the Gap statistic was high for well separated clusters, but if there is an overlap, or components are close to each other, its performance is poor. It generally ends up with one cluster. Sugar & James (2003) also found that its performance for exponentially distributed data was low.

By this point of the study, we have summarized some of the available internal indices in the literature which finds the number of clusters in hierarchically built trees, but we would like to also see the behaviour of another clustering method. Thus, we include the mixture model-based clustering to our study, and describe it in the following section.

5.3.2 Model-based clustering (Mclust)

Fraley & Raftery (2002) introduced mixture models in hierarchical clustering. They assume that the best partitioning is the one maximizing the classification likelihood, which is defined as

$$\mathcal{L}_{CL}\left(\theta_1, \ldots, \theta_k; \ell_1, \ldots, \ell_n\right) = \prod_{i=1}^{n} f_{\ell_i}\left(x_i | \theta_{\ell_i}\right),$$

where x_i is the given data point ($i \in \{1, \ldots, n\}$), k is the number of clusters, ℓ_i illustrates the cluster of x_i, θ_{ℓ_i} are the parameters for the cluster ℓ_i, and f_{ℓ_i} is the probability density function for the cluster ℓ_i. In general, any suitable distribution can be used to obtain the f_{ℓ_i}.

In Fraley & Raftery (2002), the density function, f_{ℓ_i} comes from multivariate normal (Gaussian) distribution, so the density function, $\phi(x_i; \mu_{\ell_i}, \Sigma_{\ell_i})$, is defined as

$$\phi(x_i; \mu_{\ell_i}, \Sigma_{\ell_i}) = \frac{\exp\{-1/2(x_i - \mu_{\ell_i})^T \Sigma_{\ell_i}^{-1}(x_i - \mu_{\ell_i})\}}{\sqrt{\det(2\pi\Sigma_{\ell_i})}},$$

where μ_{ℓ_i} and Σ_{ℓ_i} are the mean and variance of the cluster ℓ_i, respectively, and parameter ($\theta_{\ell_i} = \{\mu_{\ell_i}, \Sigma_{\ell_i}\}$) estimates are done using the EM algorithm for a fixed number

		Components, C_j				
		C_1	C_2	$\dots$	C_ℓ	$n_{i\cdot}$
Clusters, D_i	D_1	n_{11}	n_{12}	$\dots$	$n_{1\ell}$	$n_{1\cdot}$
	D_2	n_{21}	n_{22}	$\dots$	$n_{2\ell}$	$n_{2\cdot}$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	D_k	n_{k1}	n_{k2}	$\dots$	$n_{k\ell}$	$n_{k\cdot}$
	$n_{\cdot j}$	$n_{\cdot 1}$	$n_{\cdot 2}$	$\dots$	$n_{\cdot \ell}$	n

(a)

		Components, C_j					
		C_1	C_2	C_3	C_4	C_5	$n_{i\cdot}$
Clusters, D_i	D_1	2	0	3	0	0	5
	D_2	0	0	0	5	0	5
	D_3	0	5	2	0	0	7
	D_4	3	0	0	0	5	8
	$n_{\cdot j}$	5	5	5	5	5	25

(b)

Table 5.4: Structure of a contingency table. (a): general form of a contingency table. (b): a contingency table of a toy data set.

of clusters; details of the parameter estimates can be found in Fraley & Raftery (2002). Mclust method decides which pairs are merged in each agglomeration step by looking at the amount of increase in classification likelihood for each possible cluster. The clusters which make maximum increase on the classification likelihood are merged in that agglomeration stage. Then the number of clusters is found using the Bayesian information criteria (BIC). For each clustering pattern, $2, \dots, k$, BIC is calculated, and the partition with the maximum BIC is found as the representative clustering pattern of the data set by Mclust.

Model based hierarchical clustering is available in the **mclust** package (Fraley et al., 2012) in **R**. We cluster the toy data in Table 5.1 using the **mclust** package, and we find three clusters as $D_1 = \{x_1, x_2, x_3\}$, $D_2 = \{x_4, x_5\}$ and $D_3 = \{x_6\}$.

5.3.3 External scores

Internal indices and Mclust find the number of clusters, but they do not give any information about the performance of the partitioning found by the method of interest. We wish to measure how well the partitioning is estimated by these methods if we know the "truth" of the data. External scores measure how close the partition done by the clustering method is to the real partition, so external scores are partition similarity measures. Six different external scores from the literature are picked to compare the success of the partition found by internal indices.

External scores are calculated using contingency tables. A general table is given in Table 5.4a with the notation used when we introduce external scores. True groups are called components (C_j, where $j \in \{1, \dots, \ell\}$), and the labels after applying a clustering method are called clusters (D_i, where $i \in \{1, \dots, k\}$). Let n_{ij} be the number of objects classified to cluster D_i which are truly from component C_j, $n_{i\cdot}$ be the number of objects in cluster D_i, $n_{\cdot j}$ be the number of objects in component C_j, and n be the total number of objects. In addition, we provide the contingency table of a toy data set in Table 5.4b which is used to illustrate the calculation of each external score.

Fowlkes and Mallows index

Fowlkes & Mallows (1983) introduced an external score to check the performance of a clustering, and we call this score further this point as CC. It is defined as

$$\mathrm{CC} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{\ell}\binom{n_{ij}}{2}}{\sqrt{\sum_{i=1}^{k}\binom{n_{i\cdot}}{2}\sum_{j=1}^{\ell}\binom{n_{\cdot j}}{2}}}, \tag{5.6}$$

and later on Wallace (1983) proposed two other scores based on CC score. We call these scores CompCheck and ClustCheck

The idea is that if a component is divided into small clusters, clustering performance should be treated as high. Thus, we do not need to give any penalty for sub-clustering. Firstly, Wallace (1983) checked if objects which really belong together are clustered together:

$$\mathrm{CompCheck} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{\ell}\binom{n_{ij}}{2}}{\sum_{j=1}^{\ell}\binom{n_{\cdot j}}{2}}, \tag{5.7}$$

where $\mathrm{CompCheck} \in [0,1]$. If CompCheck is large, objects from the same components are clustered together. However, if all components are clustered together, CompCheck will also be equal to one. Thus, Wallace (1983) also considered if objects which are clustered together come from the same components, he defined

$$\mathrm{ClustCheck} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{\ell}\binom{n_{ij}}{2}}{\sum_{i=1}^{k}\binom{n_{i\cdot}}{2}}, \tag{5.8}$$

where $\mathrm{ClustCheck} \in [0,1]$. If ClustCheck is large, objects which belong together are placed in the same cluster.

For an illustrative purposes, we can use the toy data in Table 5.4b, so

$$\begin{aligned}\mathrm{CompCheck} &= 38/50 = \ 0.760,\\ \mathrm{ClustCheck} &= 38/69 = \ 0.551,\\ \mathrm{CC} &= \sqrt{0.760 \times 0.551} = \ 0.647.\end{aligned}$$

By the CC measure, we can see that the similarity between the partitioning done by the clustering method and the true components is around 65%.

Adjusted Rand index

Rand (1971) proposed the Rand index to measure the agreement between two partitions, and it is defined as

$$\mathrm{RI} = \frac{\binom{n}{2} + \sum_{i,j} n_{ij}^2 - \frac{1}{2}\left(\sum_i n_{i\cdot}^2 + \sum_j n_{\cdot j}^2\right)}{\binom{n}{2}},$$

where $\mathrm{RI} \in [0,1]$. However, the expected value of the Rand index varies from one partition to the other. Thus, Hubert & Arabie (1985) introduced the adjusted Rand index (ARI) which is the normalized version of the Rand index, and it is defined as

$$\mathrm{ARI} = \frac{\mathrm{RI} - \mathbb{E}\{\mathrm{RI}\}}{\max(\mathrm{RI}) - \mathbb{E}\{\mathrm{RI}\}}.$$

Hubert & Arabie (1985) assumed that partitioning illustrated in a contingency table (e.g. Table 5.4a) came from a hypergeometric distribution. Hence, they derived the expectation of the Rand index as

$$\mathbb{E}\{\mathrm{RI}\} = 1 + 2\left[\sum_{i=1}^{k}\binom{n_{i\cdot}}{2}\sum_{j=1}^{\ell}\binom{n_{\cdot j}}{2}\right]\Big/\binom{n}{2}^2 - \left[\sum_{i=1}^{k}\binom{n_{i\cdot}}{2} + \sum_{j=1}^{\ell}\binom{n_{\cdot j}}{2}\right]\Big/\binom{n}{2}.$$

Thus, the simplified definition of the ARI is

$$\mathrm{ARI} = \frac{\sum_{i,j}\binom{n_{ij}}{2} - \left[\sum_i \binom{n_{i\cdot}}{2}\sum_j\binom{n_{\cdot j}}{2}\right]/\binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{n_{i\cdot}}{2} + \sum_j\binom{n_{\cdot j}}{2}\right] - \left[\sum_i \binom{n_{i\cdot}}{2}\sum_j\binom{n_{\cdot j}}{2}\right]/\binom{n}{2}}, \tag{5.9}$$

where $\mathrm{ARI} \in [0,1]$. If ARI gets close to one, similarity between true partition and the partition done by clustering method is high.

The adjusted Rand index for the small data in Table 5.4b can be computed as

$$\begin{aligned}
\textstyle\sum_{i,j}\binom{n_{ij}}{2} &= \binom{2}{2} + \binom{3}{2} + \cdots + \binom{3}{2} + \binom{5}{2} = 38,\\
\textstyle\sum_{i}\binom{n_{i\cdot}}{2} &= \binom{5}{2} + \cdots + \binom{8}{2} = 69,\\
\textstyle\sum_{j}\binom{n_{\cdot j}}{2} &= \binom{5}{2} + \cdots + \binom{5}{2} = 50,\\
\mathrm{ARI} &= \frac{38 - 69 \times 50/300}{1/2 \times (69+50) - 69 \times 50/300} = 0.552.
\end{aligned}$$

For this example, the data are partitioned into four clusters, and the adjusted Rand index for the partitioning of the data shows 55% similarity with the true classification of the data.

Purity index

The purity index (Rendón et al., 2011) takes the average purity for each cluster D_i from the same component, C_j, and purity is defined as the maximum number of elements clustered together. Hence, the purity index is defined as

$$\mathrm{Purity} = \frac{1}{n}\sum_{i=1}^{k}\max_j |D_i \cap C_j|, \tag{5.10}$$

where $\mathrm{Purity} \in [0,1]$. If Purity is close to one, the similarity between the clustering and the true component structure is high.

The purity index for the contingency table in Table 5.4b is

$$\text{Purity} = 1/25(3+5+5+5) = 0.72.$$

Hence, the classification done by the clustering method shows 72% similarity with the true components of the data summarized in Table 5.4b.

Adjusted variation information

The variation information (Meilă, 2007) is an entropy based index which explains the amount of information lost or achieved in clustering, and it is defined as

$$\text{VI}(C,D) = \text{H}(C) + \text{H}(D) - 2\,\text{I}(C,D), \tag{5.11}$$

where $\text{H}(C)$, $\text{H}(D)$ and $\text{I}(C,D)$ are the entropy of true components, C, the entropy of clusters, D, and the mutual information between true components and clusters, respectively. The definition of $\text{H}(C)$, $\text{H}(D)$ and $\text{I}(C,D)$ are

$$\begin{aligned} \text{H}(C) &= -\textstyle\sum_{j=1}^{\ell} \frac{n_{\cdot j}}{n} \log_2\left(\frac{n_{\cdot j}}{n}\right), \\ \text{H}(D) &= -\textstyle\sum_{i=1}^{k} \frac{n_{i\cdot}}{n} \log_2\left(\frac{n_{i\cdot}}{n}\right), \\ \text{I}(C,D) &= \textstyle\sum_{i,j} \frac{n_{ij}}{n} \log_2\left(\frac{n_{ij}/n}{n_{i\cdot}/n \times n_{\cdot j}/n}\right). \end{aligned} \tag{5.12}$$

We can simplify Equation (5.11) using Equation (5.12), so VI can be defined as

$$\text{VI} = -\sum_{i,j} \frac{n_{ij}}{n} \left[\log_2(n_{ij}/n_{i\cdot}) + \log_2(n_{ij}/n_{\cdot j})\right], \tag{5.13}$$

where VI $\leq \log_2(n)$. Meilă (2007) suggested that we can normalize VI by $\log_2(n)$ (NVI $= \dfrac{1}{\log_2(n)}$ VI), but Vinh et al. (2010) proposed that the joint entropy, $\text{H}(C,D)$ is a stricter limit than the $\log_2(n)$. Thus, in this work, we normalize the VI index by $\text{H}(C,D)$ (NVI $= \dfrac{1}{\text{H}(C,D)}$ VI, NVI $\in [0,1]$). VI is a metric index, so we also need to apply adjustment procedure after normalization step. Vinh et al. (2010) adjusted the VI index using the procedure proposed by Hubert & Arabie (1985) ($\frac{\text{index} - \mathbb{E}\{\text{index}\}}{\max(\text{index}) - \mathbb{E}\{\text{index}\}}$), so the adjusted variation information is

$$\text{AVI} = \frac{\text{NVI} - \mathbb{E}\{\text{NVI}\}}{1 - \mathbb{E}\{\text{NVI}\}} = \frac{\text{I}(C,D) - \mathbb{E}\{\text{I}(C,D)\}}{\frac{1}{2}\left[\text{H}(C) + \text{H}(D)\right] - \mathbb{E}\{\text{I}(C,D)\}}. \tag{5.14}$$

The expectation of mutual information is derived by Vinh et al. (2009), and it is formulated as

$$\mathbb{E}\left\{\text{I}(C,D)\right\} = \sum_{i,j} \sum_{\substack{n_{ij}=\max\{n_{i\cdot} \\ +n_{\cdot j}-n,0\}}}^{\min\{n_{i\cdot},n_{\cdot j}\}} \frac{n_{ij}}{n} \log\left(\frac{n_{ij}n}{n_{i\cdot}n_{\cdot j}}\right) \frac{n_{i\cdot}!n_{\cdot j}!(n-n_{i\cdot})!(n-n_{\cdot j})!}{n!n_{ij}!(n_{i\cdot}-n_{ij})!(n_{\cdot j}-n_{ij})!(n-n_{i\cdot}-n_{\cdot j}-n_{ij})!}.$$

	n_{ij}/n	C_1	C_2	C_3	C_4	C_5	$n_{i\cdot}/n$
		Components, C_j					
Clusters, D_i	D_1	0.08	0	0.12	0	0	0.20
	D_2	0	0	0	0.20	0	0.20
	D_3	0	0.20	0.08	0	0	0.28
	D_4	0.12	0	0	0	0.20	0.32
	$n_{\cdot j}/n$	0.20	0.20	0.20	0.20	0.20	1.00

Table 5.5: Tabulated n_{ij}/n for the toy data in Table 5.4b.

After applying the adjustment process, we break the metric property of VI, so AVI measures how much of the true partitioning captured by the partitioning done by a clustering algorithm.

As an example, VI is calculated for the partition in Table 5.4b. For easy computation, the values n_{ij}/n are tabulated in Table 5.5.

Hence, AVI is

$$\mathrm{AVI} = \frac{\mathrm{I}(C,D) - \mathbb{E}\left\{\mathrm{I}(C,D)\right\}}{\frac{1}{2}\left[\mathrm{H}(C) + \mathrm{H}(D)\right] - \mathbb{E}\left\{\mathrm{I}(C,D)\right\}} = \frac{1.581 - 0.456}{\frac{1}{2}\left[2.321 + 1.970\right] - 0.456} = 0.666,$$

where $\mathrm{H}(C) = 2.321$, $\mathrm{H}(D) = 1.970$, $\mathrm{I}(C,D) = 1.581$ and $\mathbb{E}\left\{\mathrm{I}(C,D)\right\} = 0.456$. In terms of the AVI, true components and the partitioning done by the clustering algorithm show 66% similarity.

Discussion

In clustering, if there are labelled data, we can check how well the clustering algorithm performed. In this section, we discuss six of these partition similarity scores in detail. Fowlkes & Mallows (1983) proposed a score (CC) which combines two different scores proposed by Wallace (1983): $\mathrm{CompCheck}$ measures how well the objects coming from the same component are clustered together, and $\mathrm{ClustCheck}$ measures if the objects clustered together are really coming from the same component. Thus, CC does not give any penalty if one component is divided into many clusters. However, Hubert & Arabie (1985) claimed that all partition similarity scores including CC should be corrected by chance, and they proposed the adjusted Rand index (ARI) which measures the agreement between two partitions. Another index is Purity (Rendón et al., 2011) which measures the average of the maximum number of objects clustered together, so Purity does not account the small size clusters. The final score we discussed is the variation information (VI) (Meilă, 2007). VI is a metric score, and it measures the amount of information lost or gained by clustering. Then Vinh et al. (2010) proposed the adjusted variation information (AVI) which allows us to break the metric property and to comment on how much of the true partitioning is captured by the clustering algorithm. We check the similarity between the partitioning found by a clustering method and the "truth" of the toy data in Table 5.4b using different external scores. While Purity finds 70% similarity between the clusters and the "truth", the similarity found by ARI is around 55%. Thus, the higher similarity found by Purity might be the result of the allocation by chance. Other scores, CC and AVI, are around 65%.

5.4 Lifting the results of hierarchical clustering

In previous section, we discussed different cluster validity indices which find the number of clusters in a data set, and check the similarity between classification done by a clustering method and the true components. In this section, we propose a new method based on lifting which finds the location of each cluster on a hierarchically built tree.

5.4.1 Cluster selection by denoising of compactness

Jansen et al. (2009) proposed a method to use lifting on trees; details of their algorithm are given in Section 3.8.2. Hence, to apply the lifting algorithm on a tree, the neighbourhood structure and the lengths of the edges between each node and its neighbours in each agglomeration step need to be known. We also need a function value for each node. Since we can easily compute the edge lengths and merged nodes in each agglomeration step of the hierarchical clustering algorithm, we need to create a meaningful function value for each node. In some cases, there may be a suitable value available within the context of the data set. However, we wish to present a more generic methodology, and hence, we seek a function value which can always be used. One option is a compactness score.

We assume that each node in a tree is a candidate of a cluster, so to be able to calculate the compactness for each node, the distance between the midpoint of the cluster and each point in the cluster (each leaf under the node of interest) is calculated. The mean of these distances is used as a measure of compactness.

Let cluster D_i contain $n_{i\cdot}$ observations $\{x_{i_1}, \dots, x_{i_{n_{i\cdot}}}\} \in \mathbb{R}^p$, and the centroid of cluster D_i is $\bar{D}_i = n_{i\cdot}^{-1} \sum_{k=1}^{n_{i\cdot}} x_{i_k}$. We then let the Euclidean distance between the centroid $\bar{D}_i$ and observation x_{i_k} be

$$\delta_{i,i_k} = \left\| \bar{D}_i - x_{i_k} \right\|_2 .$$

We can then define the compactness γ_i of cluster D_i to be the mean distance of the points in D_i from $\bar{D}_i$, so

$$\gamma_i = \frac{1}{n_{i\cdot}} \sum_{k=1}^{n_{i\cdot}} \delta_{i,i_k}. \tag{5.15}$$

Let continue the example in Section 5.2. If the first cluster (first agglomeration order) includes x_1 and x_3, the compactness of first cluster, γ_1, will be

$$\gamma_1 = \frac{\delta_{11} + \delta_{13}}{2},$$

where

$$\begin{aligned}
\delta_{11} &= \sqrt{\left(x_{1,1} - \frac{x_{1,1}+x_{3,1}}{2}\right)^2 + \left(x_{1,2} - \frac{x_{1,2}+x_{3,2}}{2}\right)^2},\\
\delta_{13} &= \sqrt{\left(x_{3,1} - \frac{x_{1,1}+x_{3,1}}{2}\right)^2 + \left(x_{3,2} - \frac{x_{1,2}+x_{3,2}}{2}\right)^2},
\end{aligned}$$

where $x_{i,j}$ is the jth coordinate of ith data point. The second cluster (second agglomeration order) includes the first cluster $\{x_1, x_3\}$ and x_2, so the compactness of second cluster, γ_2, is

$$\gamma_2 = \frac{\delta_{21} + \delta_{23} + \delta_{22}}{3},$$

where

$$\begin{aligned}
\delta_{21} &= \sqrt{\left(x_{1,1} - \frac{x_{1,1}+x_{3,1}+x_{2,1}}{3}\right)^2 + \left(x_{1,2} - \frac{x_{1,2}+x_{3,2}+x_{2,2}}{3}\right)^2},\\
\delta_{23} &= \sqrt{\left(x_{3,1} - \frac{x_{1,1}+x_{3,1}+x_{2,1}}{3}\right)^2 + \left(x_{3,2} - \frac{x_{1,2}+x_{3,2}+x_{2,2}}{3}\right)^2},\\
\delta_{22} &= \sqrt{\left(x_{2,1} - \frac{x_{1,1}+x_{3,1}+x_{2,1}}{3}\right)^2 + \left(x_{2,2} - \frac{x_{1,2}+x_{3,2}+x_{2,2}}{3}\right)^2}.
\end{aligned}$$

Note that in an agglomerative cluster analysis, the clusters are nested, and hence a cluster D_i has sub-clusters contained in it (unless D_i is just a single observation).

After calculating the compactness for each cluster, the lifting algorithm described in Section 3.8.2 can be applied to the tree using the compactness γ_i as the function value at node i. In our algorithm, we apply the lifting algorithm using the first order neighbours, and we create the initial branch lengths using the height of each node found by hierarchical clustering. After lifting a node in the tree, we remove the lifted node from the tree. We need to relink the tree, and we do this by linking the neighbours of the removed (lifted) node using the minimum spanning tree of the remaining nodes. After applying the lifting algorithm, we denoise the lifted compactness values using one of the wavelet shrinkage methods described in Section 2.10. In this study, we use the empirical Bayesian wavelet shrinkage method, where the parameters are estimated using their marginal maximum likelihood estimators, and the threshold is estimated using the posterior median. Details of this method is discussed in Section 2.10.3, and it is available in the **EbayesThresh** package in **R** (Silverman, 2012). To apply Bayesian thresholding, we can set artificial levels using the idea explained in Section 3.6.1. Since Bayesian thresholding has the normality assumption with $\sigma^2 = 1$, we can normalize detail coefficients using the variance term defined in Equation (3.13). After denoising the detail coefficients, if the coefficient

Internal node	Joined pairs		Branch lengths	
7	1	2	0.118	0.118
8	4	5	0.401	0.401
9	7	3	0.514	0.632
10	8	6	0.454	0.855
11	10	9	2.298	2.521

Table 5.6: Hierarchical clustering results for the toy data in Table 5.1.

for the root is greater than λ, this means that there is a considerable divergence between its neighbours. Then we check all the nodes in the tree. If the denoised detail coefficients for any node and all its child nodes are less than or equal to λ, we assume that one cluster is located at this node. Thus, we consider any divergence between clusters less than or equal to λ as noise.

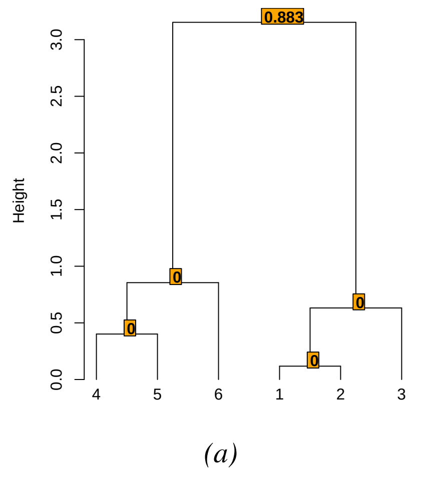
One possible choice for λ is zero, so if any cluster and all its sub-clusters are denoised to zero, we do not allow any divergence between neighbours; hence, all the objects are placed in the centroid of the cluster they are allocated to. However, there might be small departures from the centroid of clusters, and our choice of λ places these objects into separate clusters. We may end up having high number of clusters just because of forcing the algorithm to cluster only the ones located in the centroid of each cluster. To avoid this problem, allowing small departures from the centroid can be logical. Thus, we need to choose the threshold λ. For now, we shall choose λ manually; we shall consider automatic choice of λ later in Chapter 6.

Example 5.4.1. *The toy data in Table 5.1 is used in this example.*

The first step of the algorithm introduced in Section 5.4 is to cluster the data hierarchically. In this case, we use Euclidean distances and complete linkage. The process of building the tree is explained in Section 5.3. As seen in Figure 5.1, we can easily obtain the information we need to apply the lifting algorithm to this tree because all we need to know is which pairs are joined and branch lengths at each agglomeration step. These details are tabulated and given in Table 5.6.

After withdrawing the information we need from hierarchical clustering, we need one more variable to apply lifting on a tree: the compactness for each cluster (internal node) which is calculated using Equation (5.15). *Here, compactness for the clusters are found to be* $\gamma = (0.059, 0.201, 0.255, 0.339, 1.254)$*, respectively. We can apply lifting algorithm given in Section 3.8.2 to this tree, then we can denoise the lifted compactness values by applying the Bayesian wavelet shrinkage method, given in Section 2.10.3. After denoising the compactness values, the tree is given in Figure 5.3a. Internal nodes are labelled with denoised detail coefficient for each cluster.*

Using the denoised detail coefficients, we can decide where to cut the tree. If any



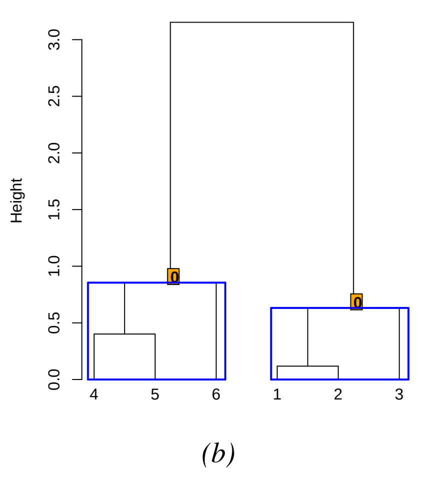


Figure 5.3: Illustration of the toy data in Table 5.1 with the results of our lifting method. (a): the labelled dendrogram with the denoised detail coefficients. (b): the clustering pattern found by our lifting method.

internal node and all its child nodes are denoised to zero or less than zero, we can treat them as one cluster. Under the light of this information, the tree is cut and illustrated in Figure 5.3b after cutting the tree. As it is seen, the tree is cut from the exact height of each cluster, and two clusters are found.

5.4.2 Dealing with outliers

When hierarchical clustering is applied, all objects within the range of the data are clustered, but data can have some outliers. How will we treat these outliers?

When the dendrogram of a data is drawn, any outlier in the data can be detected easily. Other internal indices (some of them are discussed in Section 5.3.1) find the optimal number of clusters, and they suggest that we can cut the tree from a certain height which gives the number of clusters we found. However, they do not point the exact height of each cluster. Because of that even if they treat any data point as an outlier, they create separate clusters. However, when we apply the lifting algorithm to find clusters on the tree, outliers are not clustered at all. Thus, if any objects in a data set are not clustered, they are treated as outliers.

We report the results of the clustering with the lifting algorithm in two slightly different ways.

- In the first version, we create one separate cluster for all outliers. If the algorithm detects k clusters, we will have one more cluster for outliers, so we have $k + 1$ clusters in total. We refer to this version as “Lifting”.

- In the second version, we remove outliers. Thus, we have k clusters which is the optimal number of clusters found by the lifting algorithm. We refer to this version as “Lifting2”.

We create the second version to explore the effect of removing outliers from the data set to our clustering scheme. We will observe if there is any difference between these two arrangements in comparison studies done in Sections 5.5 and 5.6.

5.5 Simulation study

In this section, we apply our Lifting algorithm, introduced in Section 5.3.3, to some synthetic data sets, and we compare the performance of our algorithm with other CVIs, given in Section 5.3.1. For this purpose, four different data structures are created for our simulation study. Two of them are mixtures of normally distributed data with five-component in two dimensions. One of these two normally distributed data sets has bigger variation in each component than the other. The third data set is structured as concentric circles with three-component in two dimensions. The final data set includes six different components in two dimension, and this data consists of non-normally distributed components. Data sets generated are given in Table 5.7, and some notations we used to generate these data structures are

$$\begin{aligned}
x_{ijk} &: j\text{th element in } k\text{th dimension of } i\text{th component of data set } x,\\
x_{i\cdot k} &: k\text{th dimension of } i\text{th component of data set } x,\\
x_{ij\cdot} &: j\text{th element of } i\text{th component of data set } x,\\
x_{\cdot\cdot k} &: k\text{th dimension of data set } x,\\
y &\sim N_2(0_2, I_2), \text{ and } y_k \text{ is the } k\text{th dimension of data set } y,\\
z &\sim N(0,1),\\
a &: \text{the equally spaced sequence from } -2 \text{ to } 8 \text{ by } 0.05.
\end{aligned}$$

After generating data structures, we standardize each data set ($x_{\cdot\cdot k} = \frac{x_{\cdot\cdot k} - \text{mean}(x_{\cdot\cdot k})}{\text{sd}(x_{\cdot\cdot k})}$, where $k \in \{1, 2\}$), and scatter plots of these simulated data sets are given in Figure 5.4. In our simulation study, we use hierarchical clustering with single linkage, and the number of the replicate data sets per data structure is set at 1000. We summarize the results by comparing the clustering results which are produced by different CVIs and our Lifting algorithm. In addition, the partition done by the CVIs and our Lifting algorithm are compared in terms of external cluster validity scores, given in Section 5.3.3.

We use the **NbClust** package (Charrad et al., 2014) in **R** for the CH index, H index and KL index, and Sil statistic, so we need to carefully decide the upper boundary for the number of clusters since this choice has a considerable effect on results. We set the maximum number of clusters as 50, but we define the upper boundary for the Gap statistic lower (15) because the bootstrapping step in the Gap statistic increases the time cost of the

First data set	Second data set	Third data set	Fourth data set
$\Sigma = \text{diag}(0.3, 0.3)$	$\Sigma = \text{diag}(0.6, 0.6)$	$\Sigma = \text{diag}(0.005, 2)$	
$x_1 \sim N_2(0_2, \Sigma)$		$x_1 = \frac{y}{\sqrt{\sum_{j=1}^{2} y_j^2}}$	$x_{1\cdot 1} = z,$
$x_2 \sim N_2((4, -6), \Sigma)$		$x_2 = 2\frac{y}{\sqrt{\sum_{j=1}^{2} y_j^2}}$	$x_{1\cdot 2} = z^2 + N(0, 0.2)$
$x_3 \sim N_2((4, 0), \Sigma)$		$x_3 \sim N_2(0_2, \Sigma)$	$x_{2\cdot 1} = z - 3,$
$x_4 \sim N_2((0, -6), \Sigma)$			$x_{2\cdot 2} = 12 - (z^2 + N(0, 0.2))$
$x_5 \sim N_2((2, -3), \Sigma)$			$x_{3\cdot 1} = z - 6.5,$
			$x_{3\cdot 2} = z^3 + N(0, 0.3) - 10$
			$x_{4\cdot 1} = z - 5.5,$
			$x_{4\cdot 2} = z^3 + N(0, 0.3) - 15$
			$x_5 \sim N_2((1, -10), I_2)$
			$x_{6.1} = a,$
			$x_{6.2} = \sin(a) + N(0, 0.3) - 20.$

Table 5.7: The formula used to generate simulated data structures.

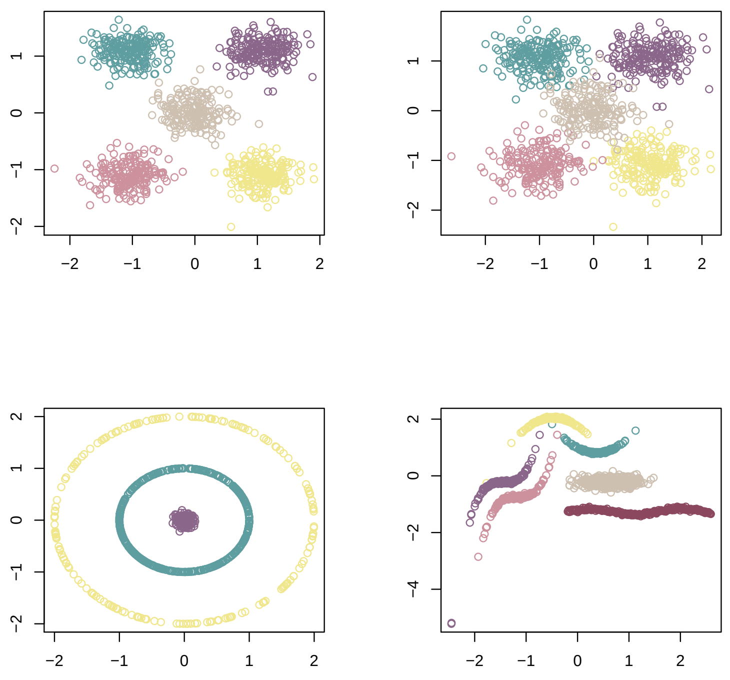


Figure 5.4: Scatter plots of four simulated data sets from different data structures. Top left: five-component normally distributed data set with small variation. Top-right: five-component normally distributed data set with larger variation. Bottom-left: three-component concentric circle data set. Bottom-right: six-component non-normally distributed data set.

study. Another arrangement for the Gap statistic is the choice of reference data sets. We pick the simplest way proposed by Tibshirani et al. (2001): they are generated uniformly over the range of the data, and the number of reference data sets is set at 10. Even though the Gap statistic is available in **NbClust** package, we use the Gap statistic implemented in **clusterGenomics** package (Nilsen & Lingjaerde, 2013) in **R** which allows us to choose which method we use to generate reference data sets.

5.5.1 Five-component normally distributed data with low variance

The first simulation is the five-component normally distributed data with small variation. In this case, the size of the each component is 200, and threshold, λ is set at 0.2. Figure 5.5 illustrates the simulation results for this data. The Gap statistic shows the poorest performance for this data structure, and the partition found by this index varies in a large range. However, variation of the partitions found by Mclust, the CH index and the KL index are almost zero, and the similarity measures (external validity scores) demonstrate that the performance of these methods are high. Other methods also capture the partitions with high performance, but there are some variations. For our Lifting method, the reason of the small variation is using the same constant threshold for each run. Lifting2 illustrates that removing the outlier cluster does not make any clear change.

The number of clusters we found for each CVI is illustrated with a bar chart given in Figure 5.6. The maximum number of clusters found by our Lifting method is taken as an upper boundary. The vertical axis shows the percentage of replicates. The Gap statistic generally finds the number of clusters less than the true component number, and Mclust always partitions the data into five clusters. However, other methods generally find the number of clusters in the range $[6, 10]$. Lifting also mostly partitions the data set in the range $[6, 10]$ clusters, but when we remove the outlier cluster, Lifting2 finds the number of clusters either five (by around 40% of the replicates) or in the range $[6, 10]$ (by around 40% of the replicates).

Table 5.8 tabulates the average number of clusters and average similarity measures which help to easily compare the performance of the indices. This tabulated results clearly shows that while the performance of the Gap statistic is much lower than the other indices, the highest similarity measures are found by Mclust. Mclust is a parametric method which assumes normally distributed data, so this result was expected because all the components are normally distributed in this data set. Our Lifting method also captures clusters with high performance. In Lifting, the average number of clusters is found as eight where one of the clusters includes just outliers. When we remove the outlier cluster, the performance of the Lifting2 method slightly increases.

Finally, scatter plots are drawn for each CVI using just one replicate. This illustration gives an idea about what kind of partitioning is done by different CVIs. This comparison is demonstrated in Figure 5.7 which illustrates that all the methods capture the main

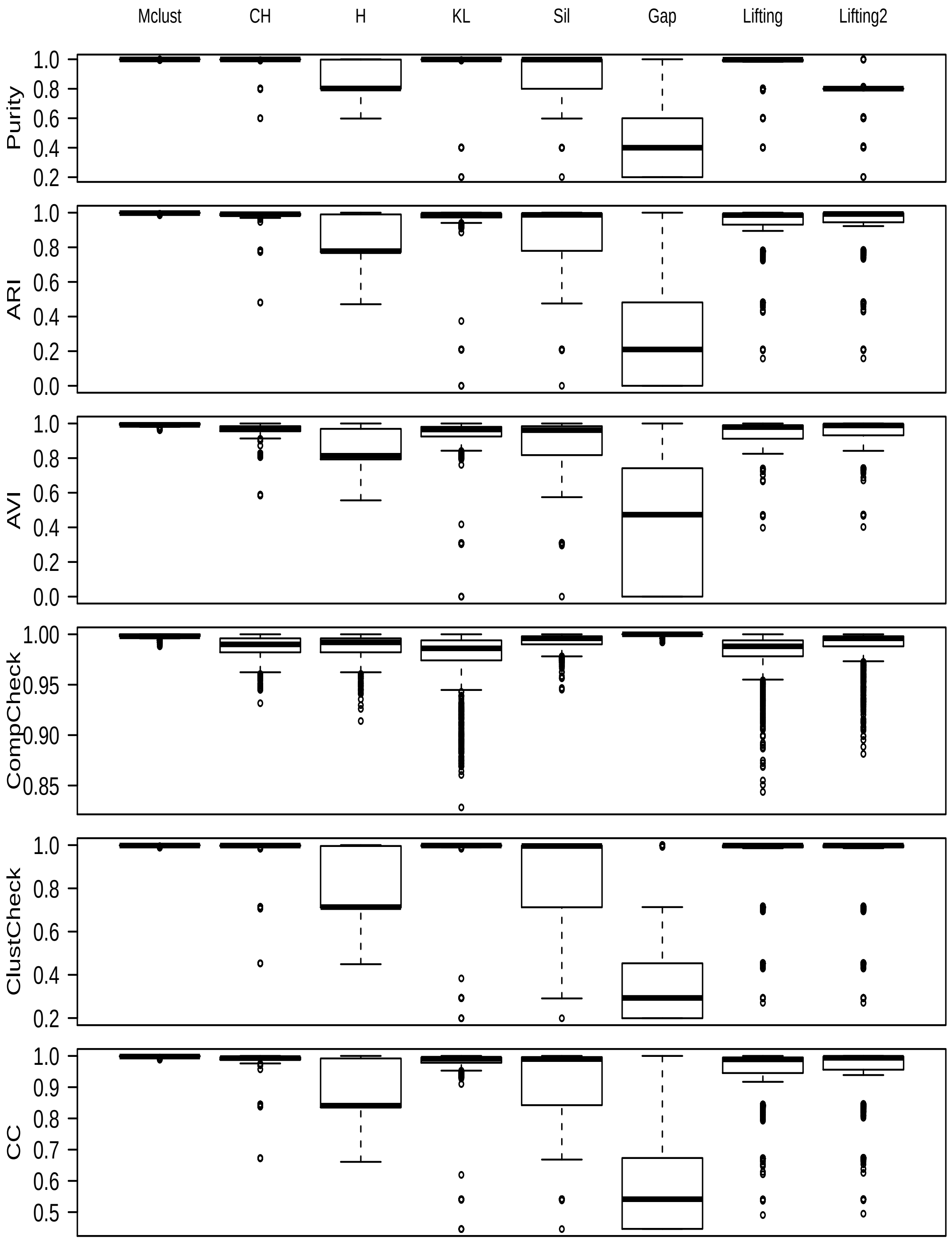


Figure 5.5: Box plot: the comparison of CVIs for the five-component normally distributed data with low variance.

clusters except the Sil and the Gap statistics. The Gap statistic does not partition the data, and it finds one big cluster. We have noticed that the Gap statistic is tended to find a big cluster, and it also finds a quite different partitioning when the reference data set changes.

The same study is repeated by clustering the data set using complete linkage, and we observe that the performance of the CVIs slightly increases except the KL index. Even

Index	Mclust	CH	H	KL	Sil	Gap	Lifting	Lifting2
N	5	9	9	14	7	3	8	7
Purity	0.999	0.985	0.845	0.951	0.881	0.506	0.937	0.759
ARI	0.998	0.976	0.815	0.921	0.853	0.373	0.916	0.921
AVI	0.993	0.961	0.837	0.899	0.870	0.519	0.946	0.954
CompCheck	0.998	0.987	0.988	0.975	0.993	1.000	0.979	0.988
ClustCheck	0.998	0.978	0.778	0.943	0.837	0.430	0.914	0.914
CC	0.998	0.982	0.871	0.950	0.903	0.633	0.942	0.946

Table 5.8: The comparison of CVIs for the five-component normally distributed data with low variance. First row is for the average number of clusters, and others are for the average similarity scores.

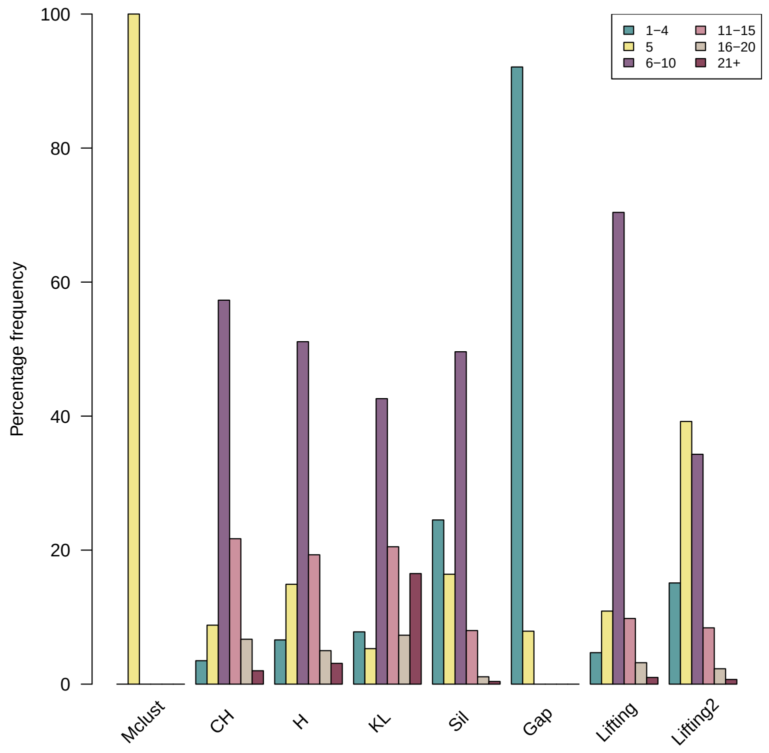


Figure 5.6: Bar chart: the comparison of number of clusters for the five-component normally distributed data with low variance. Legends illustrate the number of cluster range. Vertical axis shows the percentage of replicates, and the horizontal axis is for the different CVIs.

though the performance of our Lifting method slightly decreases, it still captures the high percentage of the true components. We also repeat the study by picking a different upper boundary for the number of clusters for the CH index, H index, KL index, and Sil statistic. The performance of these indices slightly decreases, and we provide the comparison of the tabulated results in Appendix A.

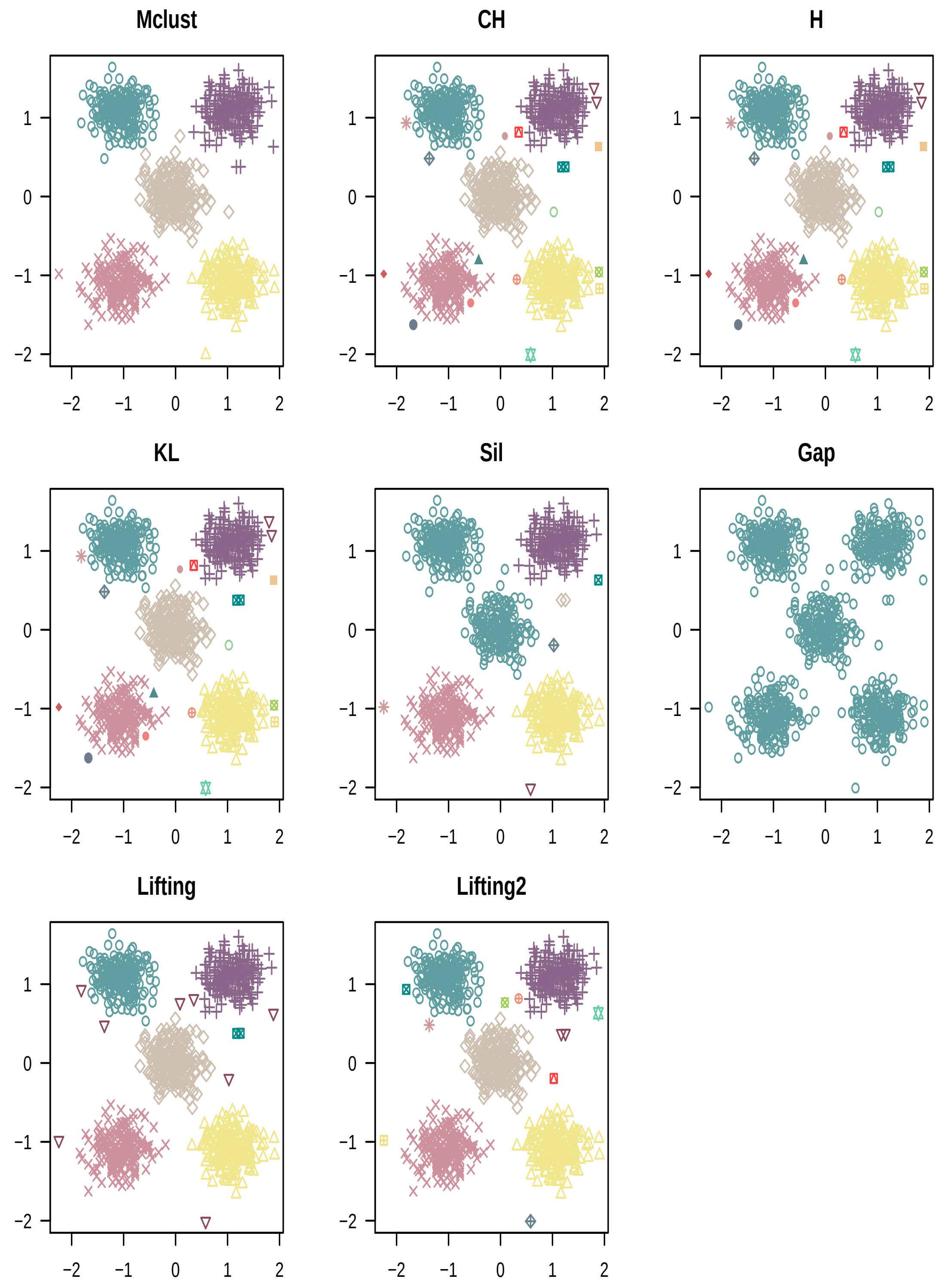


Figure 5.7: A single realisation of the five-component normally distributed data with low variance. Scatter plots illustrate the clustering solution chosen by various methods.

5.5.2 Five-component normally distributed data with larger variation

Our second data structure is the five-component normally distributed data set with larger variation. The size of the each component is 200 in this data set, and the threshold is chosen as 0.3. Simulation results for this data structure are summarized in Figure 5.8. Variation of the partitions found by Mclust is almost zero again, and the similarity measures demonstrate that the performance of Mclust is high. Our Lifting method also partitions the data successfully, but high variation is observed. The Sil and Gap statistics fail to cluster this data set since the external scores for these indices are around zero. The performance of other indices are close to zero, but the average scores for the Lifting is slightly higher than others. In addition, Lifting2 illustrates that removing the outlier cluster increases the similarity measures slightly.

Bar charts illustrating the number of clusters found by CVIs for this data structure are given in Figure 5.9. Mclust always partitions the data into five clusters. The Sil statistic generally finds the number of clusters to be less than five, and the Gap statistic always partitions the data less than five clusters. The variation of the number of clusters in other methods is high. We can check the average number of clusters and average similarity measures from Table 5.9. Mclust gives the highest similarity measures, and our Lifting method and the CH index follow Mclust. High performance of Mclust was also expected for this data structure since all the components come from normal distributions. The Gap statistic repeats its behaviour seen in the previous data structure, and it fails to partition the data. We can also check the scatter plot (Figure 5.10) for one replicate of the simulated data to have an idea what kind of partitions are done by different CVIs. Mclust and our Lifting methods capture the main five clusters, and other methods either combine two or three main components together or just find one main cluster.

We also repeat this study after applying the complete linkage in clustering stage. We notice that the performance of the CH index, H index and Sil statistic dramatically increases. There is almost no difference on the performance of our Lifting method. We also set the upper boundary for the number of clusters as 15, and we notice that the performance of the CH index, H index, KL index and the Sil index is around zero, so this proves the importance of the boundary choice us. The related tables for these extra settings are given in Appendix A.

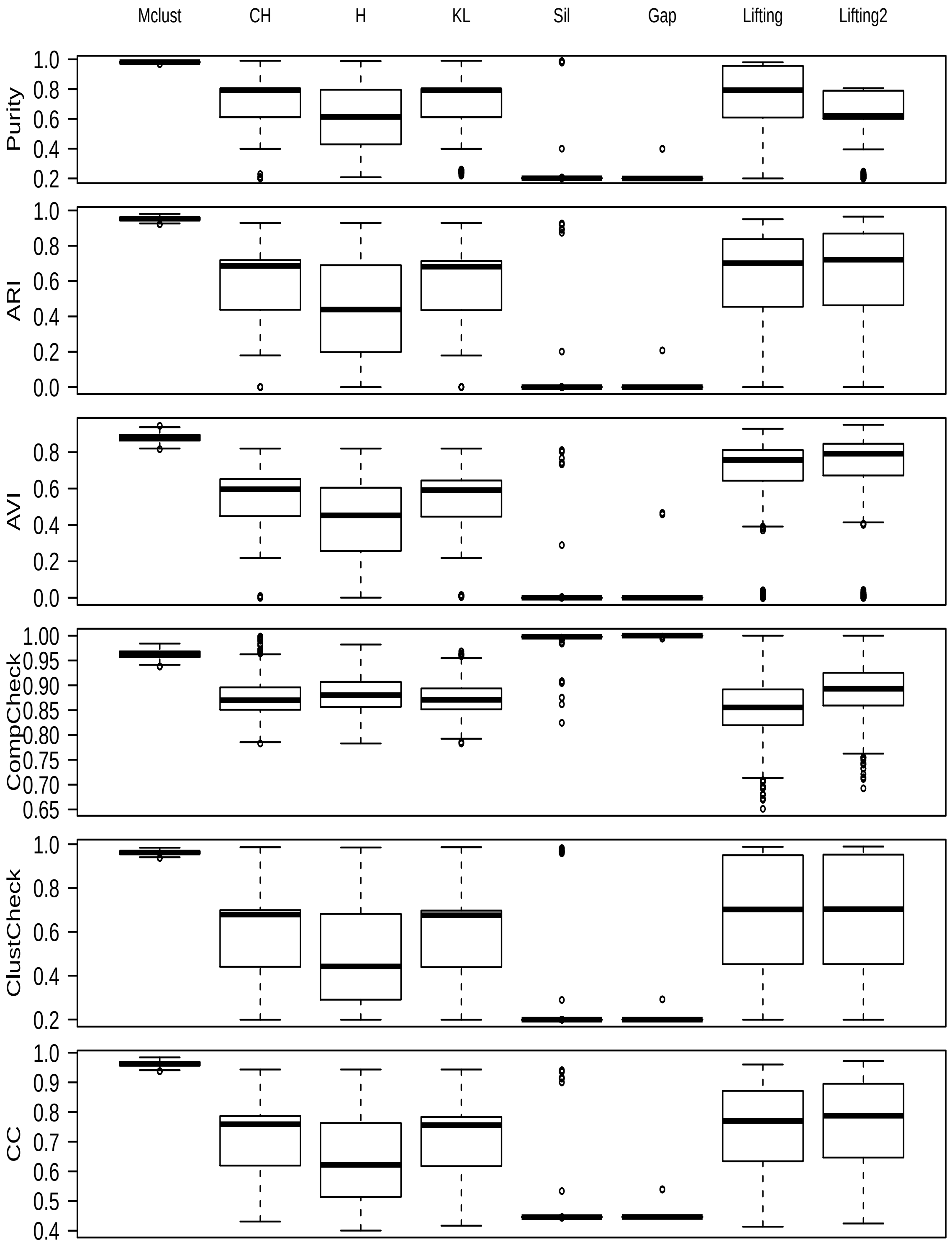


Figure 5.8: Box plot: the comparison of CVIs for the five-component normally distributed data with larger variation.

Index	Mclust	CH	H	KL	Sil	Gap	Lifting	Lifting2
N	5	38	37	39	2	1	19	18
Purity	0.981	0.724	0.595	0.712	0.205	0.200	0.750	0.627
ARI	0.953	0.588	0.424	0.573	0.005	0.000	0.629	0.646
AVI	0.879	0.544	0.418	0.533	0.004	0.001	0.690	0.719
CompCheck	0.962	0.876	0.883	0.874	0.997	1.000	0.856	0.891
ClustCheck	0.962	0.612	0.470	0.595	0.203	0.199	0.678	0.679
CC	0.962	0.718	0.628	0.708	0.448	0.447	0.744	0.760

Table 5.9: The comparison of CVIs for the five-component normally distributed data with larger variation. First row is for the average number of clusters, and others are for the average similarity scores.

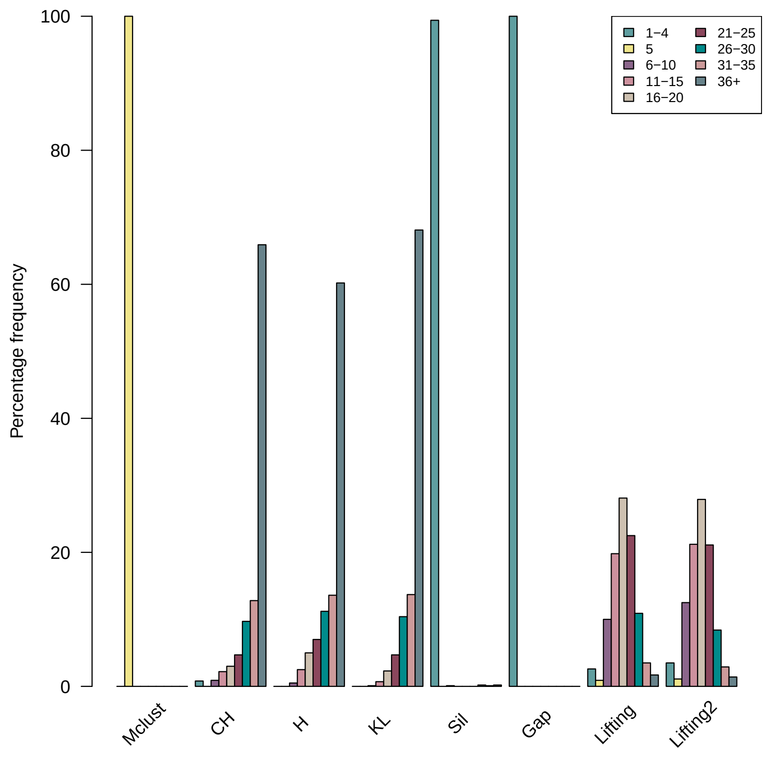


Figure 5.9: Bar chart: the comparison of number of clusters for the five-component normally distributed data with larger variation. Legends illustrate the number of cluster range. Vertical axis shows the percentage of replicates, and the horizontal axis is for the different CVIs.

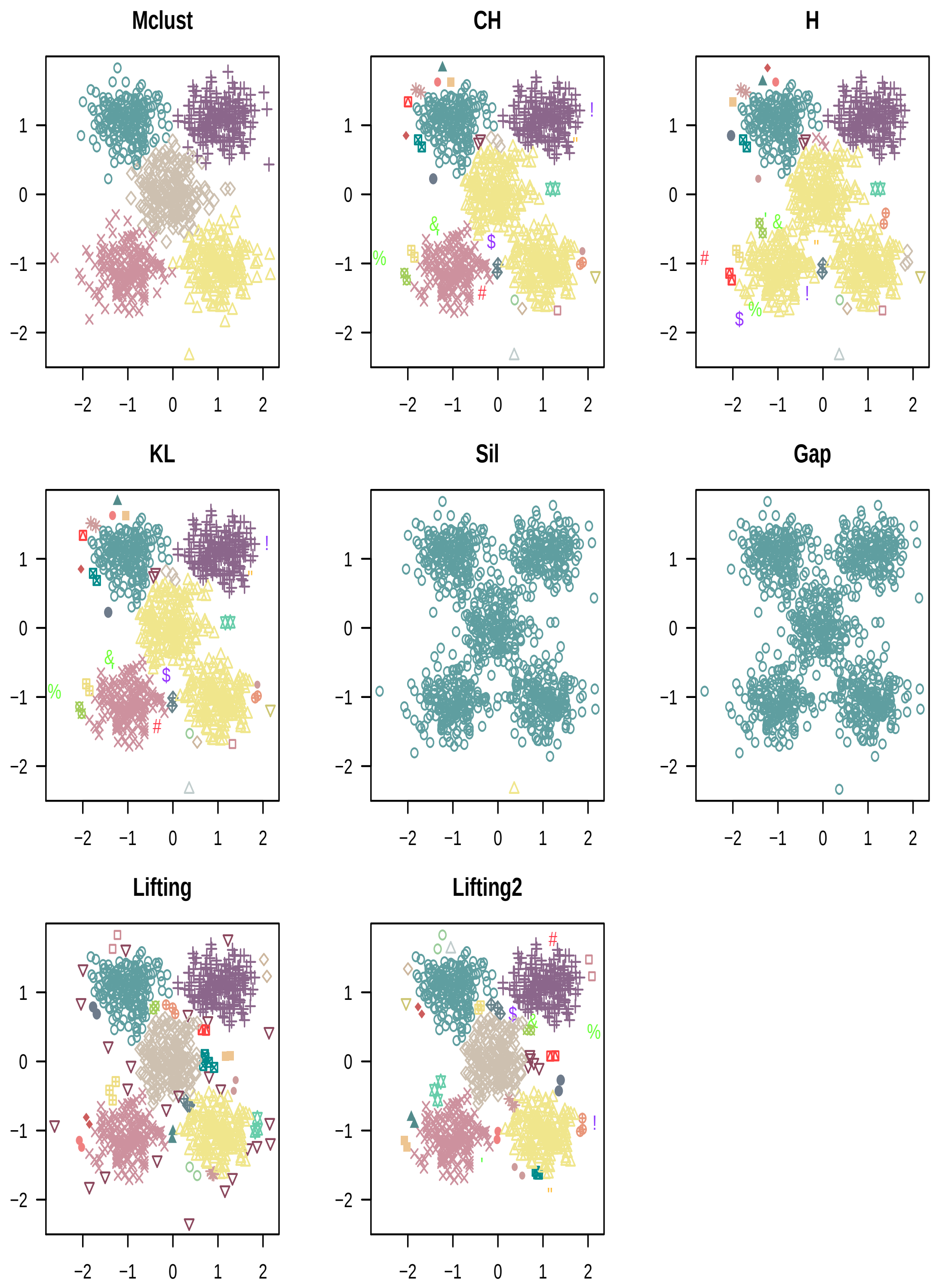


Figure 5.10: A single realisation of the five-component normally distributed data with larger variation. Scatter plots illustrate the clustering solution chosen by various methods.

5.5.3 Three-component concentric circle data

Our third data structure is the three-component concentric circle data. The size of the component in the middle is 500, and the size of the other two components are 200. The threshold parameter is set at 0.78. Box plot comparison for this data structure in Figure 5.11 illustrates that variation of the partitions found by Mclust is low. However, the similarity measures demonstrate that the performance of Mclust is not good for this data structure because Mclust's assumption of normally-distributed data is grossly violated in this case. Thus, Mclust could not capture the clusters well for this data. The variation of the number of clusters is higher for our Lifting method, but the similarity measures are higher than the Mclust. For this data structure, the CH index, H index, KL index, Gap statistic and Sil statistic do better partitioning with high similarity measures. The number of replicates where our Lifting method finds outliers are low, so the Lifting and Lifting2 results are mostly the same.

Bar charts representing the number of partitions are shown in Figure 5.12. Mclust always partitions the data into the range $[7, 9]$ clusters, and the Sil statistic generally finds the number of clusters to be less than the true number of components, three. The number of clusters found by other methods varies. Over 40% of the replicates, the CH and H indices find the number of clusters in the range $[4, 6]$ while the Gap statistic clusters over 70% of the replicates in this range. The KL index most commonly partitions the data into three clusters (40% of the replicates). We can check the average number of clusters and average similarity measures from Table 5.10. The average number of clusters found by the Sil statistic is three, and it shows the lowest performance within the CVIs to capture the true components in terms of all similarity scores. The highest similarity measure is generally calculated for the H index while the lowest one is calculated for Mclust. The average similarity measures for the CH index, Gap statistic and KL index are always high (e.g. ARI is above 90%). The average similarity measure for our Lifting method is lower than other CVIs. To see what kind of partition these methods do, we can check scatter plots for each method for just one replicate, given in Figure 5.13. Mclust finds high number of clusters, and it also combines the right part of the middle and outer circle components. This explains its poor classification for this data structure. Since the middle and outer circle components do not come from a normal distribution, Mclust presents a low performance. We can also have an idea why other CVIs perform better than the Lifting. As it is seen from Figure 5.13 which illustrates the partitioning for a single realisation of the concentric circles data structure, the Lifting finds fewer or equal number of clusters than the CH, H and KL indices and the Gap statistic, where the CH index, H index and Gap statistic do the same partitioning for this replicate. These indices divide just one true component into many clusters, but our Lifting method divides both the middle

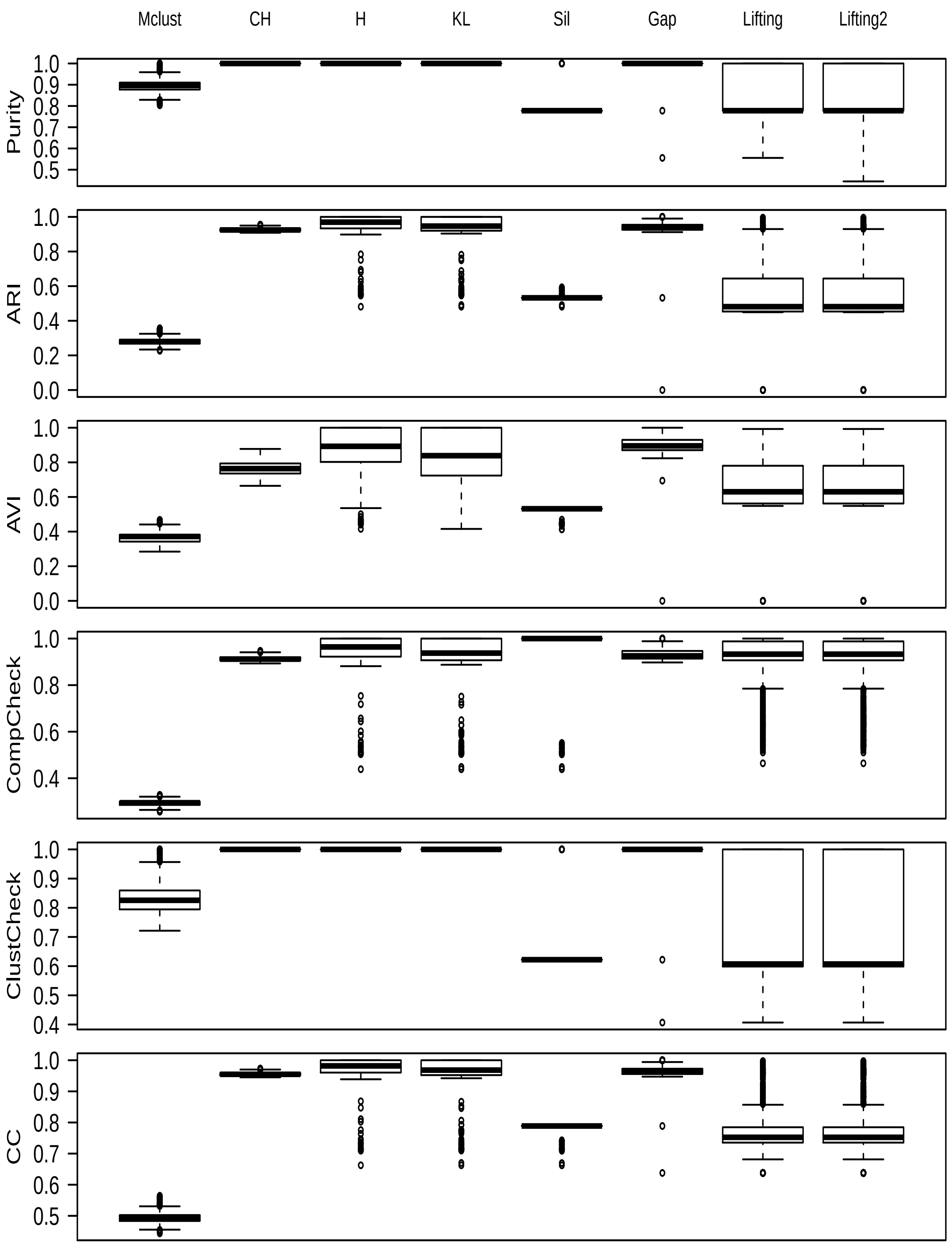


Figure 5.11: Box plot: the comparison of CVIs for the three-component concentric circle data.

and outer circle components into two separate clusters. Thus, capturing two components increases the performance of other indices. In addition, the Sil statistic finds two clusters by combining two components into one cluster, so not dividing any component into many small clusters increases its performance to capture the true components when we compare with our Lifting methods.

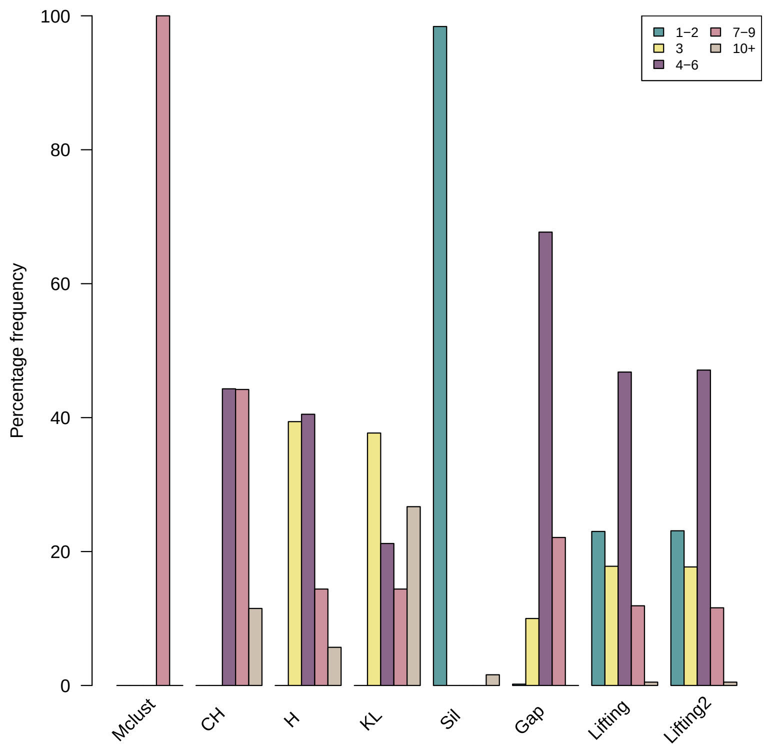


Figure 5.12: Bar chart: the comparison of number of clusters for the three-component concentric circle data. Legends illustrate the number of cluster range. Vertical axis shows the percentage of replicates, and the horizontal axis is for the different CVIs.

Index	Mclust	CH	H	KL	Sil	Gap	Lifting	Lifting2
N	9	7	6	8	3	5	4	4
Purity	0.894	1.000	1.000	1.000	0.781	0.999	0.797	0.789
ARI	0.280	0.926	0.959	0.947	0.533	0.943	0.458	0.458
AVI	0.366	0.763	0.882	0.846	0.530	0.903	0.543	0.543
CompCheck	0.293	0.913	0.952	0.939	0.992	0.935	0.888	0.888
ClustCheck	0.832	1.000	1.000	1.000	0.628	0.999	0.684	0.684
CC	0.494	0.956	0.975	0.968	0.788	0.966	0.758	0.758

Table 5.10: The comparison of CVIs for the three-component concentric circle data. First row is for the average number of clusters, and others are for the average similarity scores.

We have also tried a low choice of upper boundary for the number of clusters for the CH index, H index, KL index and Sil statistic, but for this data structure, the performance of indices do not show much difference. The results are tabulated and given in Appendix A.

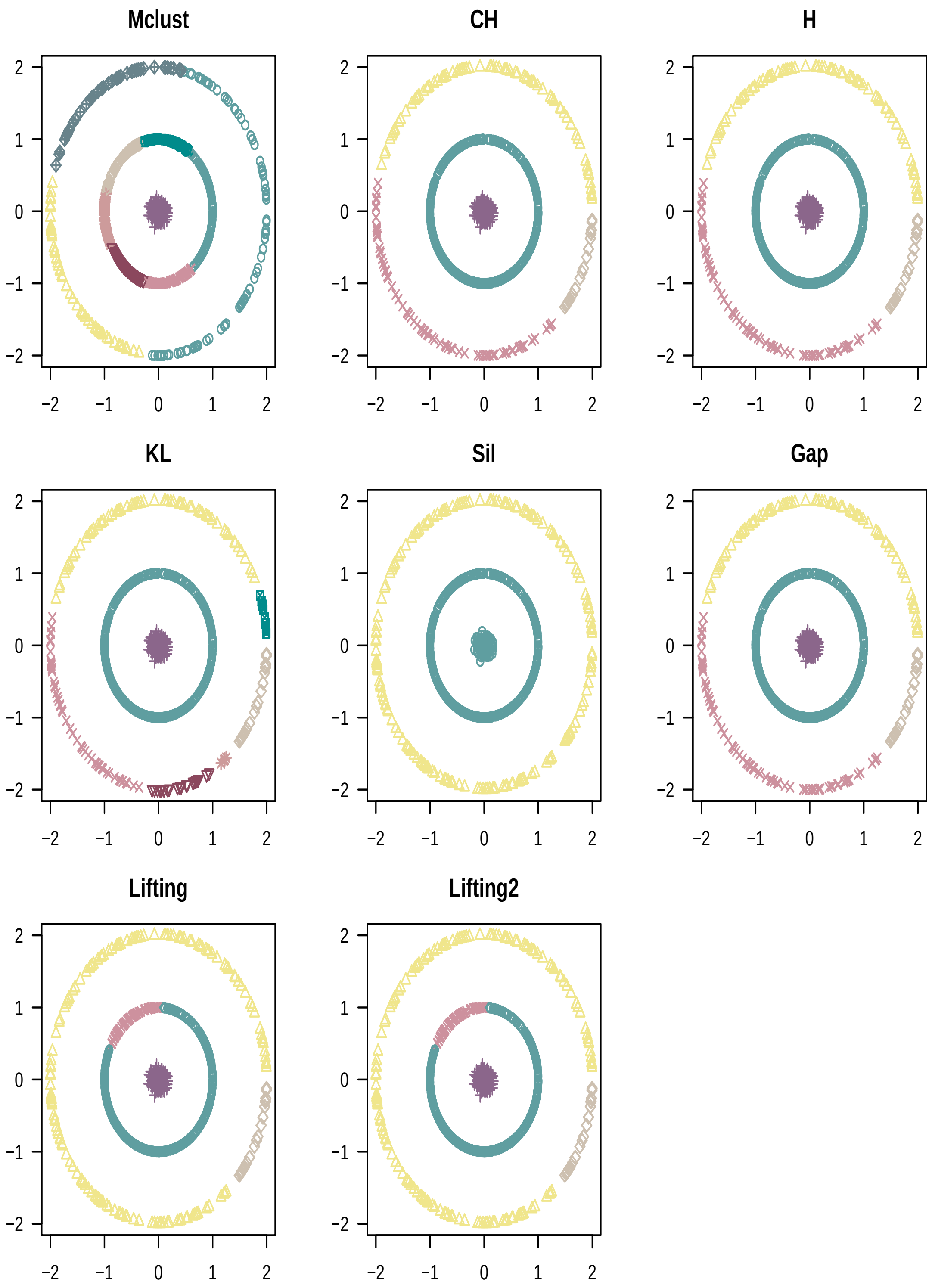


Figure 5.13: A single realisation of the three-component concentric circle data. Scatter plots illustrate the clustering solution chosen by various methods.

5.5.4 Six-component non-normally distributed data

Our final data structure is the six-component non-normally distributed data. In this data structure, there are five non-normal components with size 200, and one normally distributed data component with size 500 which is the one having the elliptic shape. The threshold parameter is set at 0.145, and box plot comparison of simulation results are summarized in Figure 5.14. When different CVIs are compared for this data structure, it is obviously seen that the Gap statistic fails to partition the data (e.g. $\text{AVI} \approx 0.1$). The performance of the H index, KL index and Sil statistic are lower than others, and the similarity scores for the Sil statistic change in wide range. The similarity measures for other methods are high, but the highest similarity measures are calculated for the CH index, then our Lifting method follows it. When the outlier cluster is removed, the performance of our Lifting method slightly increases.

Bar charts, given in Figure 5.15, demonstrate that the Gap statistic mostly finds the number of clusters to be fewer than the true component number in almost all the replicates. Mclust always finds the number of clusters to be in the range $[7, 11]$, and the number of clusters found by other indices except the Sil statistic and our Lifting methods are also mostly in this range. The average number of partitions and similarity measures can be seen in Table 5.11. It is obvious that the Gap statistic fails because the average number of clusters is two, and the average similarity measures are low (e.g. ARI ≈ 0.1). If we compare the CVIs in terms of the different similarity measures, the highest similarity measures are seen in the CH index, then the Lifting2 method follows the CH index. Results for our Lifting method are also close to those of the Lifting2. Scatter plot comparison for one replicate is also checked for this data structure to see what kind of partitioning is done by each CVI, and given in Figure 5.16. From this comparison study, the Sil statistic, KL index and Gap statistic show the poorest performance by clustering all components in one cluster, and this explains why the similarity measures for the KL index are slightly lower than others. We also have an idea why the performance of the H index is lower in the simulation study. The H index combines the closer components in one cluster, so it fails to capture true components if they are close to each other. The scatter plot for Mclust indicates that Mclust behaves differently in tails of the four components located on the left and top of the plot. Finally, these plots show why similarity measures for our Lifting methods are found slightly lower than the CH index. Our Lifting methods divide one of the components in two sub-clusters while the CH index captures this component in one cluster.

For this data structure, different upper boundary choice (set at 15) does not have a considerable effect on the performance of the CH index, H index, KL index and Sil statistic, and Appendix A includes the tabulated similarity scores for different CVIs.

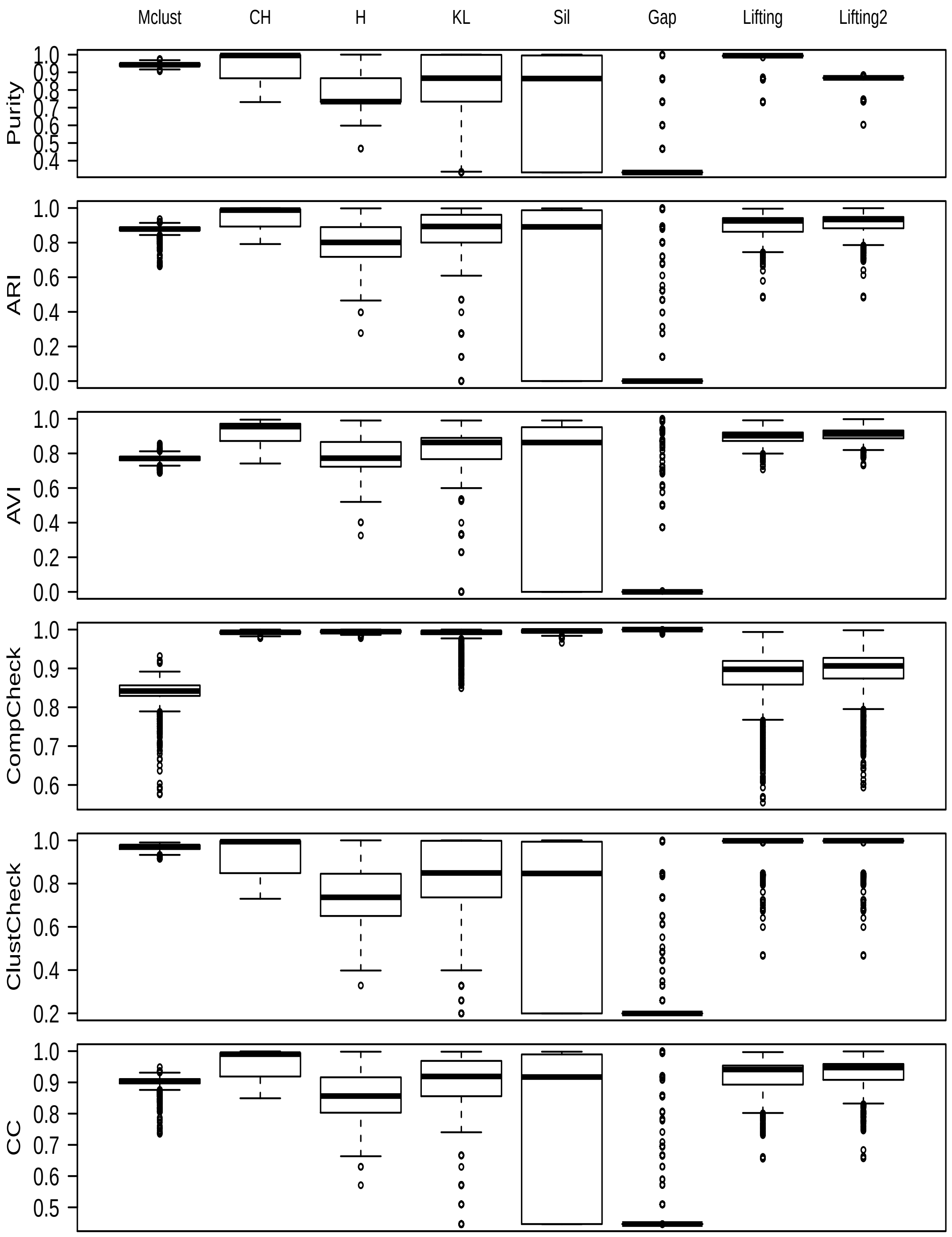


Figure 5.14: Box plot: the comparison of CVIs for the six-component non-normally distributed data.

Index	Mclust	CH	H	KL	Sil	Gap	Lifting	Lifting2
N	9	9	7	13	6	2	22	21
Purity	0.943	0.915	0.767	0.810	0.706	0.404	0.976	0.856
ARI	0.873	0.931	0.767	0.774	0.611	0.118	0.899	0.909
AVI	0.770	0.908	0.763	0.746	0.596	0.161	0.893	0.906
CompCheck	0.836	0.993	0.994	0.981	0.995	1.000	0.874	0.888
ClustCheck	0.968	0.910	0.722	0.783	0.658	0.273	0.974	0.974
CC	0.899	0.949	0.841	0.858	0.774	0.505	0.921	0.929

Table 5.11: The comparison of CVIs for the six-component non-normally distributed data. First row is for the average number of clusters, and others are for the average similarity scores.

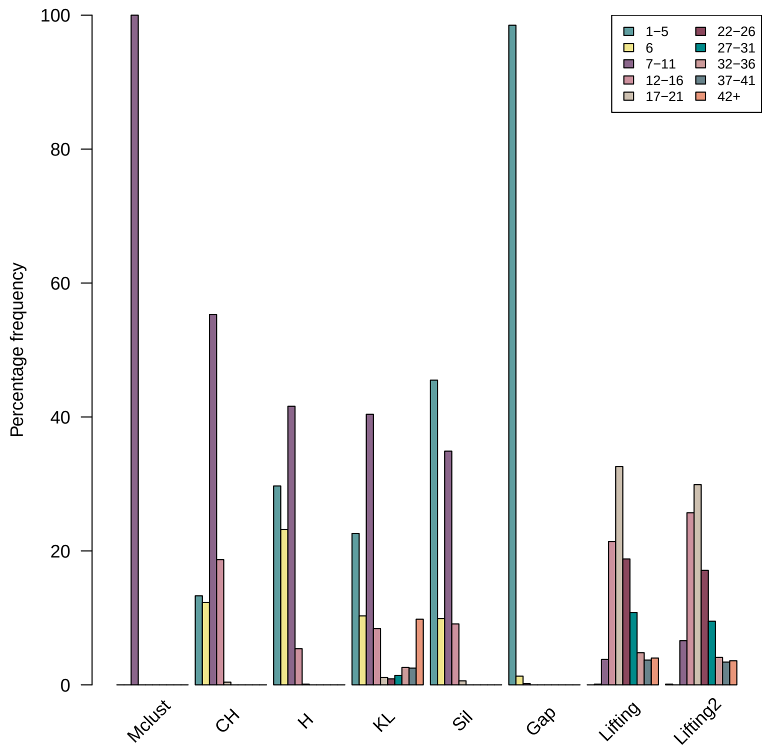


Figure 5.15: Bar chart: the comparison of number of clusters for the six non-normally distributed components data. Legends illustrate the number of cluster range. Vertical axis shows the percentage of replicates, and the horizontal axis is for the different CVIs.

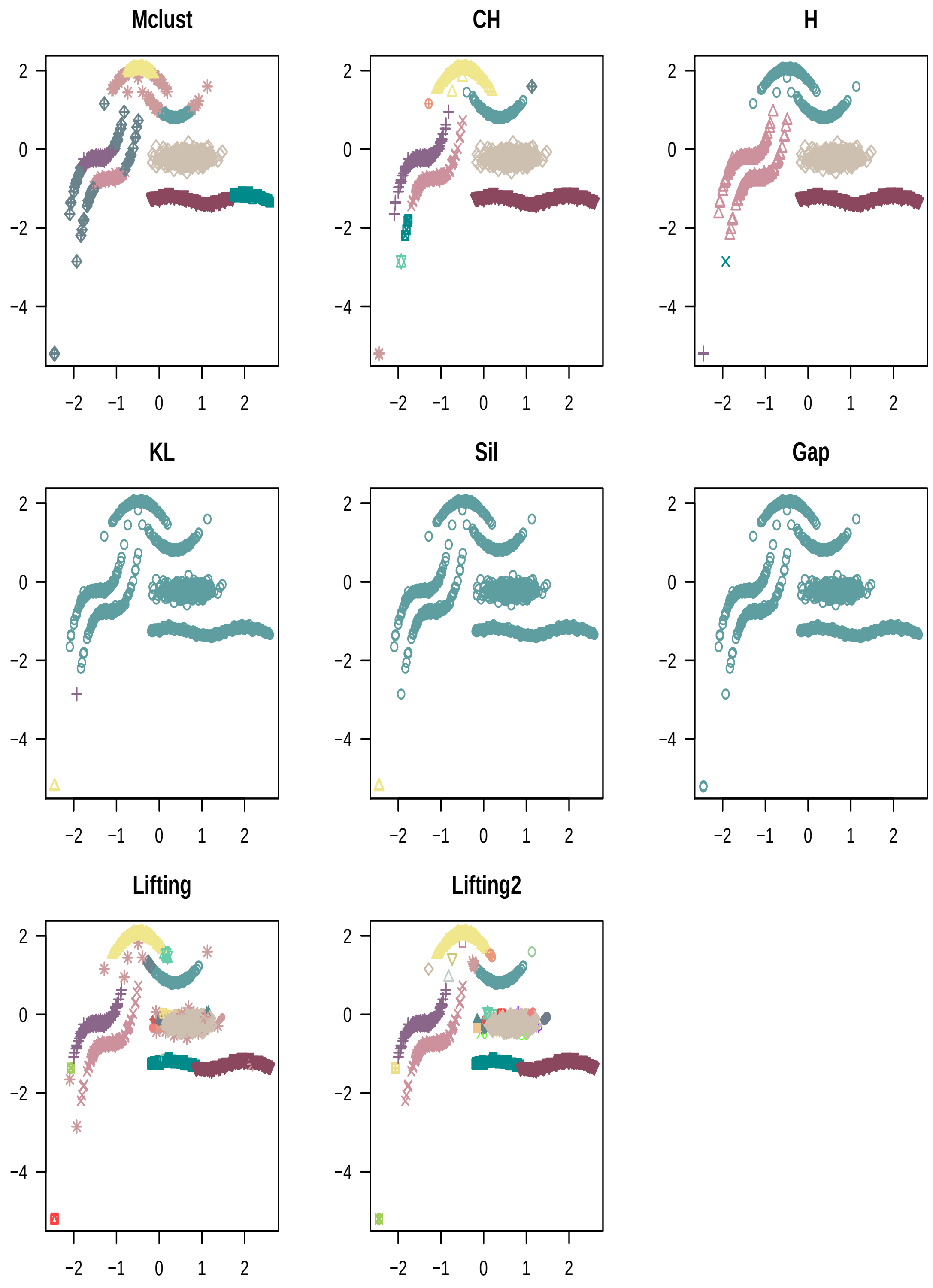


Figure 5.16: A single realisation of the six-component non-normally distributed data. Scatter plots illustrate the clustering solution chosen by various methods.

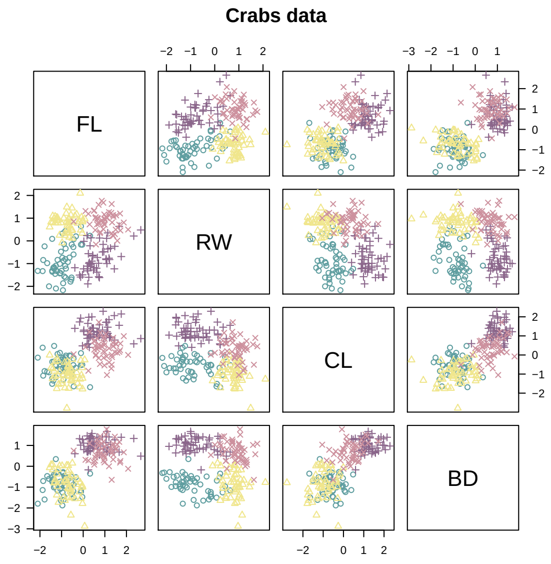


Figure 5.17: Pairs plots of scaled morphological features of crabs data.

5.6 Real data example

We also apply our method to a real data set, and we repeat the comparison study done in the previous section. The crabs data set (Campbell & Mahon, 1974) is used, and it is available in the **MASS** package in **R** (Venables & Ripley, 2002). It includes five morphological measurements of 200 crabs: frontal lobe size (FL), carapace length (CL), carapace width (CW), body depth (BD) and rear width (RW). There are four different groups of crabs formed by their colors (blue and orange) and gender (male and female) (size of 50). The data set is scaled by the size of the carapace of each crab ($\sqrt{CL \times CW}$), so we use four different scaled morphological measurements of crabs for clustering purposes: frontal lobe size, carapace length, body depth and rear width. The data set is standardized $x_k = \dfrac{x_k - \text{mean}(x_k)}{\text{sd}(x_k)}$, where $k \in \{1, 2, 3, 4\}$, and x_k is the kth measurement of the crabs data, x. Scatter plots of scaled morphological features of the crabs data are given in Figure 5.17. To be able to apply the Lifting algorithm, crabs are clustered using a hierarchical clustering algorithm with complete linkage. In this way, joined pairs and the edge lengths of joined pairs in each cluster in each agglomeration step are obtained. The final feature we need to have is the compactness value for each internal node which can be calculated using Equation (5.15). When the Lifting algorithm is applied to the crabs data, the tree in Figure 5.18 is built. Some internal nodes are labelled with the denoised detail coefficients. These nodes with all the internal nodes below these nodes are denoised as zero or less than zero. This means we can treat all the nodes below these labelled nodes as one cluster, so we find where we cut the tree. Thus, our Lifting method finds eight clusters for the crabs data where one cluster is for the outliers. A comparative table for each CVI

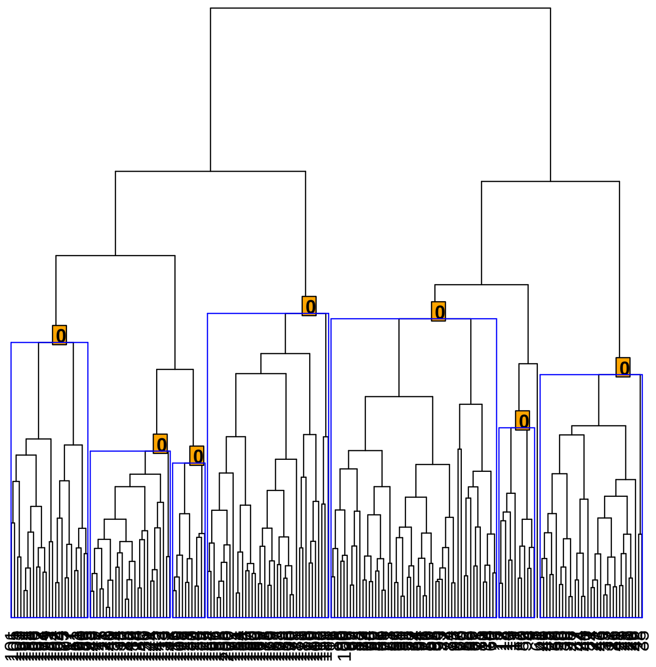


Figure 5.18: Clustering crabs data by Lifting. Labels in square boxes in internal nodes are the denoised detail coefficients.

Index	Mclust	CH	H	KL	Sil	Gap	Lifting	Lifting2
N	4	2	4	9	2	5	8	7
Purity	0.900	0.500	0.825	0.890	0.500	0.850	0.890	0.688
ARI	0.760	0.486	0.619	0.573	0.486	0.620	0.593	0.597
AVI	0.800	0.640	0.695	0.645	0.640	0.704	0.673	0.678
CompCheck	0.838	0.990	0.746	0.519	0.990	0.621	0.571	0.577
ClustCheck	0.803	0.490	0.689	0.874	0.490	0.751	0.825	0.825
CC	0.820	0.696	0.717	0.673	0.696	0.710	0.686	0.690

Table 5.12: The comparison of CVIs for crabs data. First row is for the number of clusters, and others are for the similarity scores.

in terms of the external validity scores is also generated and given in Table 5.12. The Sil statistic and CH index find the same partitioning, and their performance are poor in terms of the external validity scores. The partition done by Mclust has the highest similarity with true components, and the performance of the Gap statistic follows Mclust. All the CVIs except the CH index and Sil statistic find slightly different partitioning for the crabs data. We can see the partitioning done by each CVI from the scatter plots generated and demonstrated in Figure 5.19. The one titled as "CH" is for the CH index and Sil statistic.

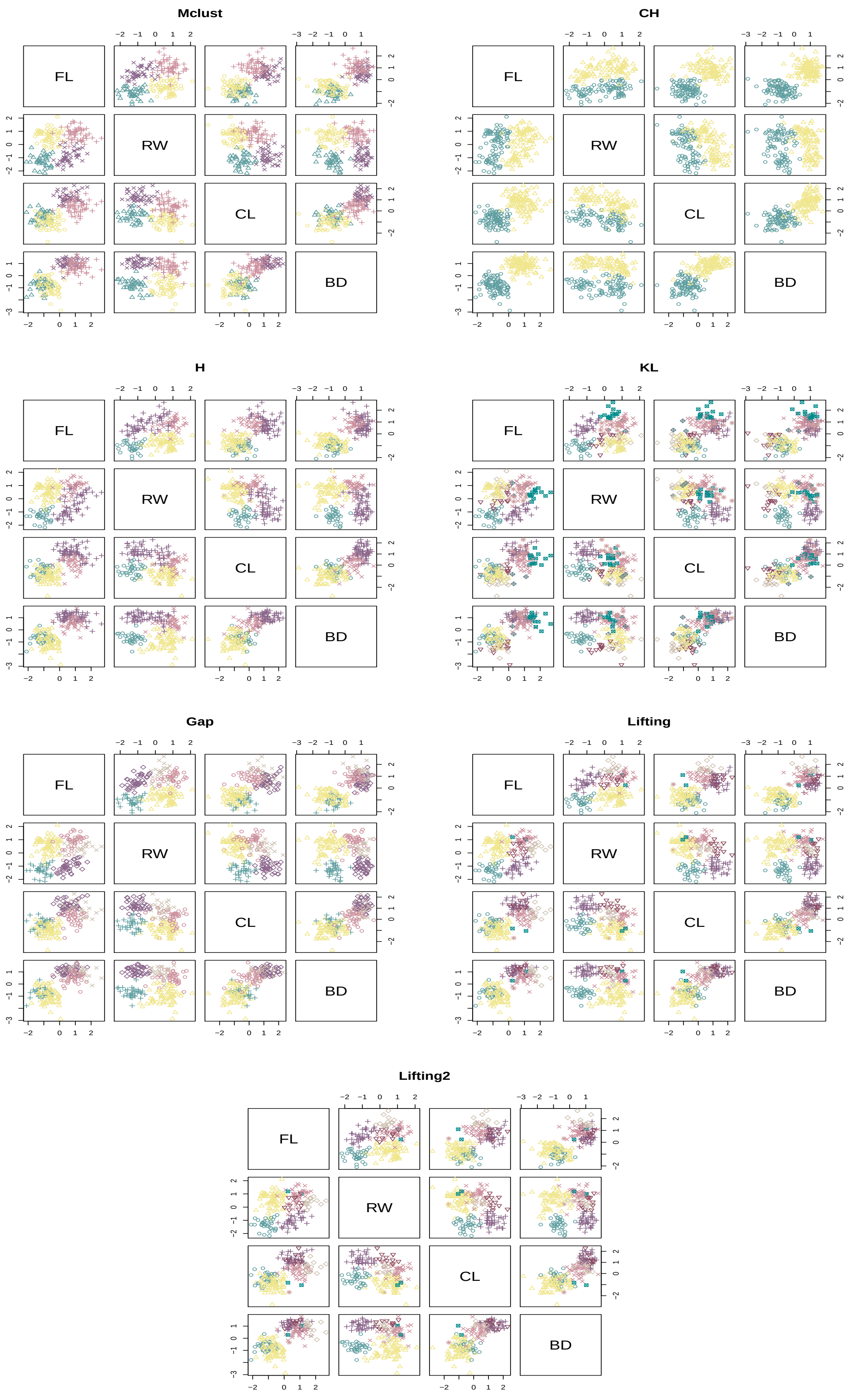


Figure 5.19: Pairs plots for different internal indices for crabs data. "CH": for the both CH index and Sil statistic.

5.7 Summary

In this chapter, we propose a new method which is based on LOCAAT algorithm to decide where to cut a tree built by hierarchical clustering. The method we proposed in Section 5.4 is computationally efficient, and it makes no parametric assumption. Since the tree is built agglomeratively, the distances higher on the tree have some uncertainties. Thus, we can denoise the tree using the LOCAAT algorithm. To be able to apply this idea to hierarchical clustering, we create compactness for each node which is the mean Euclidean distance of each data point from the centroid of their cluster. Hence, if we denoise the lifted compactness values (detail coefficients) for the nodes located in the higher part of the tree with the denoised detail coefficients of all the nodes below those nodes less than or equal to a threshold, we can treat them as one cluster. For now, we can manually pick a small number as the threshold, but we will discuss how we can pick a threshold automatically later in Chapter 6.

A simulation study in Section 5.5 shows that the performance of our method is always high. If a data set includes normally distributed components, Mclust always shows the highest performance. As Mclust is a model-based clustering algorithm, this result was expected. For non-Gaussian data sets, we could also construct an equivalent model-based clustering algorithm if we knew the distribution of the data set. However, new methods would need to be constructed for each data set, and these methods would be sensitive to mis-specification of the underlying distributions.

In the simulation study, we use single linkage method, and we observe that if a data set consists of normally distributed components with large variance, all internal cluster validity indices (CVIs) struggle to partition the data. However, if we build the tree with complete linkage, their performances are higher. Some results are added in Appendix A after building the tree with complete linkage. Mclust clusters almost all data points correctly, but also our Lifting method has a high performance in this case. If we check the scatter plots for just one replicate, our Lifting method found some outliers and some small groups located at outer part of the true components.

When we apply our algorithm to the circle data, we point out that Mclust shows poor performance in this data structure. Other indices perform slightly better than our method. The scatter plot for one replicate illustrates that other indices capture two components, and they just divide the outer circle into small clusters. However, our Lifting method divides the inner and outer circle components into two clusters. This decreases its performance because some data points gather together, and their distance to the bigger part of the component is high. Thus, our Lifting method captures them in separate clusters.

When we check the final simulation study with the six-component data which has the unique shape, the CH index shows the best performance. The performance of our Lifting method is close to the CH index. If we check the box plot comparison for the number of clusters found by different CVIs, the variation is higher in the CH index. There is a lower variation in our Lifting method, and the reason of the variation in our method is the constant threshold we use for all replicates. If we check scatter plots just for one replicate of the simulated data, we see why the performance of our Lifting method is lower than the CH index. Our Lifting method divides one of the components into two sub-clusters.

Hence, if we have a normal shaped data or well separated data, all indices perform well, but if we have a unique shape data or components that are closer to each other, some of the indices do not capture true components. For the real data set, the performance of each index is different than each other.

We also need to note that the available **R** packages for the cluster validation indices summarized in Section 5.3.1 need to have an upper boundary for the number of clusters. If we do not set an upper boundary, the calculation will take more time especially for the Gap statistic. Thus, the upper boundary should be carefully picked since it has a serious effect on results. However, the Lifting algorithm does not have a boundary. It picks the best partition over all possible partitioning to be done.

Next to the feature of capturing the true components efficiently, the Lifting also finds the exact height of each cluster. Other indices just return the number of clusters, then we need to see the dendrogram of the data to find where we need to cut the tree. However, the Lifting has a multiple cutting point where each point is for different clusters, so we cut the tree from the exact height of each cluster.

Chapter 6

Generalisation of the threshold choice

6.1 Introduction

In Chapter 5, we discussed how we can apply the lifting algorithm to find the number of clusters for multidimensional data sets. The aim of our Lifting algorithm is to find the number of clusters and where exactly clustering happens in a dendrogram automatically, but the algorithm proposed in Chapter 5 is only semi-automatic because we need to manually pick a threshold to allow small departures from the centroid of each cluster. In this chapter, we discuss how our algorithm can pick this threshold automatically. In this way, we reach our goal: automatically finding clusters by lifting. In addition, we also discuss how to apply our Lifting algorithm with a generalized threshold choice using non-decimated lifting (NLT) algorithm, described in Section 3.6.

In this chapter, we start with an explanation of the updated version of our Lifting method proposed in Chapter 5. In Section 6.2, we detail the updated version of our Lifting method to detect the number of clusters automatically. Then we introduce how to modify our Lifting algorithm using NLT algorithm in Section 6.3. The updated results for the artificial data sets generated in Section 5.5 and the real data set used in Section 5.6 follow in Sections 6.4 and 6.5, respectively. We complete the chapter with a discussion of findings in Section 6.6.

6.2 A method of picking a threshold

Lifting is a decompisition method proposed by Sweldens (1998), and a detailed summary was given in Chapter 3. We proposed a new method to identify the number of clusters automatically by lifting in Chapter 5. Our Lifting method is used to denoise the compactness value for each node in a dendrogram, where we define the average distance from the centroid of a cluster as compactness, γ_i, defined in Equation (5.15). In Chapter 5, if all denoised detail coefficients of a node and all nodes below it are less than or equal to zero, we assume that all objects below this node come from the same cluster. However, some departures from the centroid can be observed. In this case, we can pick a small threshold manually, so if the denoised detail coefficients of a node and all its child nodes are less than or equal to the chosen threshold, the objects below it create one cluster. This threshold can be regarded as a tuning parameter. So far the threshold is picked manually. If we find a way to allow our Lifting method to pick the threshold itself, we will no longer require manual intervention in the algorithm.

We can pick this threshold using some of the thresholding methods we summarized in Section 2.10. One of them is the universal threshold; details can be found in Section 2.10.2. If we limit the divergence between close neighbours to the universal threshold amount, we will allow a weighted standard deviation amount of divergence between close neighbours. However, denoised detail coefficients are a sparse set, so there is no enough information to estimate the variance. We propose that we can find the place of the clusters using denoised compactness values instead of using denoised detail coefficients. This means that any departure from the centroid of a cluster less than or equal to λ is treated as noise. We then regard any cluster which has a denoised compactness less than or equal to λ, and whose sub-clusters also have denoised compactness less than or equal to λ, as a cluster in our final solution. Thus, after denoising the detail coefficients, we can apply the reverse lifting transform, explained in Section 3.3.2 to obtain denoised compactness for each cluster. The final step is to find which clusters have denoised compactness value less than or equal to a threshold, λ, which can be automatically picked by limiting departure from the centroid to the universal threshold amount, so we allow a weighted standard deviation amount of departure from the centroid. We call this version of our algorithm as 'ALifting' ('A' stands for automatic choice of the threshold). The universal threshold, λ^u, and the estimate of the standard deviation, s, are defined in Equations (2.39) and (2.40), respectively. Hence, we threshold the denoised compactness values assuming that all of them are coming from the same resolution level. The estimator of the standard deviation is defined as

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \left[\hat{\gamma}_i - \bar{\hat{\gamma}}\right]^2}, \tag{6.1}$$

where $\hat{\gamma}_i$ are the denoised compactness values, $\bar{\hat{\gamma}}$ is the average of the denoised compactness values, and n is the number of nodes including the leaves in a dendrogram. Even though we compute s using all the nodes in the tree, we know that $\hat{\gamma}_i$ are zero for leaves. Thus, we can rearrange Equation (6.1) by splitting the summation term into two parts. We know that the first $\frac{n+1}{2}$ entries of $\hat{\gamma}_i$ are for leaves, and the rest of them are for the internal nodes. In the light of this information, we can easily see that $\sum_{i=1}^{n} \hat{\gamma}_i = \sum_{i=\left(\frac{n+1}{2}+1\right)}^{n} \hat{\gamma}_i$, and $\bar{\hat{\gamma}} = \frac{1}{n}\sum_{i=1}^{n}\hat{\gamma}_i = \frac{1}{n}\sum_{i=\left(\frac{n+1}{2}+1\right)}^{n}\hat{\gamma}_i$. Hence, Equation (6.1) becomes

$$\begin{aligned}
s^2 &= \frac{1}{n-1}\left\{\sum_{i=1}^{\frac{n+1}{2}}\left[\hat{\gamma}_i - \bar{\hat{\gamma}}\right]^2 + \sum_{i=\left(\frac{n+1}{2}+1\right)}^{n}\left[\hat{\gamma}_i - \bar{\hat{\gamma}}\right]^2\right\} \\
&= \frac{1}{n-1}\left\{\sum_{i=1}^{\frac{n+1}{2}}\left[-\bar{\hat{\gamma}}\right]^2 + \sum_{i=\left(\frac{n+1}{2}+1\right)}^{n}\left[\hat{\gamma}_i - \bar{\hat{\gamma}}\right]^2\right\} \\
&= \frac{1}{n-1}\left\{\sum_{i=\left(\frac{n+1}{2}+1\right)}^{n}\hat{\gamma}_i^2 - n\bar{\hat{\gamma}}^2\right\} \\
s &= \sqrt{\frac{1}{n-1}\left\{\sum_{i=\left(\frac{n+1}{2}+1\right)}^{n}\hat{\gamma}_i^2 - n\bar{\hat{\gamma}}^2\right\}}. \qquad (6.2)
\end{aligned}$$

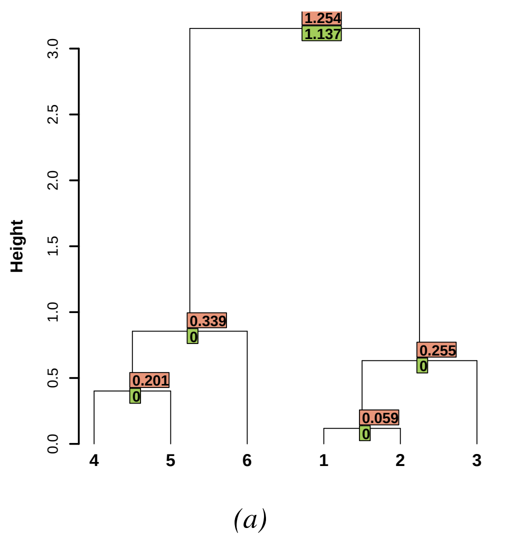
Each thresholded denoised compactness value, $\hat{\gamma}^*$, is the corresponding denoised compactness value:

$$\hat{\gamma}^* = \begin{cases} 0, & \text{if } \hat{\gamma} \leq \lambda^u \\ \hat{\gamma}, & \text{if } \hat{\gamma} > \lambda^u, \end{cases} \qquad (6.3)$$

where $\hat{\gamma}$ is the denoised compactness value.

We note that one option would be to exclude leaves from the estimate of the standard deviation, but this creates a large variation which leads us having few clusters. Thus, the probability of including different objects into the same clusters is high. In addition, our ALifting method even denoises the node values for leaves, so we need to include leaves to the estimate of the standard deviation to be consistent. We could also use the MAD, given in Equation (2.41), as an estimate of the standard deviation, but MAD is designed to ignore large values and hence is not a good estimate for the standard deviation in our method.

Example 6.2.1. *We continue Example 5.4.1. While the method we proposed in Chapter 5 finds clusters using the denoised detail coefficients, in this section, we use the denoised node values to find where exactly clustering happens. The dendrogram is given in Figure 6.1a, and each node is labelled with the compactness and denoised compactness values, which are* $\gamma = (0.059, 0.201, 0.255, 0.339, 1.254)$ *and* $\hat{\gamma} = (0.228, 0.278, 0.228, 0.278, 1.137)$*, respectively. These are the node values for internal nodes in agglomeration order. Each leaf can create one cluster alone, so we use each possible clustering scheme in a dendrogram to find the best representative clustering scheme for a data set. Thus, we threshold the denoised compactness values,* $\hat{\gamma} = (0, 0, 0, 0, 0, 0, 0.228, 0.278, 0.228, 0.278, 1.137)$*, where the first six entries of* $\hat{\gamma}$ *are the denoised compactness values for leaves. Since compactness is defined as the average*



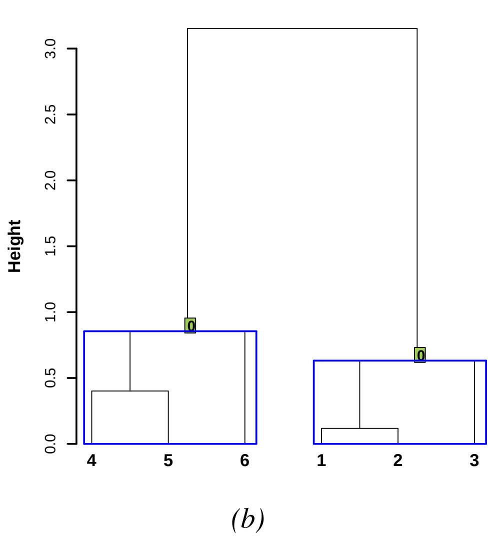


Figure 6.1: Illustration of the toy data in Table 5.1 with the results of our ALifting method. (a): the labelled dendrogram with the compactness (pink) and denoised compactness values (green). (b): clustering pattern found by our ALifting method, and labels for denoised compactness values.

distance from the centroid, both compactness and denoised compactness values for leaves are zero.

After finding $\hat{\gamma}$, we need to find the universal threshold, λ^u, for these denoised compactness values, so the λ^u is

$$\lambda^u = s\sqrt{2\log(n)} = 0.336 \times \sqrt{2\log(11)} = 0.737,$$

where $n = 11$, and s is

$$s = \sqrt{\frac{1}{10}\left\{\sum_{i=7}^{11}\hat{\gamma}_i^2 - 11\bar{\hat{\gamma}}^2\right\}} = 0.336,$$

where $\bar{\hat{\gamma}} = 0.195$. Thus, if any $\hat{\gamma}$ is less than or equal to λ^u, we will set them to zero. In this example, we find $\hat{\gamma}^* = (0,0,0,0,0,0,0,0,0,0,1.137)$ and the same clustering scheme found in Example 5.4.1. The dendrogram of the data is given in Figure 6.1b, and we label the clusters found by the denoised compactness values.

6.3 Automatic cluster detection by non-decimated lifting

Our proposed ALifting algorithm based on the LOCAAT algorithm detects where clustering occurs in a dendrogram, and we can also make a small modification on our algorithm and create a new algorithm based on the non-decimated lifting (NLT) algorithm described in Section 3.6.

The difference in the NLT algorithm is the order in which the points are lifted. Before starting the algorithm, we know which point to be lifted (path/trajectory) in each stage. Thus, the LOCAAT algorithm is applied using the known path which is one permutation of the nodes in the tree including leaves. This process is repeated P times using a different path for each repetition, where P is the number of paths given by the researcher. The NLT algorithm covered in Section 3.6 was for one dimensional data sets. To apply the NLT algorithm on tree-structured data sets, we can still use the same LOCAAT algorithm for multidimensional data sets discussed in Section 3.8 by simply changing the process of finding which node to be lifted. We can find one permutation of the nodes in the tree including leaves and set the last r entries of the path to be non-lifted nodes, where r is defined by the researcher. Then we can apply our ALifting algorithm proposed in Section 6.2, and we can repeat the algorithm P times using a different path each time. At the end of the NLT for the tree-structured data sets, we have P denoised compactness values for each node, and we have a different clustering results for each path. Thus, we need to pick the best representative of the clustering structure for the data set of interest within these P results, and we consider two methods to find the overall clustering result.

1. If any node is found as a cluster by our algorithm with the automatic choice of threshold in all P repetitions, we can regard that node as a cluster. However, this may result in few small clusters since we do not allow any variation which may be caused by the permutation stage of the NLT algorithm.

2. Another option is that we can compute the probability of being a cluster for each node in the tree. Suzuki & Shimodaira (2006) proposed an **R** package called **pvclust** which assigns a probability for each possible cluster on a tree built by hierarchical clustering. They take two different probabilities into account: the bootstrap probabilities (BP) and approximately unbiased (AU) probabilities. The BP is calculated building the tree many times over bootstrap sampled data and counting how many times that node appears in bootstrap sampling (Efron, 1979; Felsenstein, 1985a). Another option, AU, is calculated using multiscale bootstrapping developed by Efron et al. (1996), Shimodaira (2002) and Shimodaira (2004). In our NLT method, we compute the probability of being clustered using a similar procedure to BP.

   In our ALifting algorithm, if any node with all its daughter nodes have a compactness less than or equal to a threshold, λ^u, we consider that one possible cluster is located in this node. Assume that we have n nodes including the leaves and P paths. Then the clustering result of our algorithm for each path is

$$c_{ij} = \begin{cases} 0, & \text{if } \hat{\gamma}_{ij} \leq \lambda^u_j \\ 1, & \text{if } \hat{\gamma}_{ij} > \lambda^u_j, \end{cases}$$

where c_{ij} denotes the decision of having a cluster at node i in path j ($i = 1, \ldots, n$ and $j = 1, \ldots, P$), $\{\hat{\gamma}_{ij}\}$ are the denoised compactness values, and λ_j^u is the universal threshold for the path j. Our algorithm places one cluster at node i in path j if $c_{ij} = 0$. The probability of having a cluster at node i over all paths, p_i, is

$$p_i = 1 - \frac{1}{P}\sum_{j=1}^{P} c_{ij}. \tag{6.4}$$

If node i with all its daughter nodes have $p_. \geq \theta$, where $\theta \in [0, 1]$ is the chosen probability of acceptance defined by the researcher, there is a possible cluster at node i with probability p_i.

The choice of P has a significant role in our NLT algorithm. There are $n!$ different paths for the lifting order, so lower P values may increase the chance of placing a cluster at a node with a high probability, or they may increase the chance of neglecting a possible clustering scheme because of the low probability of placing a cluster in some nodes.

Example 6.3.1. *We continue Example 6.2.1 given in previous section to illustrate the working structure of our NLT algorithm. There are six data points in this toy data set, so leaves are labelled from one to six, and the internal nodes are labelled with their agglomeration order starting from seven. The compactness value for each node, $\{\gamma_i\}$, is lifted using our NLT algorithm for tree-structured data sets. For illustrative purposes, we set the number of paths, P, and the clustering probability, θ at* 10 *and* 0.5*, respectively. For each path, the denoised compactness values, $\{\hat{\gamma}_{ij}\}$, the universal threshold, $\{\lambda_j^u\}$ and the clustering decision indicator, $\{c_{ij}\}$, are found and given in Table 6.1. The final row of the table illustrates the probability of having a cluster at each node, $\{p_i\}$. The dendrogram of the toy data is illustrated in Figure 6.2a. Each node in this dendrogram is labelled with the clustering probability and the agglomeration order. The clustering probability for the root is* 0.4 *which is a low probability to set a cluster at this node. The clustering probability for the root is less than θ, so the color of this node is changed from green to red. Using the threshold $\theta = 0.5$, we find the same clustering pattern found in Example 6.2.1, where we applied our ALifting algorithm, and the possible clustering pattern found by our NLT algorithm is displayed in Figure 6.2b.*

Path	Results	Nodes											λ_j^u
		1	2	3	4	5	6	7	8	9	10	11	
	γ_i	0	0	0	0	0	0	0.059	0.201	0.255	0.339	1.254	
1	$\hat{\gamma}_{i1}$	0	0	0	0	0	0	0.716	0.217	0.241	0.255	1.117	0.802
	c_{i1}	0	0	0	0	0	0	0	0	0	0	1	
2	$\hat{\gamma}_{i2}$	0	0	0	0	0	0	0.268	0.426	0.386	0.226	0.965	0.663
	c_{i2}	0	0	0	0	0	0	0	0	0	0	1	
3	$\hat{\gamma}_{i3}$	0	0	0	0	0	0	0.212	0.801	0.200	0.743	0.519	0.690
	c_{i3}	0	0	0	0	0	0	0	1	0	1	0	
4	$\hat{\gamma}_{i4}$	0	0	0	0	0	0	0.836	0.243	0.783	0.290	0.525	0.719
	c_{i4}	0	0	0	0	0	0	1	0	1	0	0	
5	$\hat{\gamma}_{i5}$	0	0	0	0	0	0	0.597	0.435	0.580	0.457	0.516	0.600
	c_{i5}	0	0	0	0	0	0	0	0	0	0	0	
6	$\hat{\gamma}_{i6}$	0	0	0	0	0	0	0.371	0.621	0.392	0.607	0.505	0.593
	c_{i6}	0	0	0	0	0	0	0	1	0	1	0	
7	$\hat{\gamma}_{i7}$	0	0	0	0	0	0	0.543	0.123	0.559	0.531	0.607	0.606
	c_{i7}	0	0	0	0	0	0	0	0	0	0	1	
8	$\hat{\gamma}_{i8}$	0	0	0	0	0	0	0.109	0.107	0.228	0.290	1.286	0.832
	c_{i8}	0	0	0	0	0	0	0	0	0	0	1	
9	$\hat{\gamma}_{i9}$	0	0	0	0	0	0	-0.064	0.195	0.124	0.413	1.260	0.842
	c_{i9}	0	0	0	0	0	0	0	0	0	0	1	
10	$\hat{\gamma}_{i10}$	0	0	0	0	0	0	0.074	0.139	0.272	0.258	1.242	0.804
	c_{i10}	0	0	0	0	0	0	0	0	0	0	1	
p_i		1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.8	0.9	0.8	0.4	

Table 6.1: Tabulated results of our NLT algorithm for the toy data in Table 5.1.

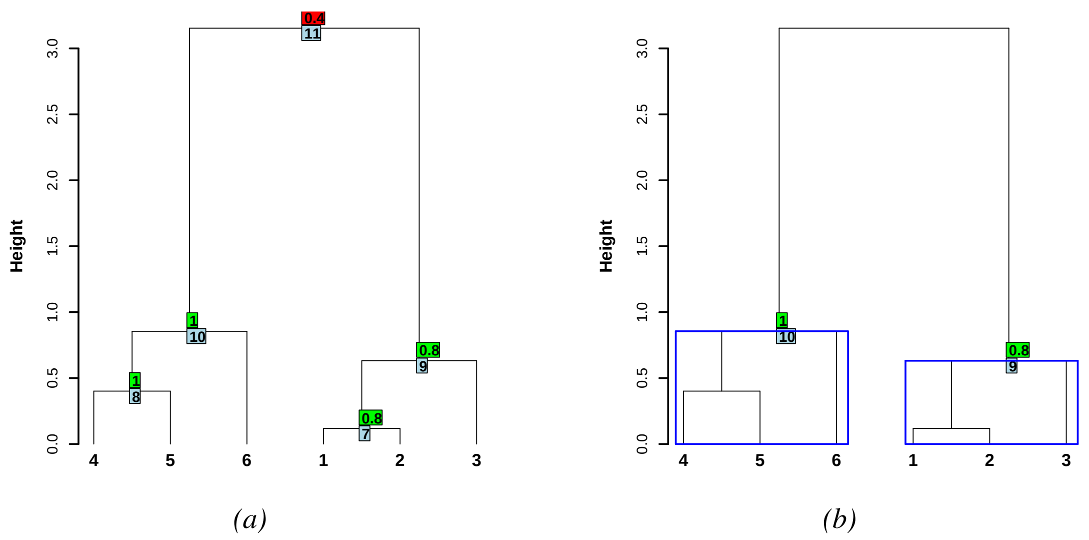


Figure 6.2: Illustration of the toy data in Table 5.1 with the results of our NLT algorithm. (a): the labelled dendrogram with the clustering probability (green or red) and the agglomeration order (blue). (b): clustering pattern found by our NLT algorithm.

6.4 Simulation study

The ALifting method is applied to the same data sets given in Section 5.5, and results are compared to the results discussed in Section 5.5. In Section 5.5, the threshold, λ, was picked manually, and we summarized the results of our algorithm by comparing with some other available internal cluster validity indices (CVIs) in the literature which were summarized in Section 5.3.1. In this section, the comparison study includes the CVI which showed the best performance for each data structure in Section 5.3.1, Mclust, Lifting and Lifting2. To be consistent with the notations we used in Section 5.3.1, Lifting and Lifting2 represent the results for the choice of λ used in that section. We also add the results for our Lifting and Lifting2 methods when we pick $\lambda = 0$, and we call these results ZLifting and ZLifting2, respectively, where 'Z' stands for the zero threshold, $\lambda = 0$. To explore the effect of the automatic choice of the threshold, the updated version of our Lifting and Lifting2 methods, labelled as ALifting and ALifting2, respectively, are included to the comparison study. The final method we compare in this study is our NLT algorithm, labelled as NLT, and we set the number of paths, P, to 100 and the clustering probability threshold, θ, to 0.5. Higher choice of P increases the time taken to analyze large data sets; because of that we pick a value of P which does not make a significant reduction in the computational efficiency. In addition, we have checked the performance of the methods on several data sets with various values of P up to 1000 for one replicate, and we notice that after $P = 100$ the probability of placing a cluster for each node is reasonably stable. Thus, the choice of $P = 100$ is a reasonable choice for the data structures we use in this study. We compare the performance of different methods in terms of different external scores explained in Section 5.3.3.

6.4.1 Five-component normally distributed data with low variance

In Section 5.5.1, we discussed this data structure in detail, and λ was fixed at 0.2 for this data structure. The highest performance within the CVIs was for the CH index, so we include the CH index results in the comparison study. Box plots comparing the different partitioning methods are shown in Figure 6.3. The performance of ZLifting and ZLifting2 are close to zero in terms of different external scores. Thus, these results support the idea of needing a threshold. When we compare our Lifting and ALifting methods, we observe that small variation occurred in partitioning the data by our Lifting method is almost zero in ALifting method, and also there is a slight increase on the proportion of capturing the true components by ALifting and ALifting2 compared to our Lifting and Lifting2 methods, respectively. In addition, our NLT algorithm behaves similar to ALifting algorithm by locating clusters in each run with high percentage (> 0.90). We

Index	Mclust	CH	ZLifting	ZLifting2	Lifting	Lifting2	ALifting	ALifting2	NLT
N	5	9	128	127	8	7	7	6	6
Purity	0.999	0.985	0.937	0.823	0.937	0.759	0.997	0.807	0.998
ARI	0.998	0.976	0.148	0.178	0.916	0.921	0.989	0.994	0.993
AVI	0.993	0.961	0.453	0.493	0.946	0.954	0.984	0.992	0.989
CompCheck	0.998	0.987	0.108	0.120	0.979	0.988	0.984	0.993	0.991
ClustCheck	0.998	0.978	0.783	1.000	0.914	0.914	0.998	0.998	0.998
CC	0.998	0.982	0.289	0.342	0.942	0.946	0.991	0.995	0.994

Table 6.2: Updated results of the comparison of CVIs for the five-component normally distributed data with low variance. First row is for the average number of clusters, and others are for the average similarity scores.

can also check the average of external scores for each CVI from Table 6.2. ZLifting partitions the data into many clusters; because of that its performance is low. On the other side, we see that ALifting and NLT methods capture the 99% of the true partitioning in terms of ARI which is the highest performance after Mclust. While we set $\lambda = 0.2$ in our Lifting method, Figure 6.4 shows that λ picked by our ALifting algorithm varies for each repetition, and the median choice for the λ is 0.33. Hence, this suggests a reason for the variation of the partition found by our Lifting algorithm.

When we check Figure 6.5, the variation of the number of clusters found by our Lifting method decreases with ALifting and NLT methods. Over 90% of 1000 repetition, ALifting finds the number of clusters within the range $[6-10]$. To have an idea what kind of partitioning each CVI and our methods do, we can check the scatter plot comparison for one realisation from Figure 6.6. Both Lifting and ALifting capture the main clusters, and both of them find some small outlier clusters (having two objects in each). Our NLT algorithm finds the same partitioning with ALifting algorithm, and we also provide a dendrogram of the data set including the clustering pattern found for one repetition by our NLT algorithm in Figure 6.7. The top part of the dendrogram is labelled with clustering probabilities and the clustering pattern found for $\theta = 0.5$. Its clearly seen that our NLT algorithm places the clusters with high probability, and it is the only possible pattern the algorithm can pick.

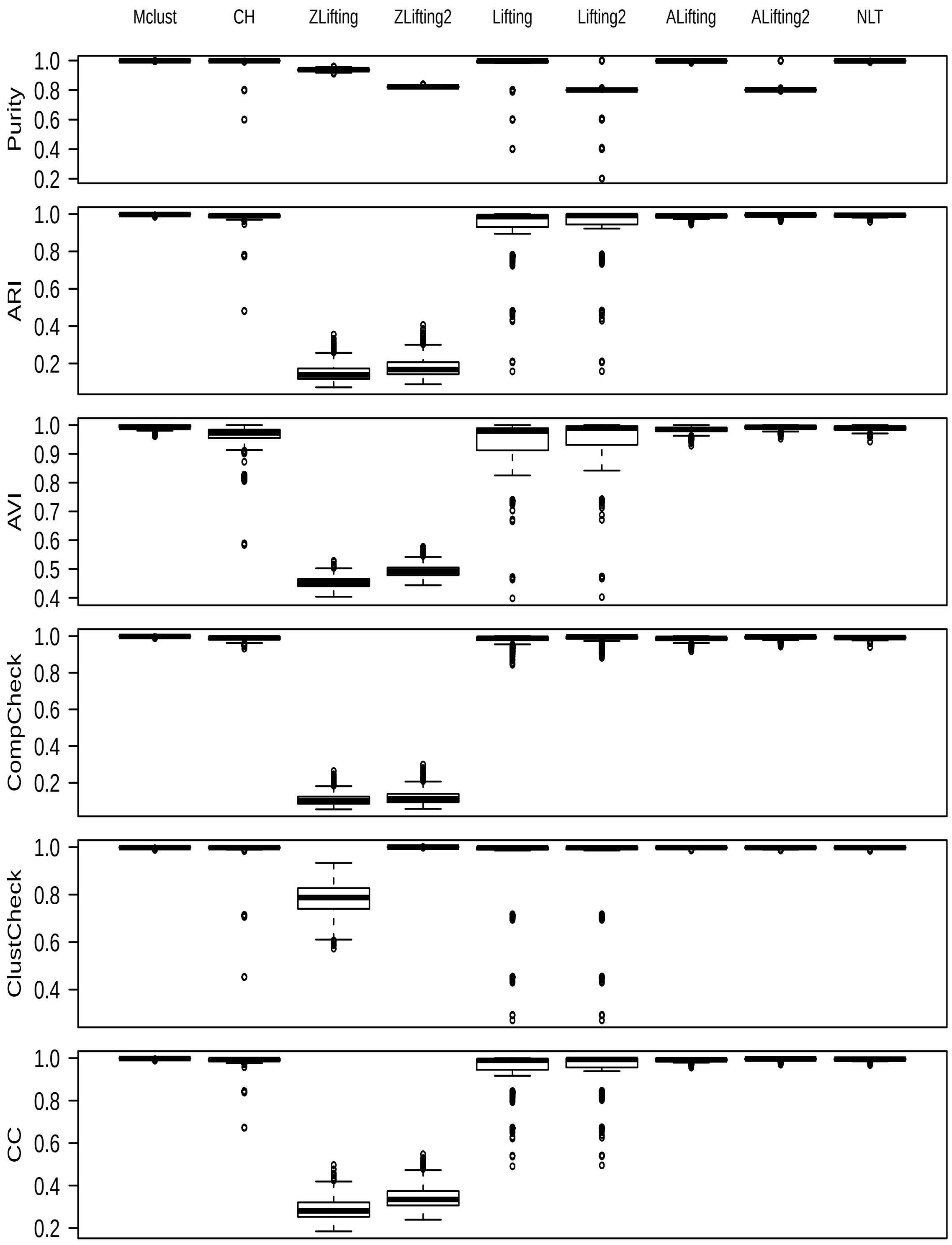


Figure 6.3: Updated box plot: the comparison of CVIs for the five-component normally distributed data with low variance.

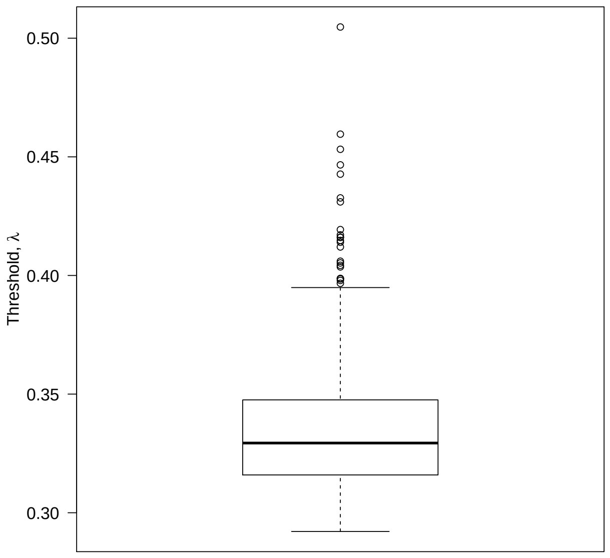


Figure 6.4: The variation of λ for the five-component normally distributed data with low variance.

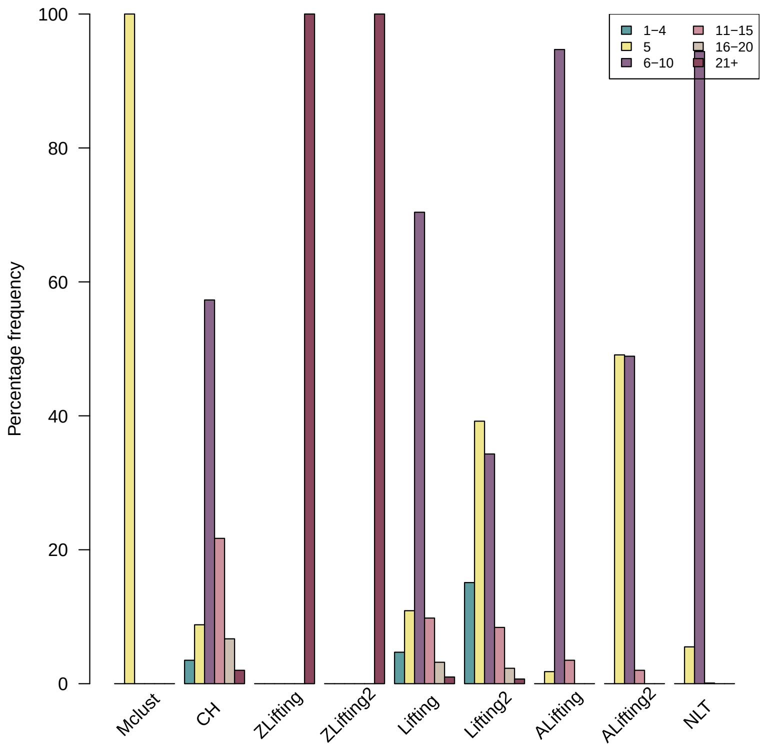


Figure 6.5: Updated bar chart: the comparison of number of clusters for the five-component normally distributed data with low variance. Legends illustrate the number of cluster range. Vertical axis shows the percentage of replicates, and the horizontal axis is for the different CVIs.

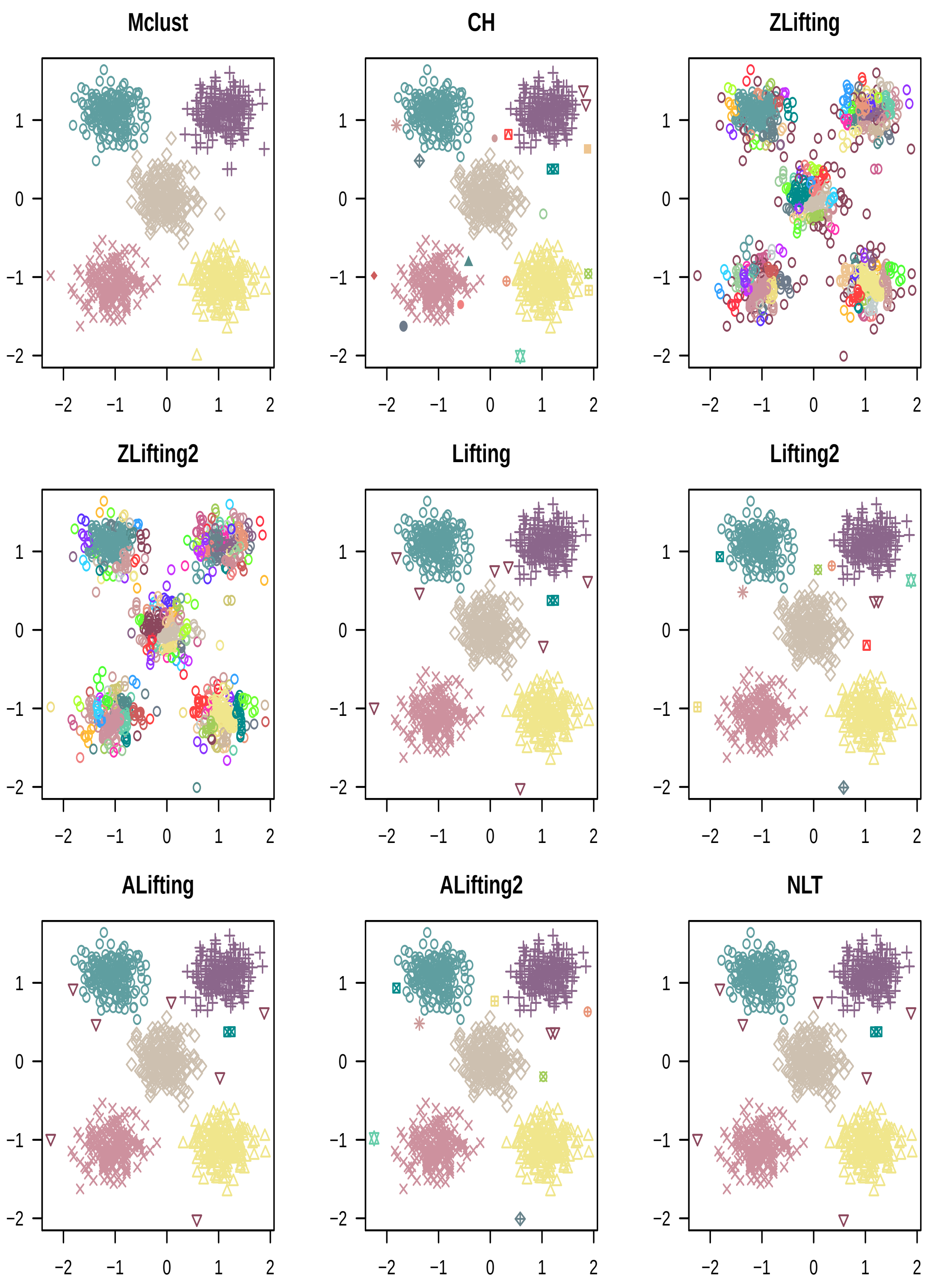


Figure 6.6: Updated results for a single realisation of the five-component normally distributed data with low variance. Scatter plots illustrate the clustering solution chosen by different methods.

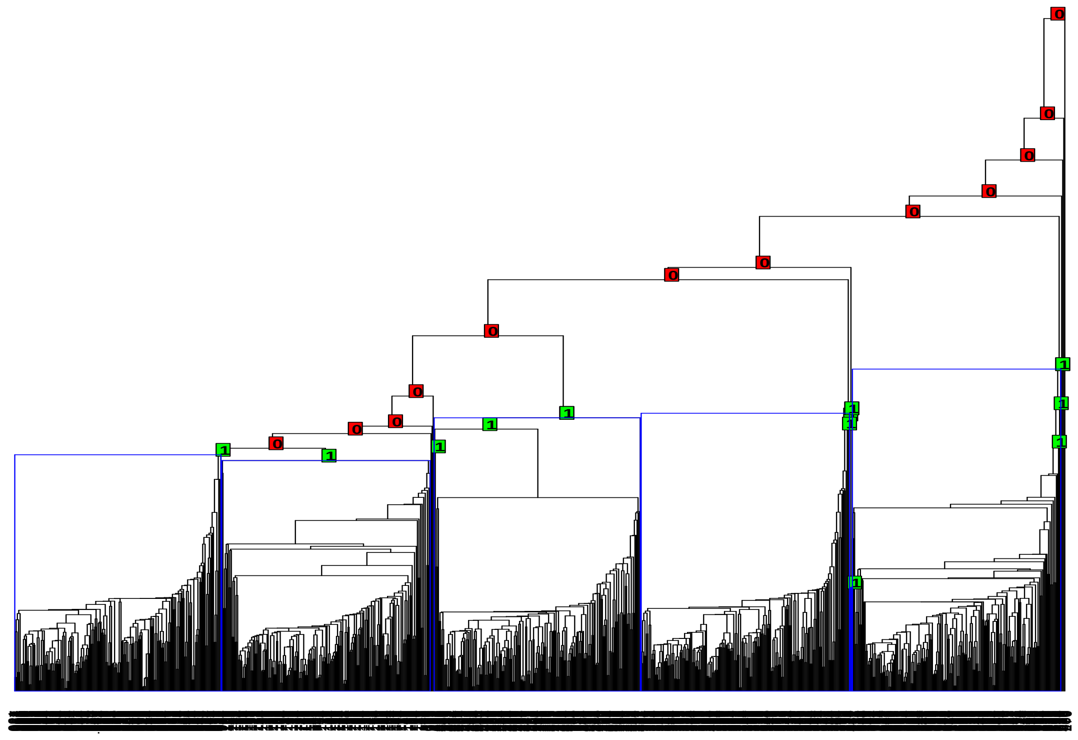


Figure 6.7: The dendrogram of our NLT algorithm for one realisation of the five-component normally distributed data with low variance. Internal nodes are labelled with the clustering probabilities colored with green, $p \geq \theta$, or red, $p < \theta$.

Index	Mclust	CH	ZLifting	ZLifting2	Lifting	Lifting2	ALifting	ALifting2	NLT
N	5	38	127	126	19	18	21	20	21
Purity	0.981	0.724	0.929	0.809	0.750	0.627	0.925	0.762	0.936
ARI	0.953	0.588	0.148	0.174	0.629	0.646	0.831	0.856	0.842
AVI	0.879	0.544	0.445	0.480	0.690	0.719	0.818	0.853	0.823
CompCheck	0.962	0.876	0.106	0.117	0.856	0.891	0.835	0.873	0.835
ClustCheck	0.962	0.612	0.809	0.991	0.678	0.679	0.902	0.904	0.918
CC	0.962	0.718	0.291	0.337	0.744	0.760	0.866	0.887	0.875

Table 6.3: Updated results of the comparison of CVIs for the five-component normally distributed data with larger variation. First row is for the average number of clusters, and others are for the average similarity scores.

6.4.2 Five-component normally distributed data with larger variation

The detailed discussion for different CVIs and our Lifting method for this data structure can be found in Section 5.5.2. The threshold, λ, was fixed at 0.3 for our Lifting method. The highest performance within the CVIs for this data structure was also for the CH index, so we include the results of the CH index in the comparison study in this section. Box plots comparing the different partitioning methods are shown in Figure 6.8. ZLifting and ZLifting2 also show poor performance for this data structure. While there is some variation in terms of capturing the true components by ALifting method, this variation range is narrower than our Lifting method. This small variation is disappeared with our NLT algorithm. We also observe a considerable increase on the proportion of capturing the true components by ALifting and ALifting2 compared to Lifting and Lifting2 methods, respectively. When we check the average of external scores for each CVI from Table 6.3, we see the increase on ALifting and NLT methods, and they capture over 80% of the true partitioning in terms of ARI which is the highest performance after Mclust for this data structure. We can also see the variation of λ picked by ALifting algorithm for each repetition in Figure 6.9. While we set $\lambda = 0.3$ in our Lifting method, Figure 6.9 shows that λ varies around 0.67. Our choice of λ for our Lifting method was much smaller than the ones algorithm picked, so the variation of the number of clusters found by our Lifting method is explained in this way.

When we check Figure 6.10, the variation of the number of clusters found by ALifting or NLT is similar to our Lifting method, but as we discuss above, their performances are higher than our Lifting method. When we compare the partitioning done by different CVIs and our methods using scatter plots for one realisation from Figure 6.11, Lifting, ALifting and NLT capture the main clusters, and they find some outliers and some small clusters. Next to the scatter plot, we also present the dendrogram of the data set including

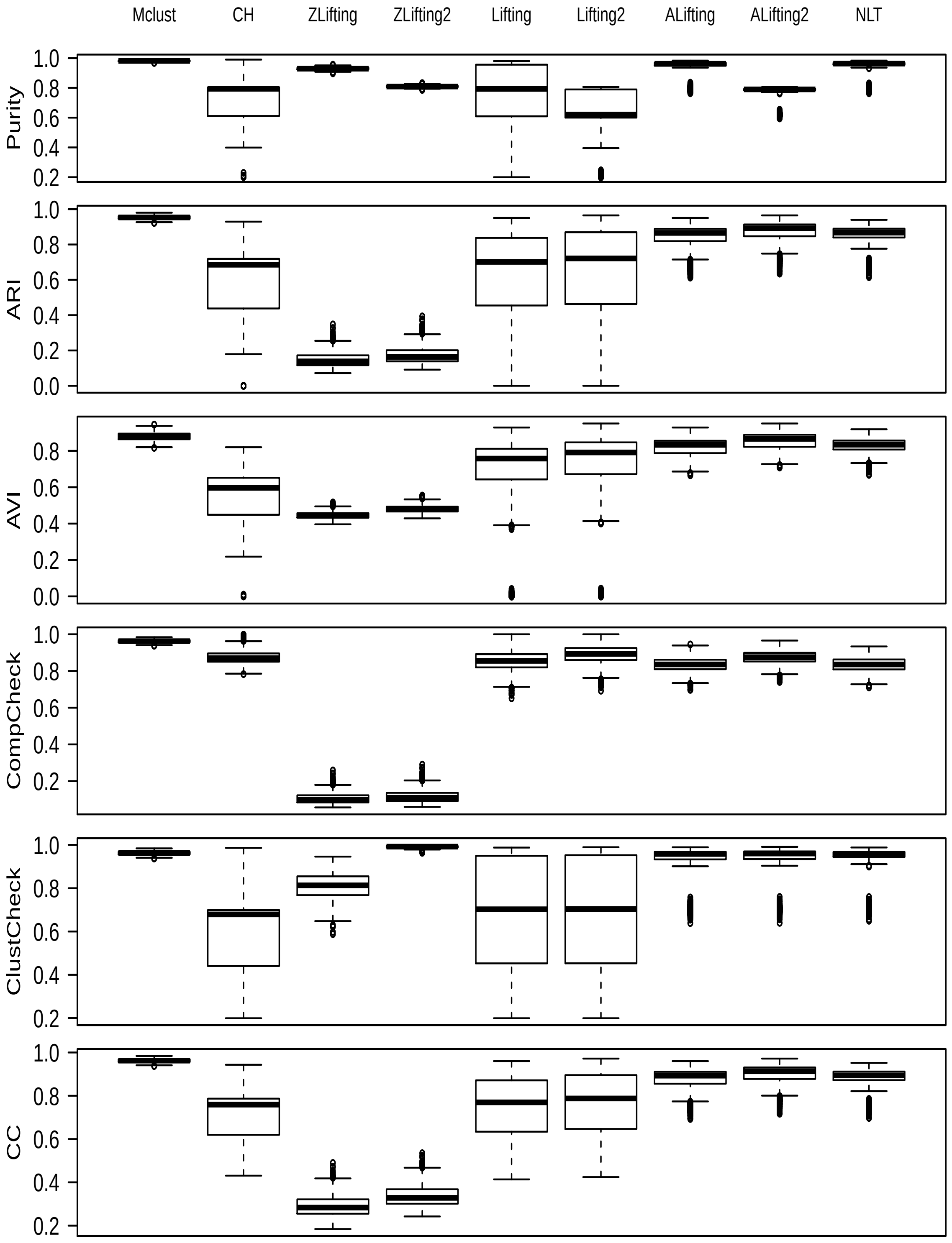


Figure 6.8: Updated box plot: the comparison of CVIs for the five-component normally distributed data with larger variation.

the clustering pattern found for one repetition by our NLT algorithm in Figure 6.12; it also places the clusters with high probability for this data structure.

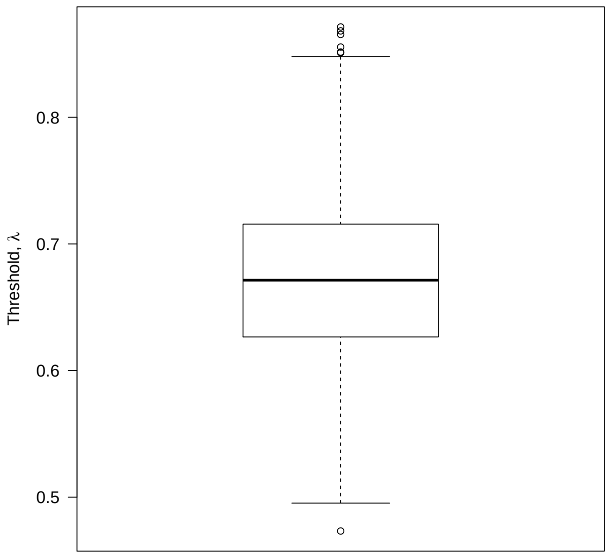


Figure 6.9: The variation of λ for the five-component normally distributed data with larger variation.

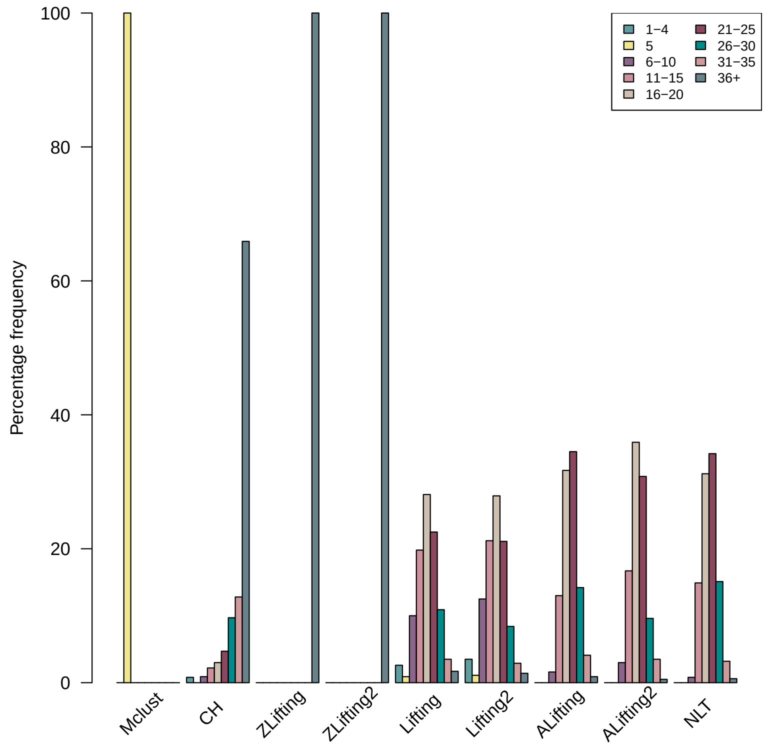


Figure 6.10: Updated bar chart: the comparison of number of clusters for the five-component normally distributed data with larger variation. Legends illustrate the number of cluster range. Vertical axis shows the percentage of replicates, and the horizontal axis is for the different CVIs.

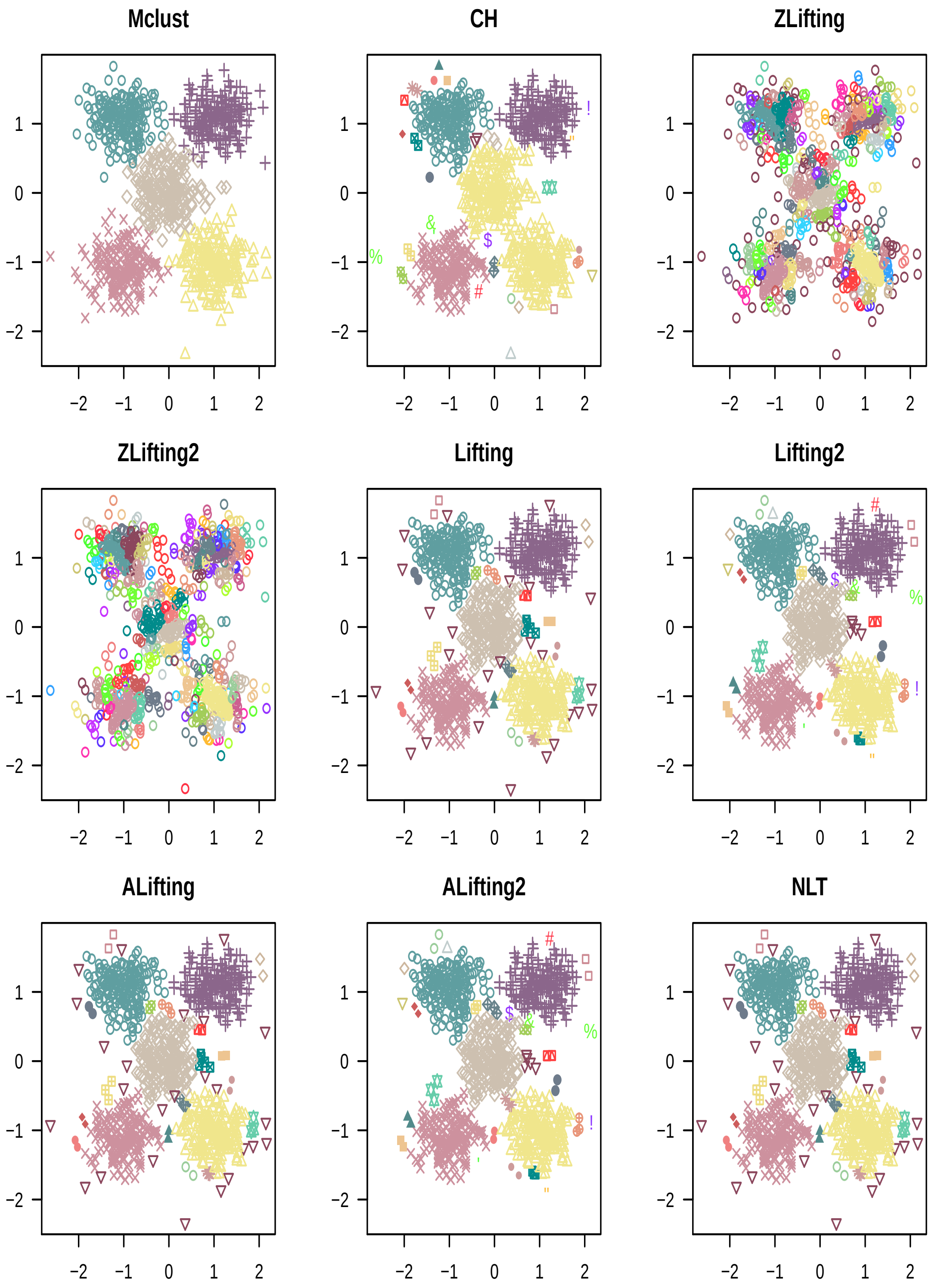


Figure 6.11: Updated results for a single realisation of the five-component normally distributed data with larger variation. Scatter plots illustrate the clustering solution chosen by different methods.

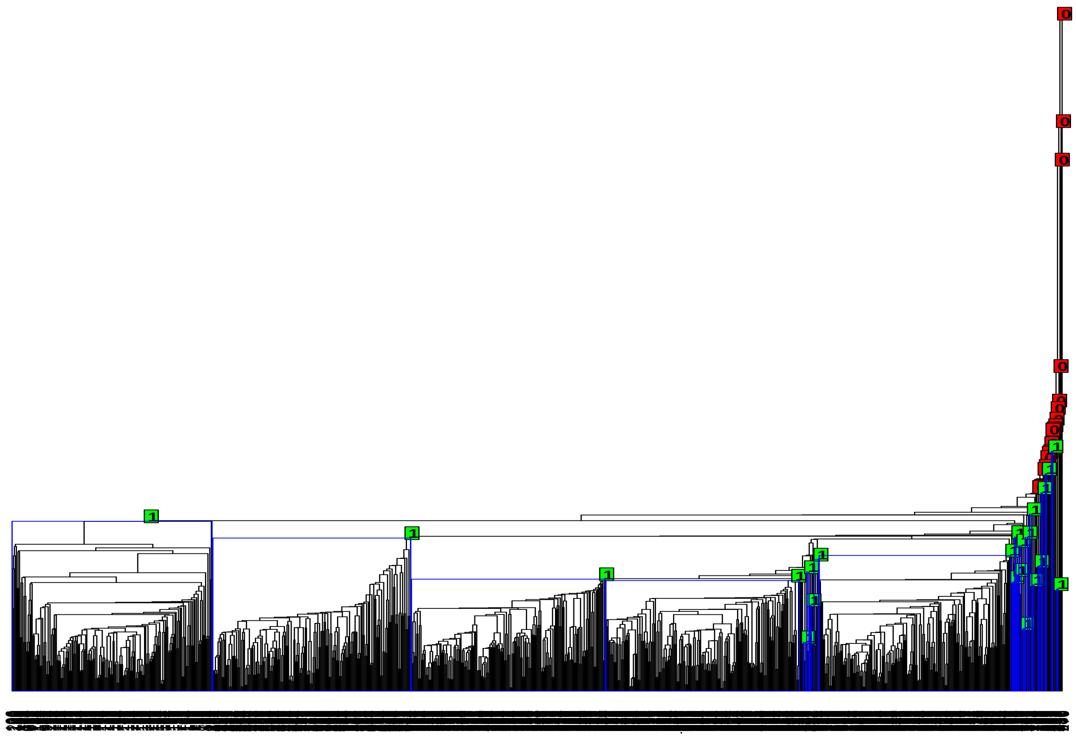


Figure 6.12: The dendrogram of our NLT algorithm for one realisation of the five-component normally distributed data with larger variation. Internal nodes are labelled with the clustering probabilities colored with green, $p \geq \theta$, or red, $p < \theta$.

Index	Mclust	H	ZLifting	ZLifting2	Lifting	Lifting2	ALifting	ALifting2	NLT
N	9	6	142	141	4	4	20	20	20
Purity	0.894	1.000	0.978	0.450	0.797	0.789	1.000	0.821	1.000
ARI	0.280	0.959	0.033	0.035	0.458	0.458	0.316	0.316	0.315
AVI	0.366	0.882	0.292	0.301	0.543	0.543	0.568	0.569	0.567
CompCheck	0.293	0.952	0.030	0.030	0.888	0.888	0.280	0.280	0.279
ClustCheck	0.832	1.000	0.911	1.000	0.684	0.684	1.000	1.000	1.000
CC	0.494	0.975	0.164	0.172	0.758	0.758	0.529	0.529	0.528

Table 6.4: Updated results of the comparison of CVIs for the three-component concentric circle data. First row is for the average number of clusters, and others are for the average similarity scores.

6.4.3 Three-component concentric circle data

The details of the circle data are discussed in detail in Section 5.5.3, and the threshold, λ, was fixed at 0.78. The highest performance within the CVIs for this data structure was for the H index, so we include the H index results in the comparison study in this section. Box plots comparing the different partitioning methods are shown in Figure 6.13. ZLifting method finds many small clusters as expected, so allowing some departures from the centroid may help us to increase the performance of our algorithm. Box plot comparison illustrates that the wide range of variation occurred in partitioning the data by our Lifting method is almost zero in ALifting and our NLT methods, but their performances are lower than our Lifting method in terms of ARI. On the other hand, AVI tells that Lifting, ALifting and NLT methods show similar performance to capture the true components, and tabulated average of AVI scores in Table 6.4 support our discussion. All of the external scores we compare agree that the highest performance of capturing true components is shown by the H index. We can also see the variation of λ picked by the algorithm for each repetition in Figure 6.14. While we set $\lambda = 0.78$ in our Lifting method, Figure 6.14 shows that λ varies around 0.42. Our choice of λ for our Lifting method was much higher than the ones algorithm picked itself. Thus, ALifting algorithm underestimates the choice of λ for this data structure. Even though ALifting algorithm decreases the variation of the number of clusters found by our Lifting method, it finds a high number of clusters.

Figure 6.15 illustrates that ALifting and our NLT methods cluster the circle data over ten clusters in all 1000 repetitions. We can check one possible partitioning found by each method to visualize how these high number of clusters are located, so a scatter plot comparison is given in Figure 6.16. While our Lifting method separates the middle and outer circle components into couple of different clusters, ALifting and NLT divide these components into many small clusters. We also provide the labelled dendrogram showing the clustering scheme found by our NLT algorithm for one repetition in Figure 6.17;

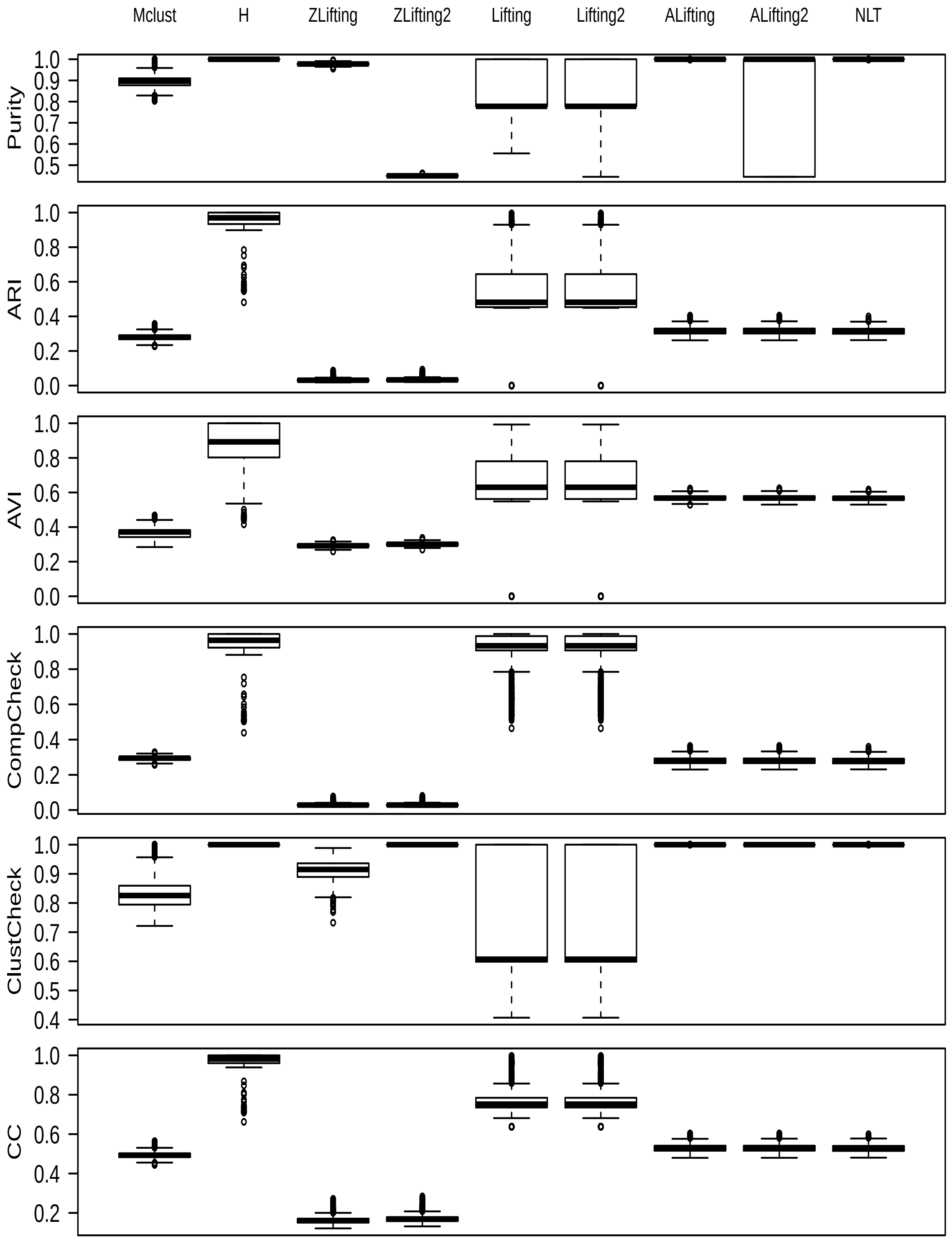


Figure 6.13: Updated box plot: the comparison of CVIs for the three-component concentric circle data.

the behaviour of the algorithm does not also change for this data structure. It places the possible clusters with a high probability. Overall, the H index still captures the true components with the highest performance (AVI≈ 0.88).

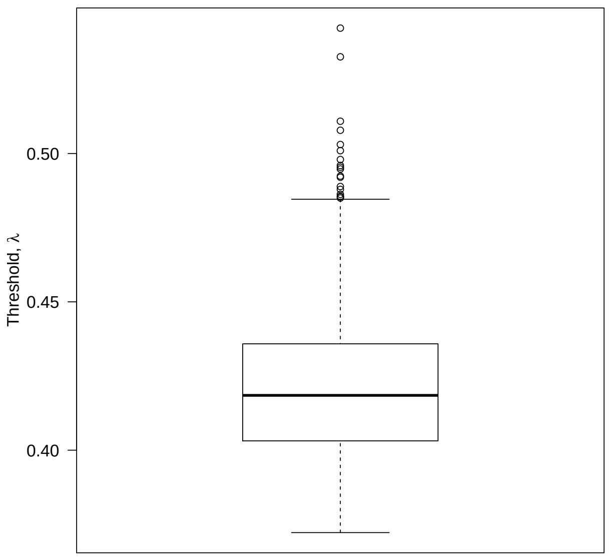


Figure 6.14: The variation of λ for the three-component concentric circle data.

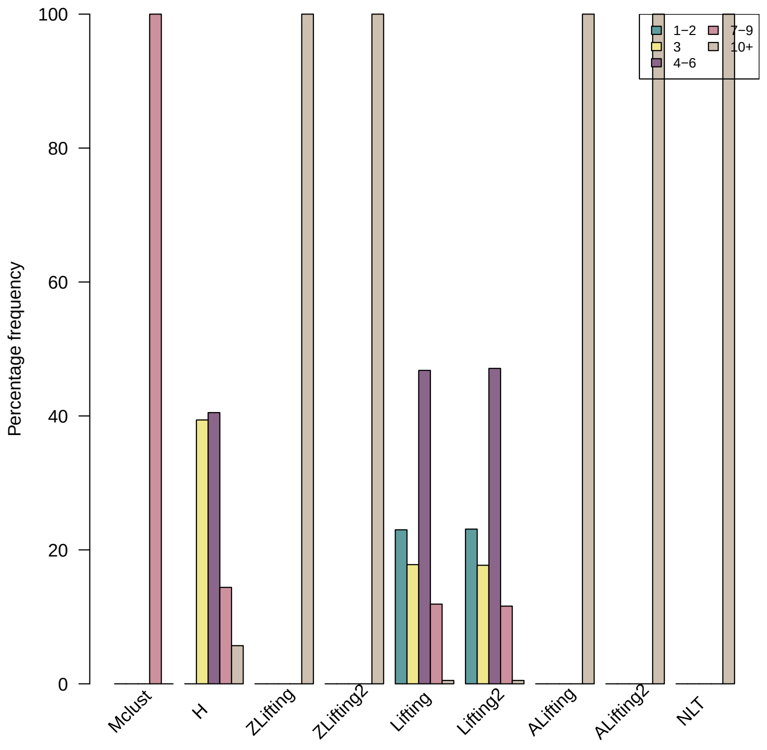


Figure 6.15: Updated bar chart: the comparison of number of clusters for the three-component concentric circle data. Legends illustrate the number of cluster range. Vertical axis shows the percentage of replicates, and the horizontal axis is for the different CVIs.

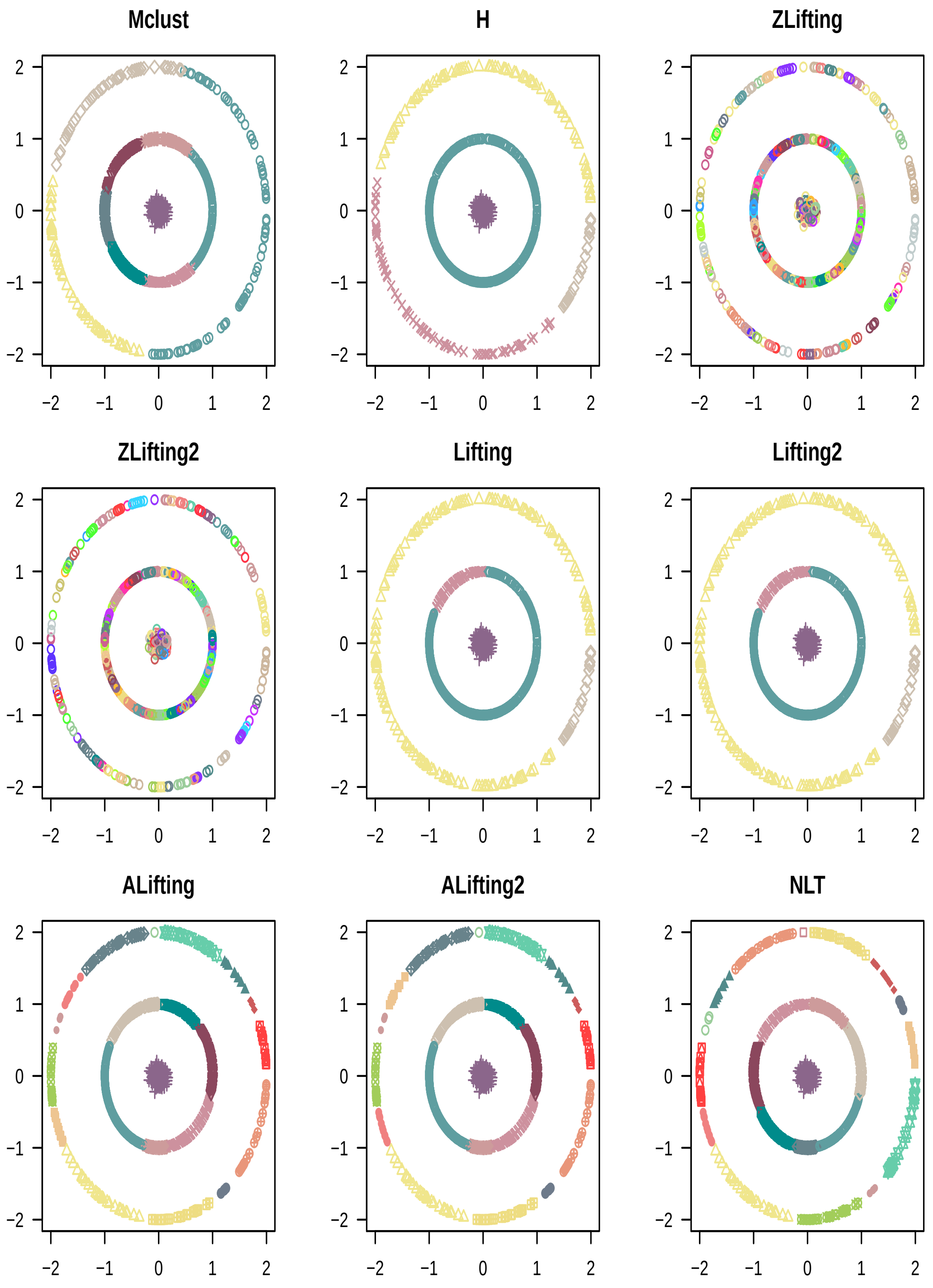


Figure 6.16: *Updated results for a single realisation of the three-component concentric circle data. Scatter plots illustrate the clustering solution chosen by different methods.*

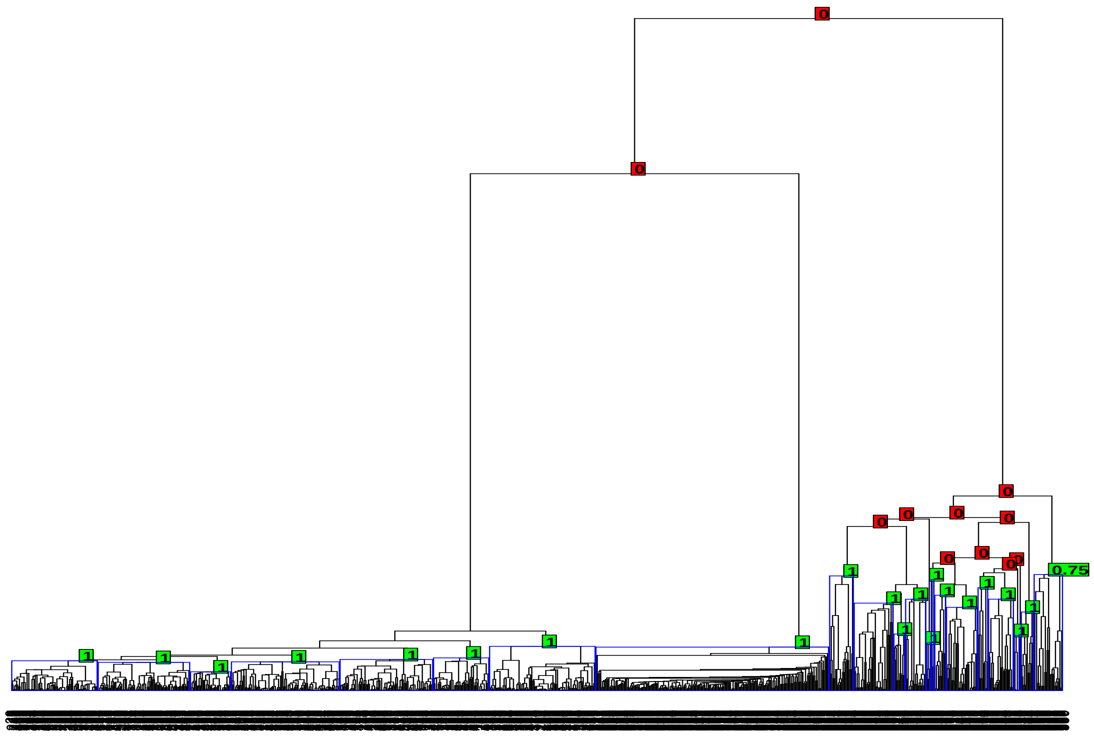


Figure 6.17: The dendrogram of our NLT algorithm for one realisation of the three-component concentric circle data. Internal nodes are labelled with the clustering probabilities colored with green, $p \geq \theta$, or red, $p < \theta$.

Index	Mclust	CH	ZLifting	ZLifting2	Lifting	Lifting2	ALifting	ALifting2	NLT
N	9	9	168	167	22	21	13	12	13
Purity	0.943	0.915	0.961	0.889	0.976	0.856	0.996	0.868	0.996
ARI	0.873	0.931	0.100	0.115	0.899	0.909	0.949	0.953	0.950
AVI	0.770	0.908	0.461	0.490	0.893	0.906	0.929	0.936	0.931
CompCheck	0.836	0.993	0.070	0.075	0.874	0.888	0.923	0.929	0.925
ClustCheck	0.968	0.910	0.817	1.00	0.974	0.974	0.997	0.997	0.997
CC	0.899	0.949	0.238	0.272	0.921	0.929	0.959	0.963	0.960

Table 6.5: Updated results of the comparison of CVIs for the six-component non-normally distributed data. First row is for the average number of clusters, and others are for the average similarity scores.

6.4.4 Six-component non-normally distributed data

The final artificial data structure we discuss is the one given in Section 5.5.4. We also update our results for this data set adding the results for our ZLifting, ALifting and NLT algorithms. The highest performance within the CVIs for this data structure was for the CH index, so we include the CH index results in the comparison study. In addition, the results of Mclust and our Lifting methods, $\lambda = 0.145$, take their place in this comparative study. Box plots comparing the different partitioning methods are shown in Figure 6.18. ZLifting method also finds high number of clusters in this data structure, so we check the thresholded results to see if there is any improvement of capturing the true components. The box plots illustrate that there is almost no variation observed in partitioning the data by ALifting method, and the proportion of capturing the true components of the data structure is higher than our Lifting method. The similar results with the ones for ALifting are found for our NLT algorithm. When we check the average of external scores for each CVI from Table 6.5, we see that ALifting and NLT capture around 95% of the true partitioning in terms of ARI. Thus, the best performance for this data set is found by our ALifting and NLT methods.

The average number of clusters is also much smaller than our Lifting method, and we also see that the number of clusters are not varied as our Lifting method as seen in Figure 6.19. In 70% of 1000 repetitions, both ALifting and our NLT algorithm find number of clusters within the range $[12-16]$. When we check one example of partitioning in Figure 6.20, ALifting method captures all of the components. It just divides one of the components into three parts (the one placed in the lower right part of the plot). It finds fewer outliers than our Lifting method, and it captures the tails of one of the components placed in the upper left part of the plot better than the CH index. In this example, our NLT algorithm behaves between ALifting and the CH index. While it behaves similar with ALifting generally, it could not capture the tails of one of the component like the

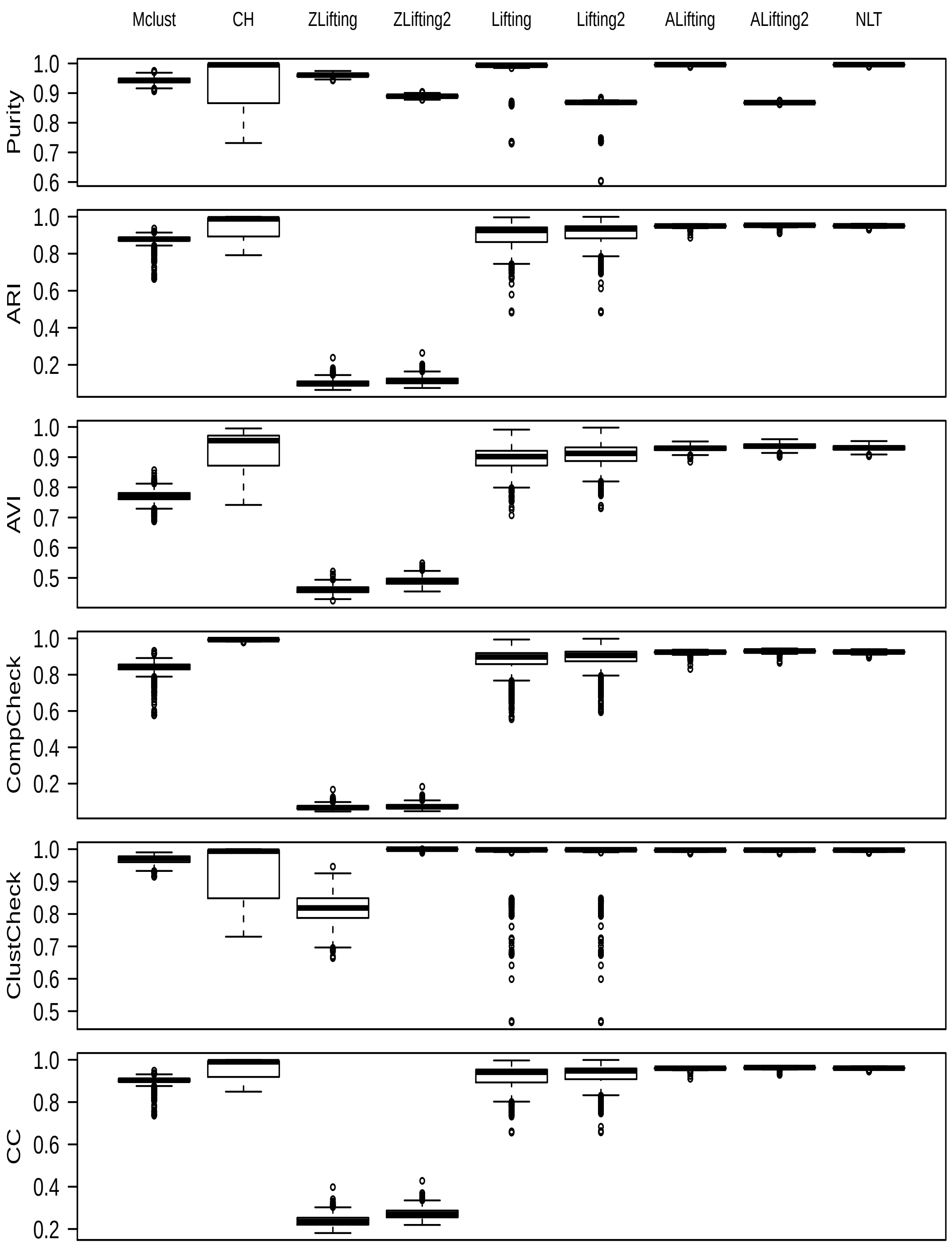


Figure 6.18: Updated box plot: the comparison of CVIs for the six-component non-normally distributed data.

CH index. Another illustration is done by drawing the labelled dendrogram of the data set including the clustering pattern found by our NLT algorithm, given in Figure 6.21. It locates the clusters at each node with high probability, but the probability of placing a cluster at one of the node is 0.42 which is close to $\theta = 0.5$. We may merge two sub-clusters under this node if we follow a different path.

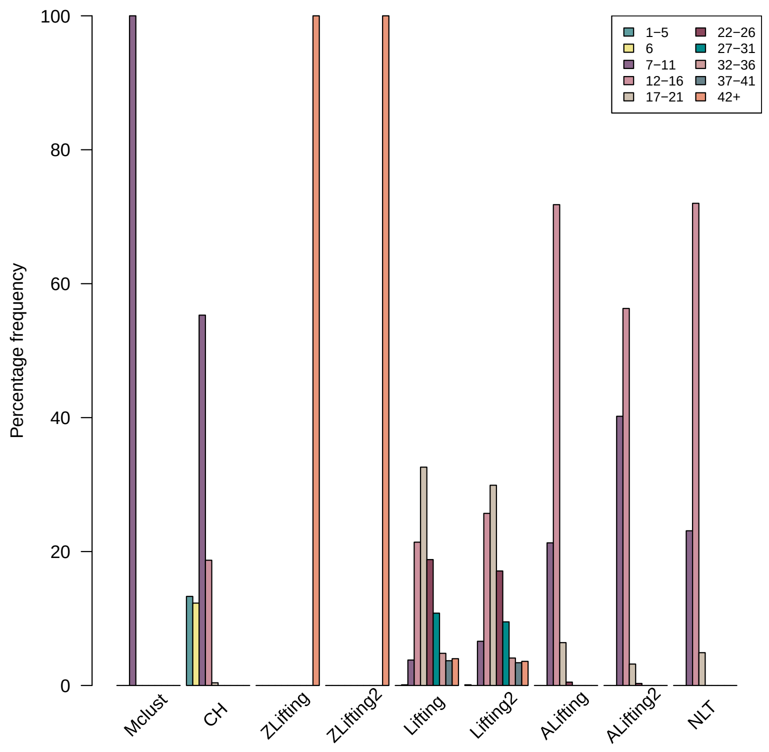


Figure 6.19: Updated bar chart: the comparison of number of clusters for the six-component non-normally distributed data. Legends illustrate the number of cluster range. Vertical axis shows the percentage of replicates, and the horizontal axis is for the different CVIs.

We can also see the variation of λ picked by our ALifting algorithm for each repetition in Figure 6.22. While we set $\lambda = 0.145$ in our Lifting method, Figure 6.22 shows that λ varies around 0.3. Our choice of λ for our Lifting method was lower than the ones algorithm picked itself, so the performance of our Lifting method shows some variation for each repetition. Thus, the variation of the number of clusters found by our Lifting method is decreased by ALifting method, and the performance of the algorithm is slightly increased.

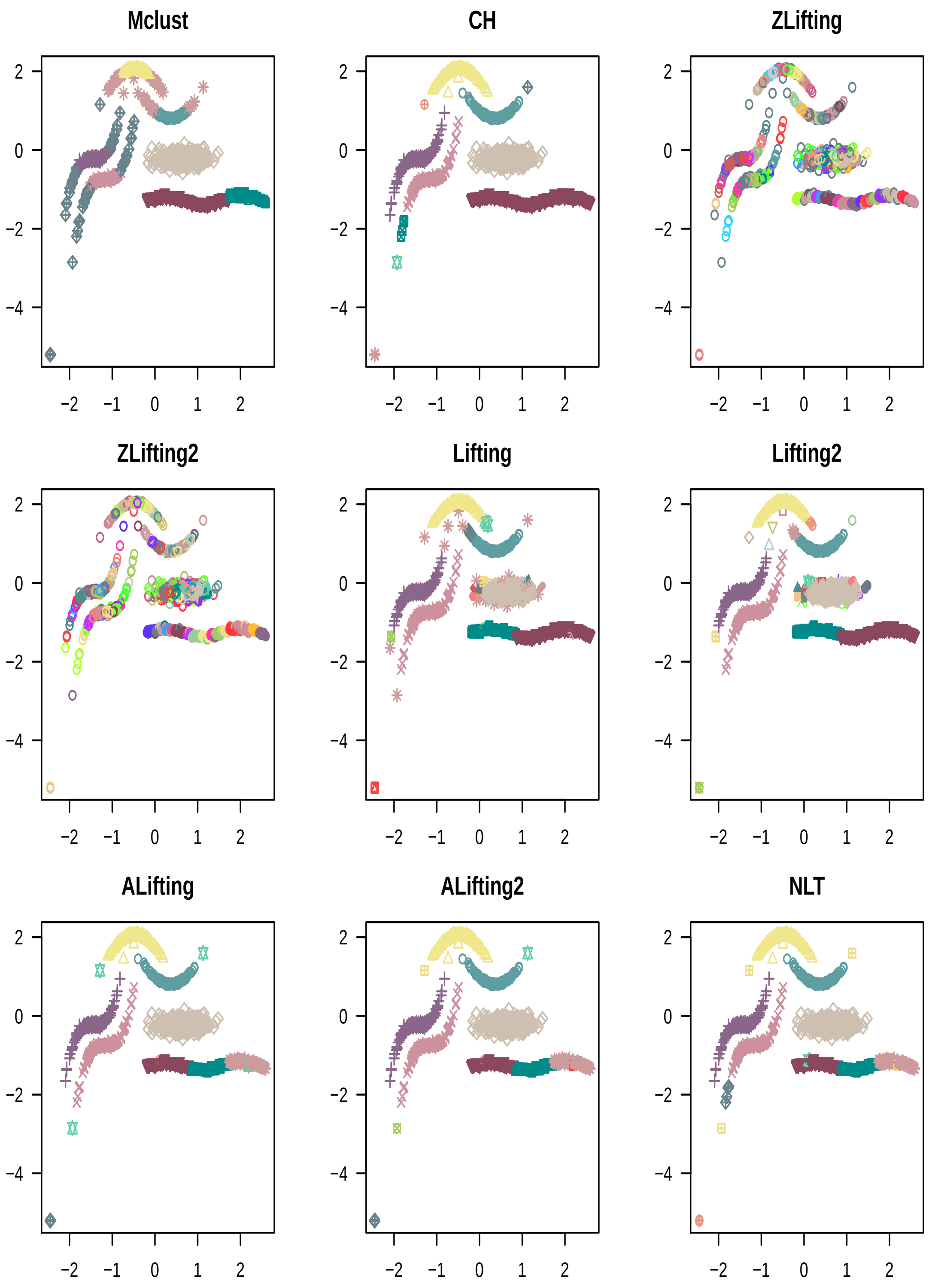


Figure 6.20: Updated results for a single realisation of the six-component non-normally distributed data. Scatter plots illustrate the clustering solution chosen by different methods.

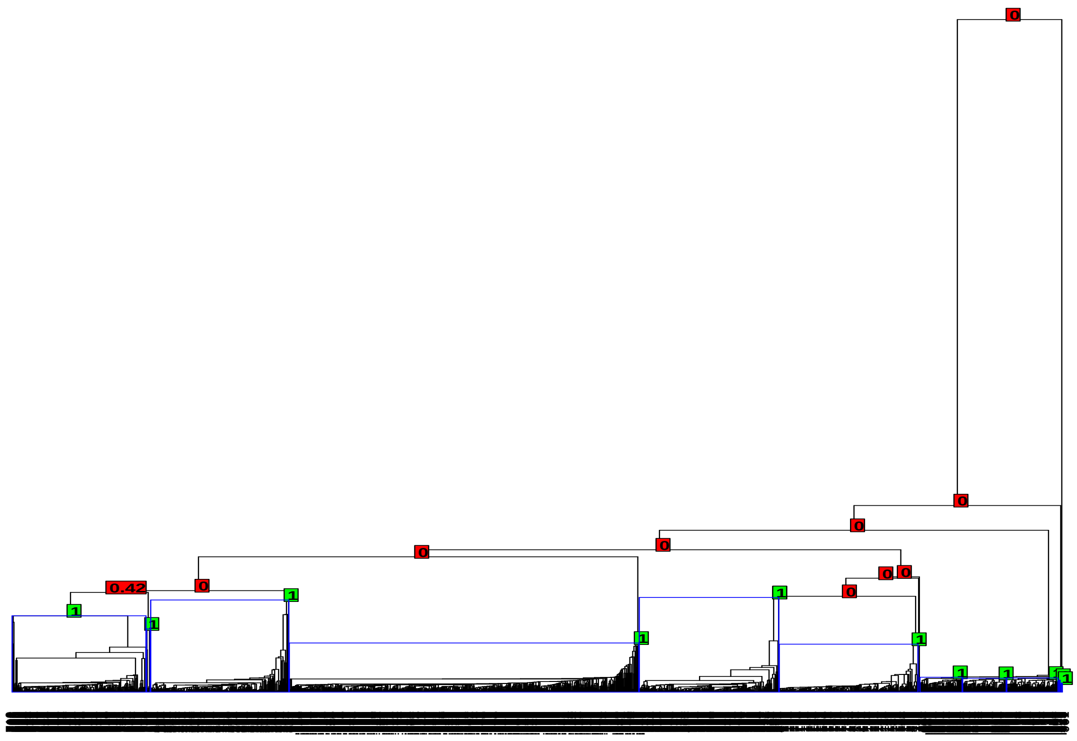


Figure 6.21: The dendrogram of our NLT algorithm for one realisation of the six-component non-normally distributed data. Internal nodes are labelled with the clustering probabilities colored with green, $p \geq \theta$, or red, $p < \theta$.

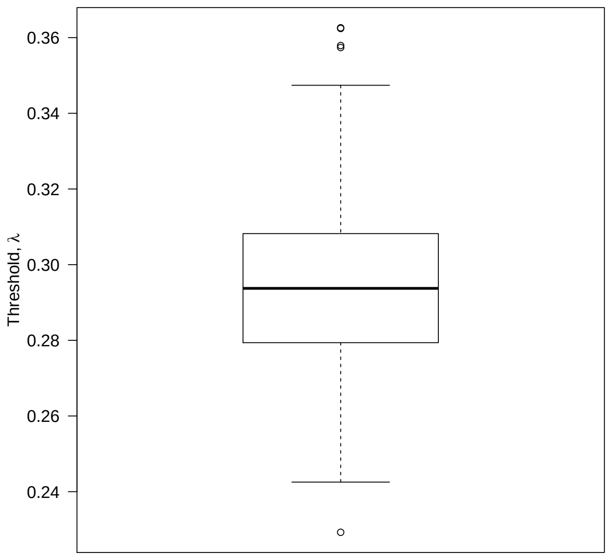


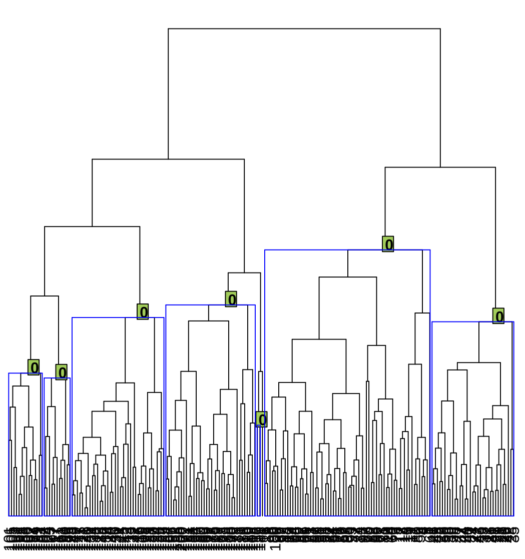
Figure 6.22: The variation of λ for the six-component non-normally distributed data.

Index	Mclust	Gap	Lifting	Lifting2	ALifting	ALifting2	NLT
N	4	5	8	7	8	7	4
Purity	0.900	0.850	0.890	0.688	0.860	0.693	0.825
ARI	0.760	0.620	0.593	0.597	0.611	0.615	0.619
AVI	0.800	0.704	0.673	0.678	0.683	0.689	0.695
CompCheck	0.838	0.671	0.571	0.577	0.642	0.649	0.746
ClustCheck	0.803	0.751	0.825	0.825	0.765	0.765	0.689
CC	0.820	0.710	0.686	0.690	0.701	0.704	0.717

Table 6.6: Updated results of the comparison of CVIs for crabs data.

6.5 Real data example

Results for the crabs data, used in Section 5.6 are updated and summarized in this section. The details of the data can be found in Section 5.6. In our Lifting method, we set the threshold to zero, so our ZLifting and Lifting methods are the same. We compare the results of Mclust, the Gap statistic (which had the highest performance in Section 5.6), Lifting, Lifting2, ALifting and ALifting2. In addition, we also add the results of our NLT algorithm in this comparison study by setting $\theta = 0.5$ and $P = 1000$. Even though the high number of paths ($P = 1000$) increases the computational load, we would like to decrease the probability of locating a wrong cluster by a high choice of P. The threshold, λ, found by our ALifting method is 0.67. Even though λ is noticeably larger than the threshold we picked for our Lifting method, Table 6.6 illustrates that the performance of ALifting is not much different than our Lifting method. There is only a small increase in the performance of ALifting compared to our Lifting algorithm, and the number of clusters found by both Lifting and ALifting algorithms are the same. Lifting and ALifting methods find one cluster more than Lifting2 and ALifting2, so there are some outliers captured by both methods. We notice that within our proposed methods, the NLT algorithm has the highest performance in terms of the similarity measures. In addition, both our NLT algorithm and Mclust find four clusters, but the partitioning can be different since Mclust is a different clustering method. Overall, the performance of capturing the true components by the Gap statistic and our NLT algorithm are almost the same. ALifting finds seven clusters and some outliers, and this can be seen from the dendrogram of the data, given in Figure 6.23a. Clusters found by ALifting are highlighted in the dendrogram, and the nodes, where clustering happen, are labelled with the denoised compactness values. We also produce a dendrogram with the clustering pattern and labelled with the clustering probabilities for our NLT algorithm, given in Figure 6.23b. We can easily see that the number of small clusters found by Lifting and ALifting is eliminated by our NLT algorithm. The probability of having a cluster at node 396 is 0.511 which is just above $\theta = 0.5$. When we compare the partitioning found by ALifting and NLT, ALifting divides


(a) ALifting.

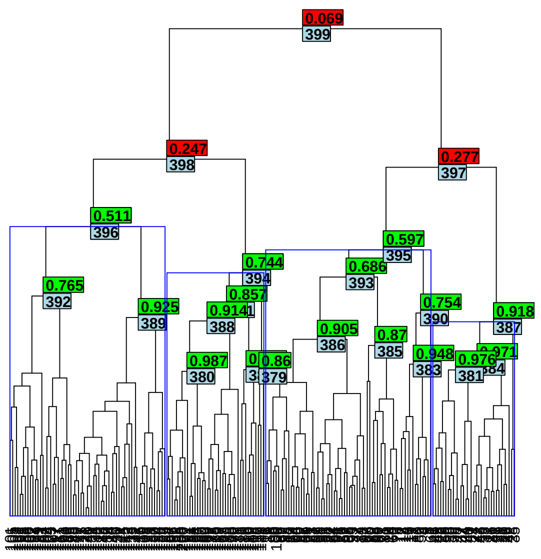

(b) NLT.

Figure 6.23: The dendrogram of the crabs data including clustering pattern found by ALifting and NLT. (a): internal nodes are labelled with the denoised compactness values. (b): internal nodes are labelled with the agglomeration order colored with light blue and the clustering probabilities colored with green, $p \geq \theta$, or red, $p < \theta$.

this cluster into two clusters where NLT also gives a higher chance to locate clusters at nodes 392 ($p = 0.765$) and 389 ($p = 0.925$). Another visual comparison is made by pairs plots, given in Figure 6.24.

We have also tabulated the number of crabs allocated to different clusters by Mclust, the Gap statistic, our NLT algorithm and ALifting in terms of true components, and given in Tables 6.7a , 6.7c , 6.7d and 6.7b, respectively. While the rows illustrate the partitioning found by different methods (Clusters, D_i), the columns illustrate the true partitioning of the crabs data (Components, C_j). Thus, column names are the combination of the colors (Blue (B), Orange (O)) and gender (Female (F), Male (M)) of crabs. We have noticed that all of them do similar partitioning, but ALifting finds some small sub-clusters for orange crabs. Thus, the performance of ALifting is lower than others, and we also need to remember that cluster D_1 includes outliers found by ALifting (Table 6.7b). While all four methods differentiate different colored crabs, none of them perfectly differentiate the crabs in terms of their gender.

		Components, C_j			
		BM	BF	OM	OF
Clusters, D_i	D_1	13	50	0	1
	D_2	37	0	0	0
	D_3	0	0	6	49
	D_4	0	0	44	0

(a) Mclust.

		Components, C_j			
		BM	BF	OM	OF
Clusters, D_i	D_1	1	0	0	0
	D_2	0	0	0	2
	D_3	0	0	6	5
	D_4	0	0	4	10
	D_5	16	50	0	0
	D_6	33	0	0	0
	D_7	0	0	3	33
	D_8	0	0	37	0

(b) ALifting.

		Components, C_j			
		BM	BF	OM	OF
Clusters, D_i	D_1	16	50	0	0
	D_2	33	0	0	0
	D_3	1	0	3	35
	D_4	0	0	10	15
	D_5	0	0	37	0

(c) The Gap statistic.

		Components, C_j			
		BM	BF	OM	OF
Clusters, D_i	D_1	16	50	0	0
	D_2	33	0	0	0
	D_3	1	0	3	35
	D_4	0	0	47	15

(d) NLT.

Table 6.7: Tabulated number of objects in each cluster found by various methods in terms of true clusters of the crabs data.

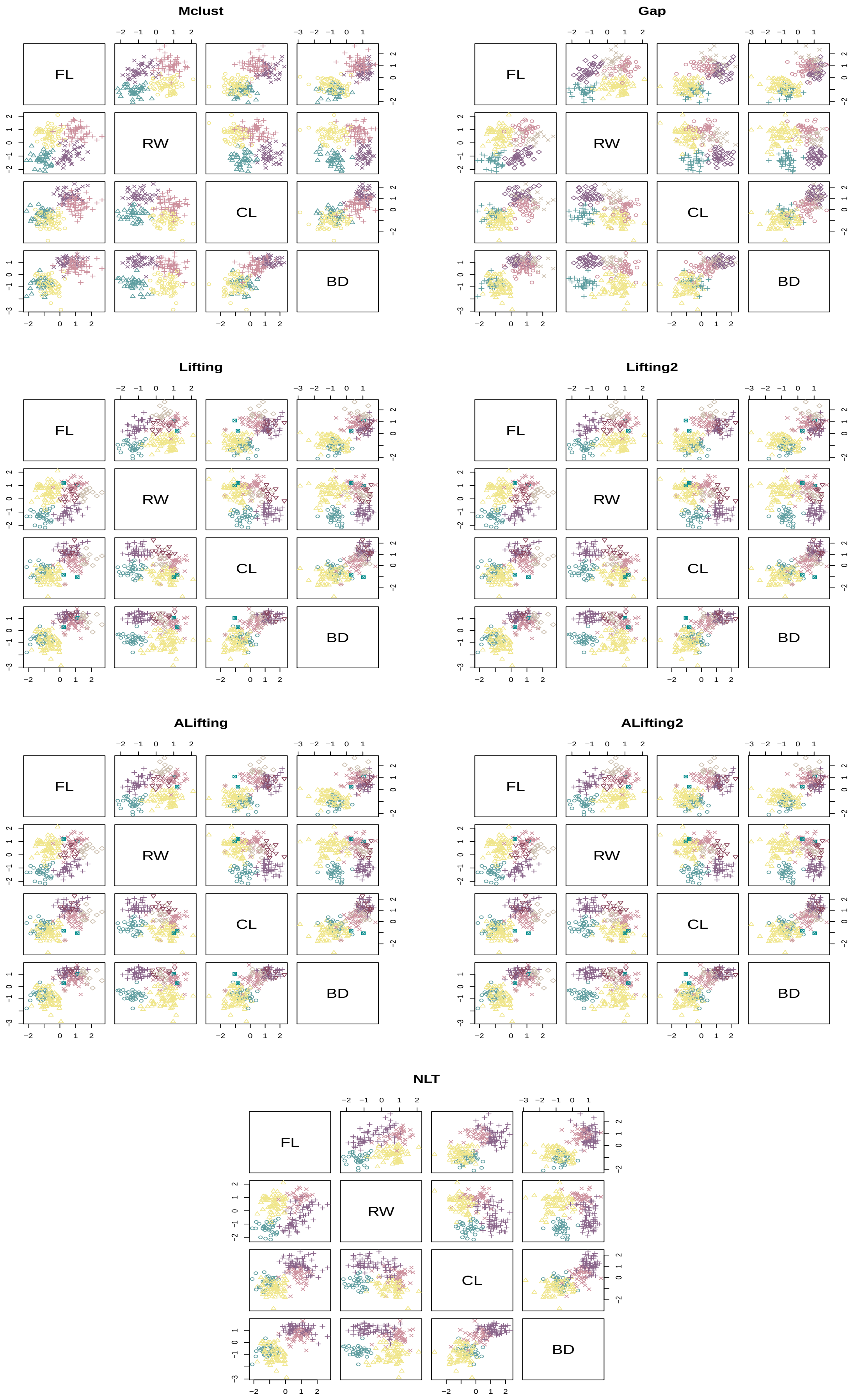


Figure 6.24: Illustration of partitions found by different CVIs for the crabs data.

6.6 Summary

In Chapter 5, we proposed a new method which automatically finds the number of clusters in hierarchical clustering, and it also showed us exactly where clustering happens in a dendrogram. The proposed method was based on the LOCAAT algorithm. The idea was to find a general node value for each node in a tree and denoise these node values by lifting. Thus, we suggested that we could use compactness values for each node, where we defined the compactness as the average distance from the centroid. Then if the denoised detail coefficients of a node and all its child nodes were zero or less than zero, we could highlight this node to be exactly where one of the clusters happened. Denoising to zero means all the objects below that node are placed in the centroid of the cluster, but we can allow some departures from the centroid since some of the objects can be close enough to the ones located in the centroid. Thus, we need to allow small departures from the centroid. In Chapter 5, we picked the allowed magnitude of these small departures manually. However, we prefer the algorithm to choose the tolerance parameter itself since we would like our algorithm to place clusters automatically in a dendrogram. Hence, in this chapter, we suggest that we can threshold the denoised compactness values using the idea of universal thresholding which is explained in Section 2.10.2. We apply this idea to the simulated data sets and real data set introduced in Chapter 5, and results are summarized in Sections 6.4 and 6.5. In simulation study, we have done 1000 replication, and we set the same threshold for each replicate when we apply our Lifting method. However, the threshold may vary from one replicate to another one, and picking the threshold automatically gets over this problem. Thus, we eliminate the arbitrary choice of a threshold by ALifting method.

In this chapter, as well as comparing Lifting and ALifting, we also include ZLifting (our Lifting method with zero threshold) in the comparison study. The results of ZLifting support the idea of needing a threshold. It always finds high number of clusters in a data set which means ZLifting finds small sized many clusters, but some of these clusters can be combined. In this case, the idea of setting a threshold either manually or automatically appears. Since manual choice of a threshold is an arbitrary choice, we can allow the algorithm to pick the threshold itself. Our findings show that the automatic choice of the threshold considerably increases the performance of our Lifting method. Results indicate that ALifting method finds the best representative partitioning to true components after Mclust for the five-component normally distributed data set. The best performance was already expected for these data sets from Mclust since true components are normally distributed. For three-component concentric circle data, the performance of the algorithm slightly increases in terms of AVI, but the average number of clusters increases. Thus, in our Lifting method, we set the threshold parameter higher than what ALifting method finds. The performance for the final simulation setting, six-component non-normally distributed data, increases substantially. When we applied our Lifting method, the best results were found for the CH index. However, now, ALifting method obtains the best results for this data structure; it clusters over 90% of true components correctly. For the real data set, the performance of the algorithm increases, and it finds a similar partition found by the Gap statistic which has the highest similarity scores after Mclust.

In this chapter, we also propose the application of our algorithm using non-decimated lifting transform (NLT) instead of using the standard LOCAAT algorithm. This provides us with a different perspective to consider the location of possible clusters calculating a probability of placing a cluster for each node. Thus, we have a chance to see how strongly a cluster can be located on a node in the tree by this version of our algorithm. When we check the results for each data structure, it finds the similar results to our ALifting algorithm.

We have also tried different settings in the denoising stage: denoising with and without artificial levels and normalization. We notice that assuming all detail coefficients come from the same resolution level and applying Bayesian thresholding in this situation gives the same results as applying Bayesian thresholding after setting artificial levels in the way presented by Jansen et al. (2009). We have also tried the case of ignoring the normalization step for Bayesian thresholding since Bayesian threshold assumes that the detail coefficients come from a normal distribution with $\sigma^2 = 1$, as explained in Section 2.10.3. We find that normalization does not have an effect on denoising with artificial levels, but if we denoise without artificial levels, normalization of the detail coefficients decreases the performance of our algorithm slightly. However, assuming that the detail coefficients come from the normal distribution with the required variance still gives the same results with other settings.

Overall, our ALifting and NLT methods find exactly where clustering happens in hierarchical clustering. Their performance for catching true components are always high and are close to each other, but if a data set is uniquely shaped, they achieve the best performance when we compare with other CVIs.

Chapter 7

Lifting on phylogenetic trees

7.1 Introduction

The relationships between different species are explained by phylogenetic trees; a detailed discussion on phylogenetic trees and their basic properties was given in Chapter 4. One popular question in phylogenetics is also to explore which organisms share the same ancestor, which links well with our question in Chapter 5: where exactly does clustering happen on a binary tree? A binary tree is tree structured data, where each node has at most two child nodes. Hence, we can apply the algorithm we proposed in Chapter 5 and expanded in Chapter 6 to phylogenetic trees.

In terms of phylogenetics, the purpose of our algorithm is to find where exactly the cluster of related species are located using DNA sequences of different organisms. As long as we have DNA sequences of different organisms, we can easily modify our algorithm. Thus, in this chapter, we discuss how we rearrange the algorithm proposed in Chapter 6 to find coherent clusters of species. After we describe these modifications in Section 7.2, we continue with a sequence based simulation study and the application of the algorithm on a real data set in Sections 7.3 and 7.4, respectively. We also check the behaviour of our algorithm on cophylogenetic data sets in Section 7.5, where we first give a small background discussion on cophylogeny and continue with the application of our algorithm on two different real data sets. We finalize the chapter with a discussion of the behaviour of our algorithm on phylogenetic data sets in Section 7.6.

7.2 Finding number of clusters for phylogenetic data

In this section, we introduce a modified version of the algorithm given in Chapter 6 which we can use to analyze phylogenetic data sets. The proposed algorithm finds where exactly clustering happens by denoising a tree using the ALifting algorithm. When we talk about phylogenetic data sets, we basically refer to DNA sequences constructed by four different

Species	DNA sequences
s_1	A C A A T T C T C G G G C G A C C T G A
s_2	A C G G A G C C C T A A T T A C C T A C
	0 0 1 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 1

Table 7.1: A small example of DNA sequences. The length of aligned sequences are equal to 20*. The third row shows the comparison of sequences: the position of different nucleotides coded as 1 and the matching nucleotides coded as 0.*

nucleotides: adenine (A), thymine (T), guanine (G) and cytosine (C). Thus, our data sets are a number of sequences of different lengths constructed by various combinations of the four letters, A, T, G and C. To have a better understanding of the process, we start with an explanation of building a tree using the DNA sequences.

Firstly, we need to align sequences. An alignment procedure was explained in Section 4.2, and sequences can be aligned using MUSCLE (Edgar, 2004). The package **ape** (Paradis et al., 2004) in **R** allows us to call MUSCLE. Secondly, the distance matrix needs to be determined for the aligned sequences, which could be measured using one of two methods:

1. Matching distances. We find the pairwise matching distances between sequences by counting the number of loci which have different nucleotides. Thus, the pairwise matching distances between sequences, s_i and s_j, are defined as

$$d_{s_i s_j} = \sqrt{\sum_{k=1}^{\ell} \mathbb{1}_k}, \tag{7.1}$$

where

$$\mathbb{1}_k = \begin{cases} 1, & s_{ik} \neq s_{jk} \\ 0, & s_{ik} = s_{jk}, \end{cases}$$

where s_{ik} is the nucleotide at locus k in ith sequence, $i, j \in \{1, \ldots, n\}$, n is the number of sequences, and ℓ is the length of sequences after alignment including gaps. We also need to note that sequences may have some gaps after alignment. If there are any gaps in any position in any sequence, those loci are coded as zero since we do not have the exact information for them.

We can illustrate how to find the matching distances between sequences using a toy data set. Assume that we have two aligned sequences such as the example in Table 7.1, where the third row illustrates the coded comparison of two sequences s_1 and s_2. If two sequences share the same nucleotide in the same position, we will code by zero, else we will code by one. There are 12 loci labelled as one. Thus, the matching distance between species s_1 and s_2 is

$$d_{s_1 s_2} = \sqrt{12} \approx 3.464.$$

2. Phylogenetic distances. Another option of distance matrix calculation is to use one of the evolutionary models described in Section 4.4. Phylogenetic distances can be easily calculated using the **dist.dna()** function in the **ape** package (Paradis et al., 2004) in **R**.

Thirdly, we can build phylogenetic trees using hierarchical clustering. Building the tree provides two fundamental properties required for our algorithm: which species are joined in each agglomeration stage and branch lengths between each node in the tree. The final condition of our algorithm is a node value for each node in the tree. Some data sets may come with a node value, but it is not always possible. Thus, we propose two different node values for phylogenetic trees:

1. Compactness. We proposed a node value called compactness in Section 5.4. Our definition of compactness was the average Euclidean distances from the centroid of each cluster. We can still use the same definition for the compactness for phylogenetic data sets, but we need to do some pre-processing since DNA sequences are not points in $\mathbb{R}^p$. We start the algorithm with DNA sequences, so we need to find the position of these sequences in some representative space. In this stage, we use multidimensional scaling (MDS) to find coordinates of a point representing each sequence in the space. To do this, we need to check eigenvalues to find how many dimensions are needed to explain our data. Eigenvalues can easily be checked using the **eigen()** function, and MDS can be applied using the **cmdscale()** function in **R**. Then we can compute compactness for each cluster (each node in the tree) using Equation (5.15).

2. Dissimilarity score. We propose that dissimilarity scores can be used as a node value for phylogenetic trees. We basically compute the dissimilarities between each sequence under the node of interest. We compute $n-1$ different scores for a tree since there are $n-1$ nodes in a rooted tree. Thus, we can compute the dissimilarity score for the node h, $\mathrm{Dis_h}$, ($h \in \{1, \ldots, n-1\}$) in a tree using

$$\mathrm{Dis}_h = \frac{1}{\ell} \sum_{k=1}^{\ell} \mathbb{1}_k, \tag{7.2}$$

where

$$\mathbb{1}_k = \begin{cases} 1, & \text{if any } s_{ik} \neq s_{jk}, \\ 0, & \text{if all } s_{ik} = s_{jk}, \end{cases}$$

where s_{ik} is the nucleotide at locus k in ith sequence, $s_i \in \Omega_h$, and $s_j \in \Omega_h \setminus s_i$, Ω_h represents the sequence space for sequences under the node h, and ℓ is the length of sequences after alignment including gaps. We treat gaps in aligned sequences in the same way we discussed earlier.

Assume that two aligned sequences given in Table 7.1 share the same last common ancestor (LCA). In this case, the dissimilarity score for this node is the average number of loci where these two sequences have different nucleotides, so using the third row of Table 7.1,

$$\mathrm{Dis} = 12/20 = 0.60.$$

After finding node values, we are able to apply our Lifting algorithm for multidimensional data sets introduced in Section 5.4.1 to phylogenetic data. For multidimensional data sets, we proposed that we could use the compactness value as a node value. Then we found the lifting transformation of compactness values by applying the lifting algorithm described in Section 3.8.2. The lifting process was followed by a denoising stage using the Bayesian wavelet shrinkage approach discussed in Section 2.10.3. In this way, we obtained the denoised detail coefficients for each node in the tree. After denoising the tree, we proposed that if any node, including all the nodes below it had denoised detail coefficients less than or equal to a threshold, λ, we could treat them as one cluster since all the species under that node would be placed around the centroid of the cluster. In addition, we proposed ALifting algorithm which automatically picks the threshold, λ, in Section 6.2. In ALifting, we find the clustering pattern using denoised compactness values instead of using denoised detail coefficients. We will use the same procedure for phylogenetic trees if compactness values are used as node values. In the case of using dissimilarity scores as node values, the procedure is the same, but interpretation of results is slightly different. Assume that we apply the ALifting algorithm. If a node in a phylogenetic tree with all the nodes below that node have denoised dissimilarity scores less than or equal to λ, we will interpret that all the species under that node create one cluster since they share similar DNA sequences.

Example 7.2.1. *To illustrate the mechanism of our algorithm on phylogenetic data sets, we generate a toy data set which includes aligned DNA sequences of ten species with length of* 20*. The aligned sequences are illustrated in Table 7.2. We will illustrate the case if we use matching distances and dissimilarity scores as node values. To find the distances between each pair, we count the different nucleotides in the same position of sequences as we illustrate in Table 7.1 for DNA sequences of species s_1 and s_2. For illustrative purposes, we can repeat the process for species e.g. s_3 and s_9. Table 7.3 depicts the loci of different nucleotides for species s_3 and s_9, so the matching distance between these two species is $d_{39} = 1$.*

After finding distances between each pair, the distance matrix, D, is generated and

Species	DNA sequences
s_1	A C A A T T C T C G G G C G A C C T G A
s_2	A C G G A G C C C T A A T T A C C T A C
s_3	A G G A T A T C C G A G T T A C T T G C
s_4	A G A A T A T T T G A A T T A C T C A A
s_5	C C G A C A C T T G A G T G A T C C C C
s_6	A C G G A T C C C C A A T T A T C T A C
s_7	A C A A T A T C C G A G T C A C C T G A
s_8	A C G A T A T T T G A G T T G C C T G A
s_9	A G G A T A T C C G A G T T A C C T G C
s_{10}	A C G G A T C C C C A A T T A T C T A C

Table 7.2: Toy phylogenetic data. The length of aligned sequences are equal to 20.

Species	DNA sequences
s_3	A G G A T A T C C G A G T T A C T T G C
s_9	A G G A T A T C C G A G T T A C C T G C
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

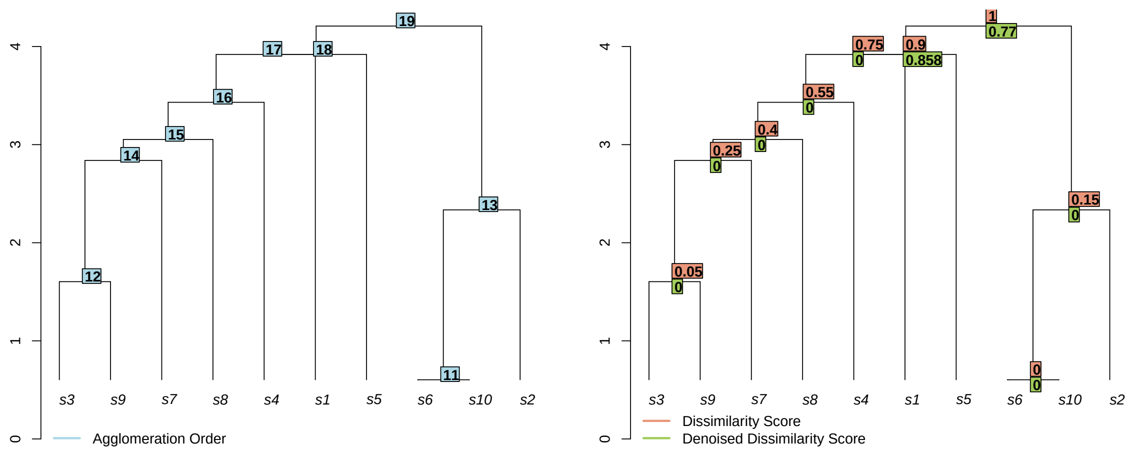
Table 7.3: Aligned species s_3 and s_9. The third row shows the comparison of sequences: the position of different nucleotides coded as 1 and the matching nucleotides coded as 0.

given in Equation (7.3).

$$D = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \\ s_9 \\ s_{10} \end{array} \begin{array}{c} \begin{array}{cccccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} \end{array} \\ \begin{bmatrix} 0.000 & 3.464 & 3.162 & 3.317 & 3.317 & 3.464 & 2.449 & 2.828 & 3.000 & 3.464 \\ 3.464 & 0.000 & 3.000 & 3.464 & 3.464 & 1.732 & 3.162 & 3.317 & 2.828 & 1.732 \\ 3.162 & 3.000 & 0.000 & 2.646 & 3.317 & 3.162 & 2.236 & 2.449 & 1.000 & 3.162 \\ 3.317 & 3.464 & 2.646 & 0.000 & 3.317 & 3.606 & 2.828 & 2.646 & 2.828 & 3.606 \\ 3.317 & 3.464 & 3.317 & 3.317 & 0.000 & 3.317 & 3.317 & 3.000 & 3.162 & 3.317 \\ 3.464 & 1.732 & 3.162 & 3.606 & 3.317 & 0.000 & 3.317 & 3.464 & 3.000 & 0.000 \\ 2.449 & 3.162 & 2.236 & 2.828 & 3.317 & 3.317 & 0.000 & 2.236 & 2.000 & 3.317 \\ 2.828 & 3.317 & 2.449 & 2.646 & 3.000 & 3.464 & 2.236 & 0.000 & 2.236 & 3.464 \\ 3.000 & 2.828 & 1.000 & 2.828 & 3.162 & 3.000 & 2.000 & 2.236 & 0.000 & 3.000 \\ 3.464 & 1.732 & 3.162 & 3.606 & 3.317 & 0.000 & 3.317 & 3.464 & 3.000 & 0.000 \end{bmatrix} \end{array} . \tag{7.3}$$

*Since we have the distance matrix, we can easily build the phylogenetic tree using hierarchical clustering, and we use complete linkage in this illustration. The tree is built using the **hclust()** function in **R** and given in Figure 7.1a.*

Before starting the ALifting algorithm, we compute dissimilarity scores for each node in the tree. The first joined pair on the phylogenetic tree is s_6 and s_{10}, and they generate node 11. *The dissimilarity score for node* 11 *is based on the number of different nucleotides in the same position of each species under node* 11. *Sequences for species*



(a) *(b)*

Figure 7.1: Illustration of the toy phylogenetic data with the results of our ALifting method. (a): the nodes are labelled with the agglomeration order started from $n+1 = 11$*, where* $n = 10$*. (b): the nodes are labelled with the results of our ALifting method.*

Species	DNA sequences
s_6	A C G G A T C C C C A A T T A T C T A C
s_{10}	A C G G A T C C C C A A T T A T C T A C
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Table 7.4: Aligned species s_6 *and* s_{10}*. The third row shows the comparison of sequences: the position of different nucleotides coded as 1 and the matching nucleotides coded as 0.*

s_6 *and* s_{10} *with the sign vector which spots the place of different nucleotides between these two sequences are given in Table 7.4. These two sequences are identical, so the dissimilarity score for node* 11 *is* $\text{Dis}_{11} = 0/20 = 0$.

We find dissimilarity scores for each node using the same method. If an internal node in a tree includes more than two species, calculations will be exactly the same. For example, Figure 7.1a illustrates that node 11 *and species* s_2 *join in the third agglomeration stage and generate node* 13*. Node* 13 *has three species (*s_6*,* s_{10} *and* s_2*) under it. The sign vector for these species is given in Table 7.5. Thus, the dissimilarity score for node* 13 *is* $\text{Dis}_{13} = 3/20 = 0.15$.

The dissimilarity scores for each node in agglomeration order are found as $(0.00, 0.05, 0.15, 0.25, 0.40, 0.55, 0.75, 0.90, 1.00)$*. Now, we can apply our ALifting algorithm, given in Section 6.2. The first part of our ALifting algorithm is to denoise the tree by lifting algorithm. Thus, the denoised tree is obtained, and given in Figure 7.1b, where internal nodes are labelled with the dissimilarity scores and the denoised dissimi-*

Species	DNA sequences
s_6	A C G G A T C C C C A A T T A T C T A C
s_{10}	A C G G A T C C C C A A T T A T C T A C
s_2	A C G G A G C C C T A A T T A C C T A C
	0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0

Table 7.5: Aligned species s_6, s_{10} and s_2. The fourth row shows the comparison of sequences: the position of different nucleotides coded as 1 and the matching nucleotides coded as 0.

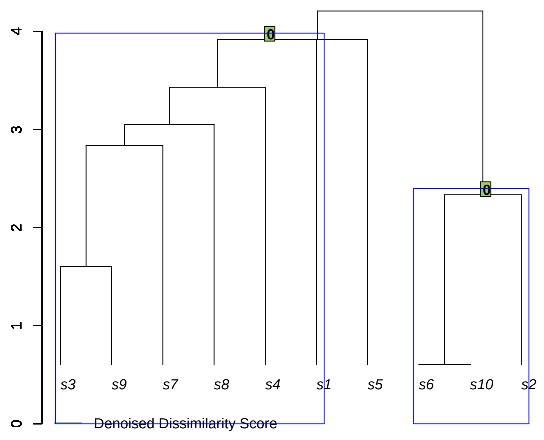


Figure 7.2: The clustering scheme found for the toy phylogenetic data by our ALifting method.

larity scores for each cluster. Then, using the denoised dissimilarity scores, we can decide where to cut the tree. If any dissimilarity score is less than or equal to $\lambda = 0.821$, we will set them as zero. Thus, if any node and all its child nodes are set as zero, we can treat them as one cluster. Under the light of this information, the tree is cut and illustrated in Figure 7.2. The tree is cut from the exact height of each cluster, and two clusters are found. One of the species is also found to be an outlier, and it is not clustered.

7.3 Simulation study

To see the behaviour of our algorithm, we present a sequence base simulation study. We generate a number of DNA sequences for a fixed topology of a tree. To generate sequences, we need a base tree, so we randomly generate a tree. Then we simulate DNA sequences using the generated tree. Thus, we need two different stages before starting the application of our algorithm: generating a fixed topology of a tree and simulating sequences based on the generated tree. To find the base tree, Hudson (2002) proposed a method (**ms**) based on the Wright-Fisher algorithm introduced by Ewens (1979). Thus, we generate the base tree via **ms** implemented in the **ms()** function in the **phyclust** package (Chen, 2011) in **R**. The details of **ms** are discussed in Section 7.3.1. Then we continue with the procedure of generating sequences based on the tree generated by **ms**. We use **Seq-Gen** (Rambaut & Grassly, 1997) which generates sequences using different evolutionary models, which is also available as the **seqgen()** function in the **phyclust** package (Chen, 2011) in **R**. The process of generating sequences is discussed in Section 7.3.2. Then we present the result of our algorithm for the simulated DNA sequences in Section 7.3.3.

7.3.1 Finding base tree structure

In population genetics, one of the preferred models is the Wright-Fisher model (Ewens, 1979) because of its mathematical convenience. It is based on Markov chain theory. The simple case of the Wright-Fisher model is a population without selective differences, where the population size is fixed at N. The Wright-Fisher model assumes that if there are N individuals in Generation 0, there will also be N individuals in Generation 1. In addition, there are $2N$ genes in the population since each individual has two genes in each locus. If we focus on a locus "B", two possible alleles may occur: "B_1" and "B_2". The possible genotypes at this locus are B_1B_1, B_1B_2 and B_2B_2. Thus, we can focus on one of the genes in each locus. Assume that X is the number of B_1 genes in any generation, so $X \in \{0, 1, \ldots, 2N\}$. The number of B_1 genes in generation t are shown by $X(t)$.

Using the Wright-Fisher model, we can estimate the number of "B_1" genes in generation $t + 1$, denoted by $X(t + 1)$, by sampling with replacement the "B_1" genes in generation t. Thus, $X(t + 1)$ is a binomial random variable and p_{ij} be the probability of having j number B_1 genes in generation $t+1$ when the number of B_1 genes in generation t is i. Then p_{ij} is defined as

$$p_{ij} = \binom{2N}{j} \left(\frac{i}{2N}\right)^{j} \left(1 - \frac{i}{2N}\right)^{2N-j}, \tag{7.4}$$

where $X(t) = i$, $X(t + 1) = j$ and $i, j \in \{0, 1, \ldots, 2N\}$. Each individual in Generation 0 produces many gametes, and individuals in Generation 1 are drawn from this gametes pool randomly. Thus, the probability of picking any gene from the Generation 0 is equally likely with probability $1/(2N)$.

Hudson (2002) proposed a method, **ms**, based on standard coalescent theory and the Wright-Fisher model, to simulate some evolving populations. Coalescent theory deals with parameter estimation in population genetics (i.e. time of the last common ancestor, population size when coalescence happened, and information about the extinction or migration of the population). The topology of the genealogy can be randomly chosen, so picking any topology is equally likely, but coalescent times (branch lengths) are exponentially distributed. Thus, for k individuals, $k \in \{1, \dots, n\}$ ($n \ll N$),

$$\begin{aligned} P(\text{no coalescence 1 generation back}) &= \prod_{\ell=0}^{k-1} \exp\left\{-\frac{\ell}{2N}\right\} \\ &\leq \exp\left\{-\frac{\binom{k}{2}}{2N}\right\}, \end{aligned}$$

and

$$P(\text{no coalescence } t \text{ generations back}) \leq \exp\left\{-\frac{\binom{k}{2}t}{2N}\right\}.$$

In addition, the expected time of coalescence for k individuals is

$$E(\text{coalescent time for } k \text{ individuals}) = \frac{1}{\binom{k}{2}}.$$

The **ms** function generates a random tree topology using the Wright-Fisher model and assumes that branch lengths are exponentially distributed with mean $\binom{k}{2}^{-1}$. Then it randomly places a number of mutations, θ, in each branch, where $\theta \sim \text{Po}(\mu\tau)$; here μ is the mutation rate and τ is the branch length.

We can also consider another type of base tree in our study by generating isolated populations. This can be done considering some sub-populations which allow migration between each other. One well-known model of migration is the island model (Wright, 1943), which assumes that a large population can contain many sub-populations, and they have geographical distance between each other like islands. Each sub-population is considered as a large population, so the migration between them can not easily be recognized. Assume that the frequency rate of B_1 and B_2 alleles differ for each sub-population, and the frequency of alleles between migrants is equal to the average number of alleles between sub-populations, $\bar{p}$ for B_1 and $\bar{q}$ for B_2. The migration rate, m, is the probability that a randomly picked allele from any sub-population belongs to a migrant. Thus, at time t, a randomly chosen allele may come from the same population at time $t-1$ with probability $1-m$, or from a migrant with probability m. Hence, the frequency of allele B_1 at time t is defined as

$$p_t = p_{t-1}(1-m) + \bar{p}m,$$

where p_{t-1} is the frequency of B_1 allele in generation $t-1$, and this can be generalized for a number of generations as

$$p_t = \bar{p} + (p_0 - \bar{p})(1-m)^t,$$

where p_0 is the initial frequency of B_1 allele in the considered sub-population.

After building a tree with the desired evolutionary history, we need to generate sequences using the tree as a base tree in our simulation study. Thus, we continue with the discussion how to generate DNA sequences in the following section.

7.3.2 Generating sequences

In DNA sequence generation, we use **Seq-Gen** (Rambaut & Grassly, 1997) which shares the same idea with the probabilistic matrix approach proposed by Schoöniger & Haeseler (1995). **Seq-Gen** offers a number of evolutionary model choices, and it also offers site-specific rate heterogeneity.

Schoöniger & Haeseler (1995) picked the HKY model (Hasegawa et al., 1985) as a base evolutionary model for their method since it is the general version of all other models. The HKY model and its transition probability matrix were defined in Section 4.4.4. The software needs the length of the sequences, transition/transversion rate (α/β), nucleotide frequencies (π_i, $i \in \{A, C, G, T\}$) and a base tree which includes branch lengths. Schoöniger & Haeseler (1995) defined

$$\tau = -t \sum_i \pi_i q_{ii}, \tag{7.5}$$

where τ is the number of substitutions during time t, and $\{q_{ii}\}$ are the diagonal elements of the matrix Q for the HKY model, defined in Section 4.4.4. Since we start the procedure of generating sequences with a base tree, we know the expected number of substitutions between time t and $t + 1$, τ, and branch lengths coming from the base tree. Thus, we can find the time parameter, t, from Equation (7.5), and then we can easily calculate the transition probabilities as explained in Section 4.4.4.

To generate sequences, **Seq-Gen** randomly allocates a sequence to the root with the desired length of the sequence. Then it evolves the tree until it reaches the leaves using the given parameters. Using the allocated sequence for the root, **Seq-Gen** computes the transition probabilities for the first position in the sequence and changes the nucleotide in the first position with the specified transition probability. Assume that nucleotide i is placed in the first position. The nucleotide i is replaced by nucleotide $j \in \{A, C, G, T\}$ with the probability of $p_{ij}(t)$. This process is repeated for each position in the sequence in turn. Thus, the simulated sequence for the next generation is found, and the procedure is repeated until the sequences for tips are found.

Site-specific rate heterogeneity choice needs a different transition probability calculation. Yang (1993) proposed a maximum likelihood method to generate sequences, where substitution rates differ over sites and come from a Gamma distribution. The mean of the distribution is fixed at one, requiring the shape and scale parameters to be equal to each other. Thus, allowing site-specific rate heterogeneity choice in **Seq-Gen** needs one more parameter to be specified, which is the shape parameter for the Gamma distribution.

7.3.3 Simulation results

In simulation settings, we generate three different tree-structures using **ms** which are given in Figure 7.3. Two of them are generated including only mutations in their evolutionary history using the same parameters with different seeds, and there is no information for true classification in these settings. These trees are referred as the first and second base trees with no sub-populations. Both trees include 100 species, and mutation parameter is set at 0.2 in units of $2N$ generations, where N is the population size. For the third tree (referred as a population with three sub-populations), we add more complication, and we generate 30 sequences by taking migration into account along with mutation. Mutation and migration parameters are set at 2 and 1, respectively, and three equal size sub-populations are generated. When we evaluate populations with migration history in terms of classification idea, we consider sub-populations as true components, and compare our classifications to this "truth". Thus, we can check the performance of different classification methods using external classification scores, introduced in Section 5.3.3 (e.g. purity index). After building the tree of interest, DNA sequences are generated by **Seq-Gen**. The Kimura 2-parameter model is used with transition/transversion rate 2, and the length of sequences is fixed at 9000. For each base tree, 1000 replicates are generated, so for each replicate, a new set of sequences is generated with the same parameters. We start our algorithm with the simulated DNA sequences, so we rebuild the tree within our algorithm using one of the distance measures defined in Section 7.2. In this simulation study, the partitioning results found by other internal cluster validity indices (CVIs), described in Section 5.3.1, are based on the same tree built within our algorithm, and we use the same distance matrix used in our algorithm for model-based clustering (Mclust). We set the upper boundary for the internal indices as 50 and 20 for the data sets with no sub-populations and for the one with three sub-populations, respectively. However, for the Gap statistic, we set the upper boundary for all three different tree settings at 10 to limit the computational cost.

Within our algorithm, we have explored different linkage methods (single and complete linkage), different evolution models and different resolution level status (denoising with artificial levels and without artificial levels). We notice that these different settings have almost no effect on the results. Thus, we discuss one of these settings in detail in this section. We present the results for the case of building the tree with single linkage and denoising the node values by setting artificial levels. In the MDS step, we find that a relatively high dimension explains the data with no sub-populations better, so the dimension is set at 15 for both the first and second base tree with no sub-populations. For the tree which includes three sub-populations, the data can be explained with much lower dimension, so the dimension is set at four. We label our algorithms when we use compact-

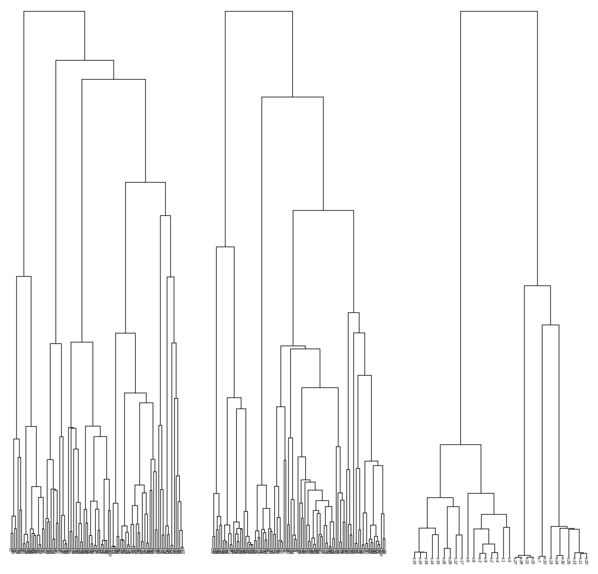


Figure 7.3: Simulation tree settings. The tree setting from left to right: the first and second base trees having mutation history (with no sub-population), and the population with a migration history (with three sub-population), respectively.

ness as a node value as Lifting-1, ALifting-1 and NLT-1; Lifting-2, ALifting-2 and NLT-2 are for the results of our algorithms when we use dissimilarity scores as a node value. While we set the threshold to zero in the ones labelled as Lifting, the algorithm picks the threshold automatically in the ones labelled as ALifting using the procedure described in Section 6.2. For NLT algorithms, we set the number of trajectories, P, to 100 and the threshold for the probability of acceptance, θ, to 0.5.

Simulation results for the first base tree with no sub-populations

Using the proposed algorithm for DNA sequences, given in Section 7.2, we compare the number of clusters found by different CVIs and Mclust. The detailed explanation of CVIs and Mclust can be found in Sections 5.3.1 and 5.3.2, respectively. Note that we do not know the true classification, so we can not check the performance of each method using external scores. Thus, we only compare how many different types of species exist. We do this comparison when we build the tree by the matching and phylogenetic distances. The comparison is done using box plots, and the results are given in Figure 7.4. We label each box plot with the median of the number of clusters found by each method, and the labels are placed on the right side of each box plot with blue color. Figure 7.4 illustrates that the CH and Sil indices fail to cluster species since they find the number of clusters either 50 or close to 50 which is the upper boundary of the number of clusters we set for these indices. A discussion of issues relating to the choice of the upper boundary can be found in Section 5.5. The Gap statistic also has the tendency to find one cluster, so it

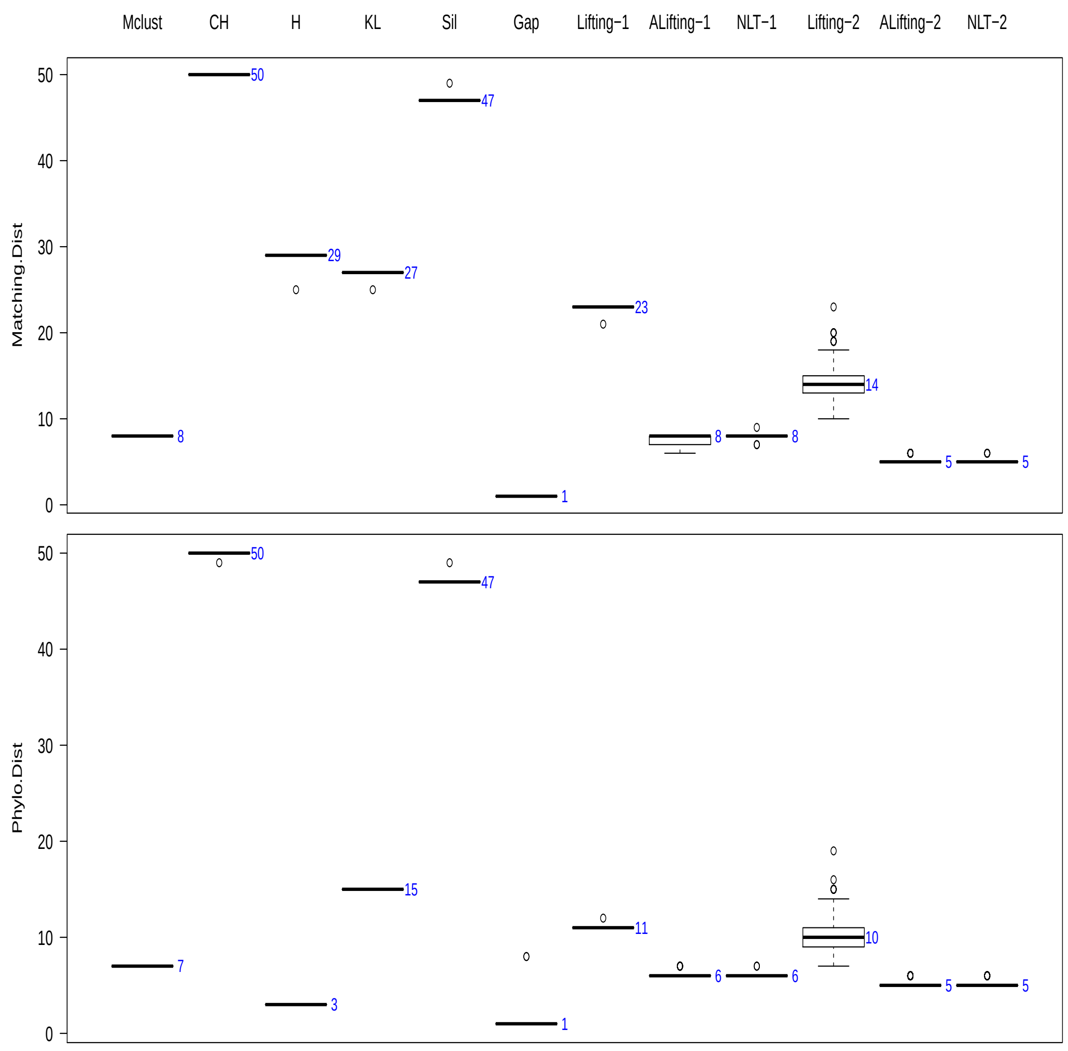


Figure 7.4: The comparison of number of clusters found by different methods for the first base tree with no sub-populations. The top row: matching distances are used to build the trees. The bottom row: phylogenetic distances are used to build the trees.

fails to cluster the species. When we build the tree using the matching distances, the H and KL indices also find a high number of small clusters. However, the partition found by these indices are quite different when we build the tree with phylogenetic distances. Mclust finds a similar clustering scheme with both the matching and the phylogenetic distances. Lifting algorithms find high number of clusters, but the number of clusters decreases when the algorithm thresholds the denoised node values (ALifting). When we build the tree with matching distances, the thresholds found by ALifting-1 and ALifting-2 algorithms vary around 15.69 and 0.39, respectively. The thresholds for the tree built by phylogenetic distances for ALifting-1 and ALifting-2 algorithms vary around 0.09 and 0.40, respectively. The final method is our NLT method. The comparison done by box plots clearly shows that the behaviour of our NLT algorithm is similar to ALifting algorithm. The only difference is that when we build the tree with matching distances, there are some variations on ALifting-1 results. However, these variations are extinct with

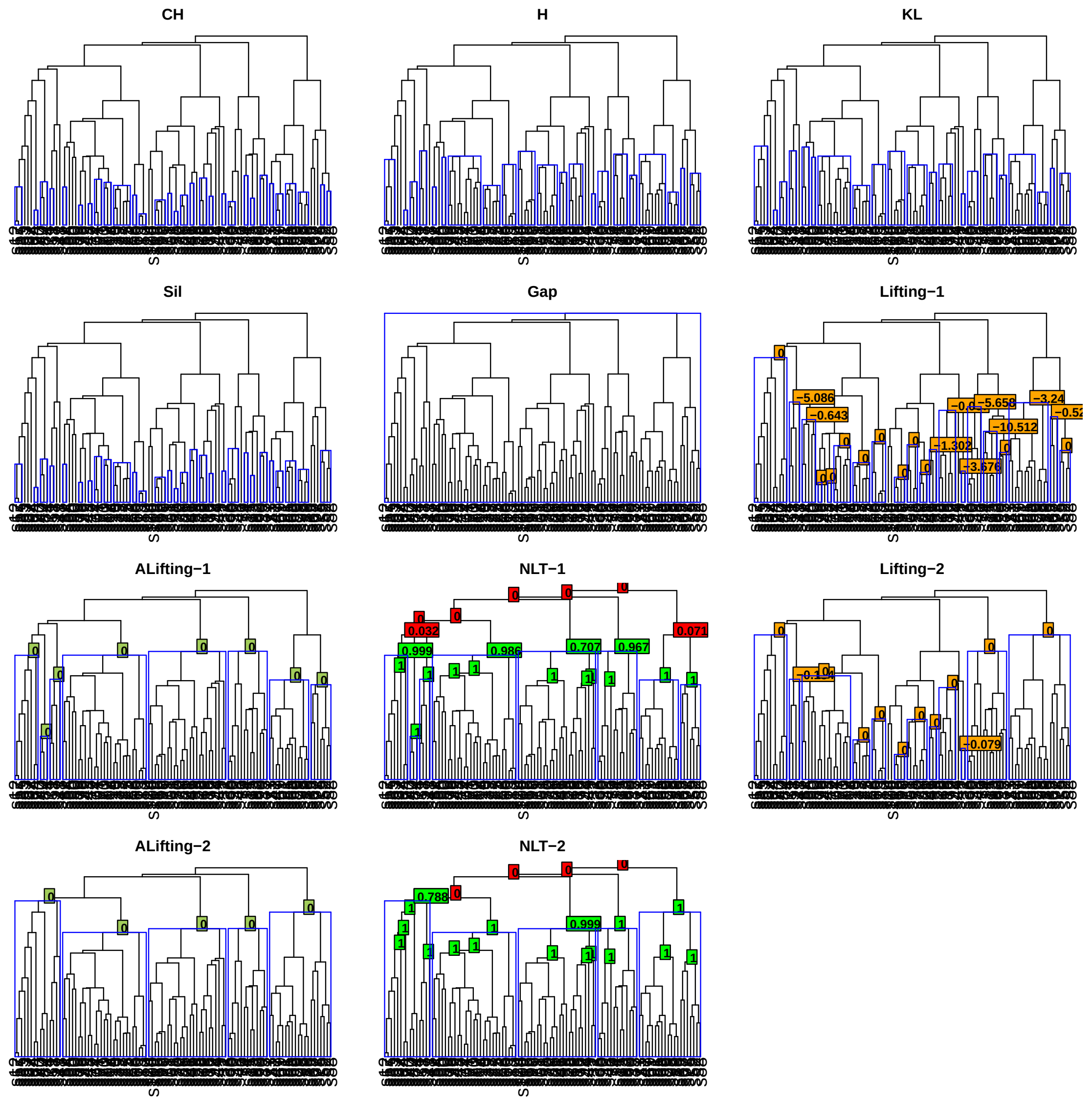


Figure 7.5: The comparison of clustering scheme found by different CVIs with matching distances for the first base tree with no sub-populations. Lifting-1 and Lifting-2: nodes are labelled with denoised detail coefficients. ALifting-1 and ALifting-2: nodes are labelled with denoised compactness and denoised dissimilarity scores, respectively. NLT-1 and NLT-2: nodes are labelled with the clustering probabilities colored with green, $p \geq \theta$, or red, $p < \theta$.

NLT-1 algorithm.

Figure 7.4 illustrates that each CVI finds the same number of clusters in almost all replicates, so there are few variations on the number of clusters. Thus, we can easily compare the partitions found by different CVIs by dendrograms. Dendrograms after building trees with matching distances and phylogenetic distances are shown in Figures 7.5 and 7.6, respectively. Since there are some variations on the number of clusters for Lifting-1 and Lifting-2 algorithms, we illustrate the partition found by the first replicate in these dendrograms. We also see exactly where clustering happens in these

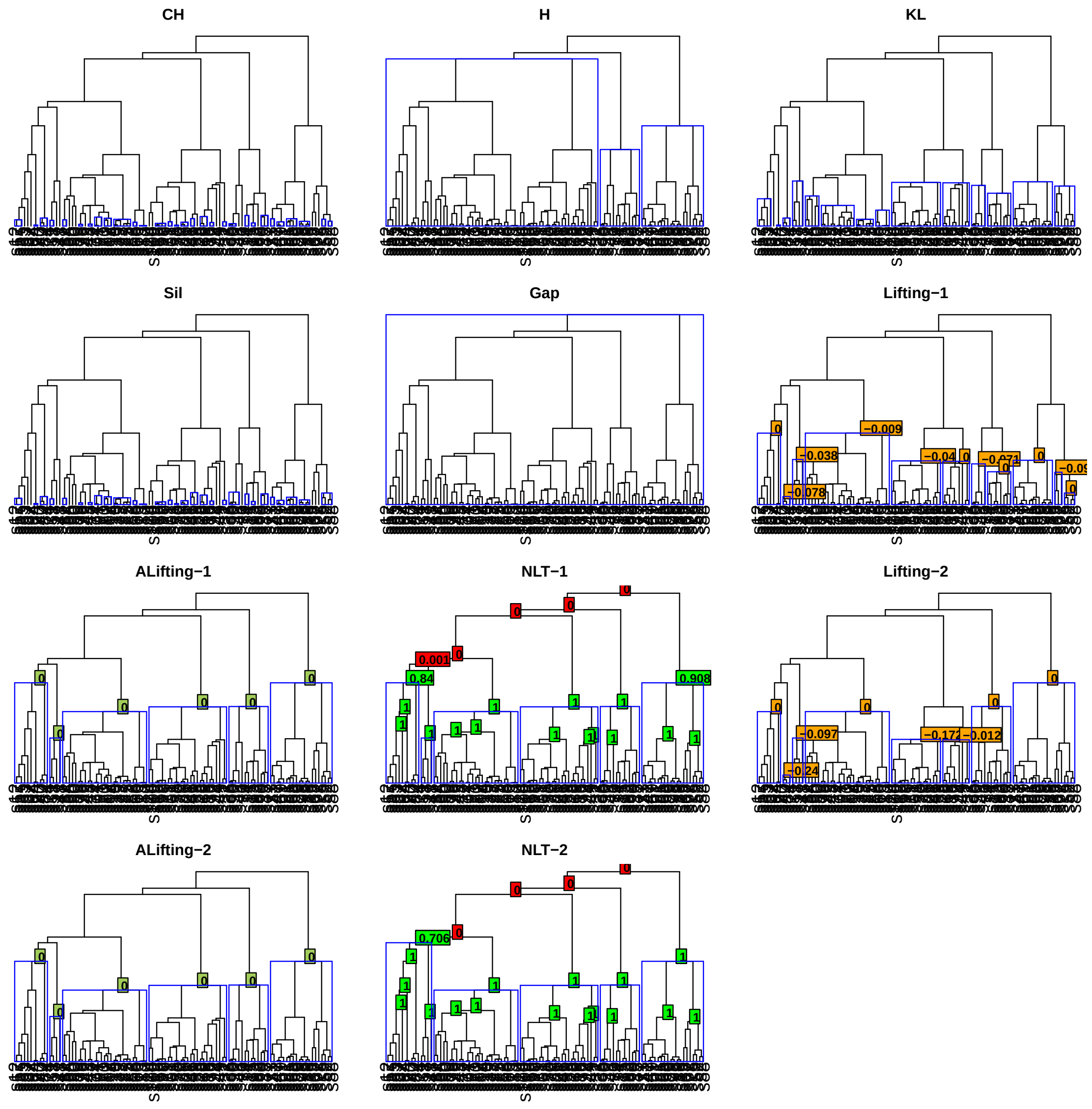


Figure 7.6: The comparison of clustering scheme found by different CVIs with phylogenetic distances for the first base tree with no sub-populations. Lifting-1 and Lifting-2: nodes are labelled with denoised detail coefficients. ALifting-1 and ALifting-2: nodes are labelled with denoised compactness and denoised dissimilarity scores, respectively. NLT-1 and NLT-2: nodes are labelled with the clustering probabilities colored with green, $p \geq \theta$, or red, $p < \theta$.

dendrograms.

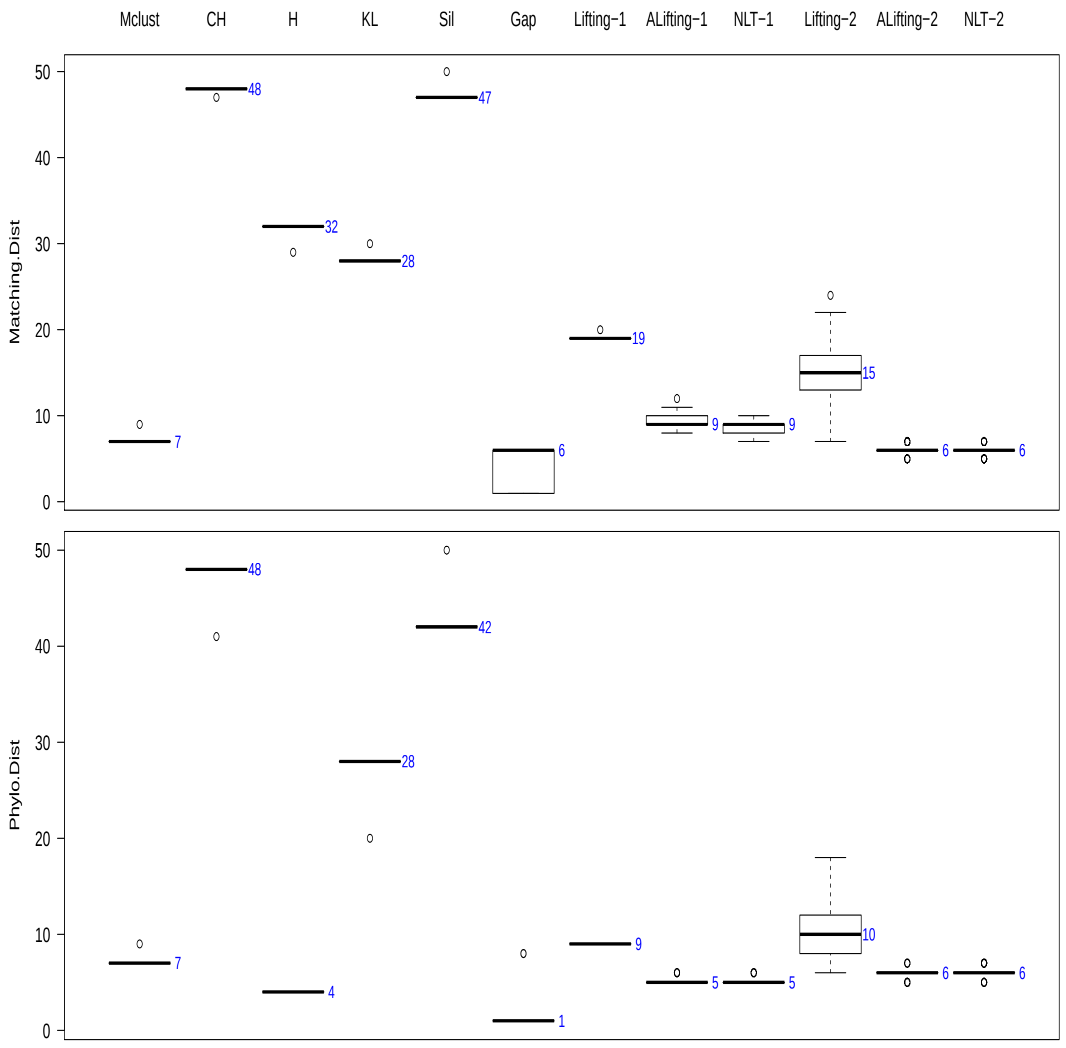


Figure 7.7: The comparison of number of clusters found by different methods for the second base tree with no sub-populations. The top row: matching distances are used to build the trees. The bottom row: phylogenetic distances are used to build the trees.

Simulation results for the second base tree with no sub-populations

We generate a new base tree using the same parameters with a different seed for the first base tree with no sub-populations. As it is seen in Figure 7.3, they have similar tree structures. Thus, we would like to check the behaviour of different methods on two similar tree structures. We repeat the comparison study done for the first tree structure with no sub-populations to this tree setting, and Figure 7.7 shows the box plot comparison for the number of clusters found by different methods. It is clearly seen that results are similar to the previous tree structure. Each method finds a similar number of clusters for both trees, and if any method fails to partition the data in previous tree setting, it could not partition this tree either. The behaviour of our methods are almost same in both settings.

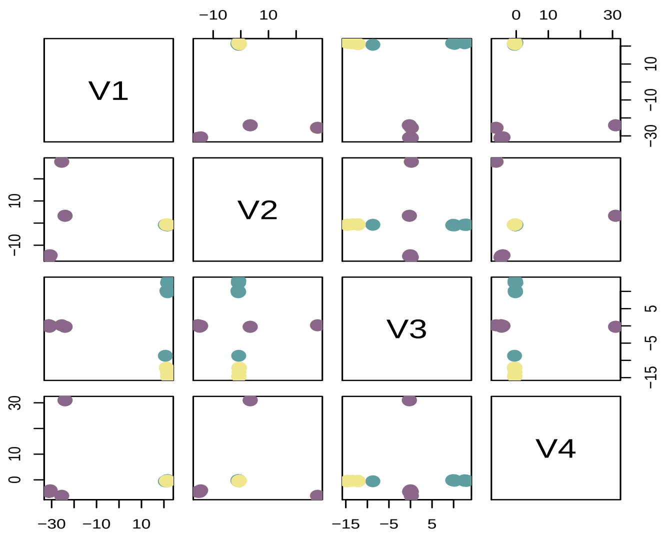


Figure 7.8: The comparison of pairs plots for the simulated population with three sub-populations. Different sub-populations are illustrated with different color, and matching distance matrix is used.

Simulation results with three abstract sub-populations

The final simulated data set is for a population with migration history. In this setting, we provide a comparison study similar to the ones we have done for multidimensional data sets (e.g. see Section 5.5). Using the sub-populations as components, we can check the performance of different clustering scheme found by different methods. We generate a population which has three sub-populations, and each of them is size of ten. We provide the multidimensional representation of the population in $\mathbb{R}^4$ in Figure 7.8.

First we build the tree using matching distances, and then we also repeat the study with phylogenetic distances to see how the methods are affected by different distance methods. When we build the tree with matching distances, the tabulated results of the average number of clusters in 1000 replicates is given in Table 7.6. The results clearly show that the CH index and Sil statistic fail to partition the data. Both of them found 19 clusters which is just below the upper boundary for these indices. Mclust captures the sub-populations with the highest performance, then the Gap statistic follows it. When our algorithm picks its threshold, the performance of ALifting-1, ALifting-2, NLT-1 and NLT-2 are exactly the same. Thus, we combine them in one column and label it as ALifting/NLT. Within our algorithms, Lifting-2 shows better performance than others, so setting the threshold, λ, to zero gives better performance than other version of our algorithms. This means that the variation of the data is high, so the universal threshold is overestimated. When we build the tree with phylogenetic distances, the performance of Mclust and ALifting/NLT do not change (see Table 7.7). However, the performance of other methods slightly in-

Index	Mclust	CH	H	KL	Sil	Gap	Lifting-1	Lifting-2	ALifting/NLT
N	5	19	5	8	19	4	7	7	3
Purity	0.933	0.967	0.886	0.945	0.967	0.866	0.898	0.960	0.733
ARI	0.629	0.112	0.577	0.482	0.111	0.582	0.470	0.528	0.491
AVI	0.668	0.242	0.634	0.563	0.236	0.642	0.553	0.624	0.592
CompCheck	0.607	0.095	0.623	0.456	0.094	0.665	0.457	0.470	0.778
ClustCheck	0.891	0.808	0.812	0.886	0.809	0.790	0.829	0.939	0.590
CC	0.736	0.277	0.706	0.613	0.276	0.716	0.613	0.663	0.677

Table 7.6: The comparison of CVIs with matching distances for the simulated population with three sub-populations. First row is for the average number of clusters, and others are for the average similarity scores. ALifting/NLT is for ALifting-1, ALifting-2, NLT-1 and NLT-2.

Index	Mclust	CH	H	KL	Sil	Gap	Lifting-1	Lifting-2	ALifting/NLT
N	5	19	5	6	19	5	4	4	3
Purity	0.933	0.967	0.933	0.936	0.966	0.931	0.936	0.937	0.733
ARI	0.629	0.122	0.629	0.604	0.112	0.625	0.722	0.706	0.491
AVI	0.668	0.253	0.668	0.650	0.238	0.666	0.732	0.723	0.592
CompCheck	0.607	0.103	0.607	0.581	0.095	0.607	0.709	0.691	0.778
ClustCheck	0.891	0.814	0.891	0.891	0.809	0.887	0.911	0.909	0.590
CC	0.736	0.287	0.736	0.715	0.277	0.733	0.804	0.792	0.677

Table 7.7: The comparison of CVIs with phylogenetic distances for the simulated population with three sub-populations. First row is for the average number of clusters, and others are for the average similarity scores. ALifting/NLT is for ALifting-1, ALifting-2, NLT-1 and NLT-2.

creases, and the best performance comes from Lifting-2. To see the partitions found by different methods, we provide the dendrogram comparison of different methods by highlighting the clusters for one replicate. The dendrogram which is built using phylogenetic distances are given in Figure 7.9, and the one with the matching distances are given in Figure 7.10. These figures clearly show that the performance of these algorithms is not affected by choice of distance method. Our ALifting/NLT methods combine two components on the right part of the dendrogram in one cluster, and this decreases its performance when we compare with the H index, KL index and Gap statistic. In addition, our NLT-1 and NLT-2 algorithms locate the clusters with high probabilities all the time. We label the top 10 nodes with the clustering probability which clearly show the high probability of being clustered. We can also check the pairs plots for the methods which find different partitioning for one replicate. As we discussed above the performance of the algorithms are not affected by different distance methods, so we just illustrate the pairs plots comparison for one setting. In addition, we exclude the partitions found by the CH index

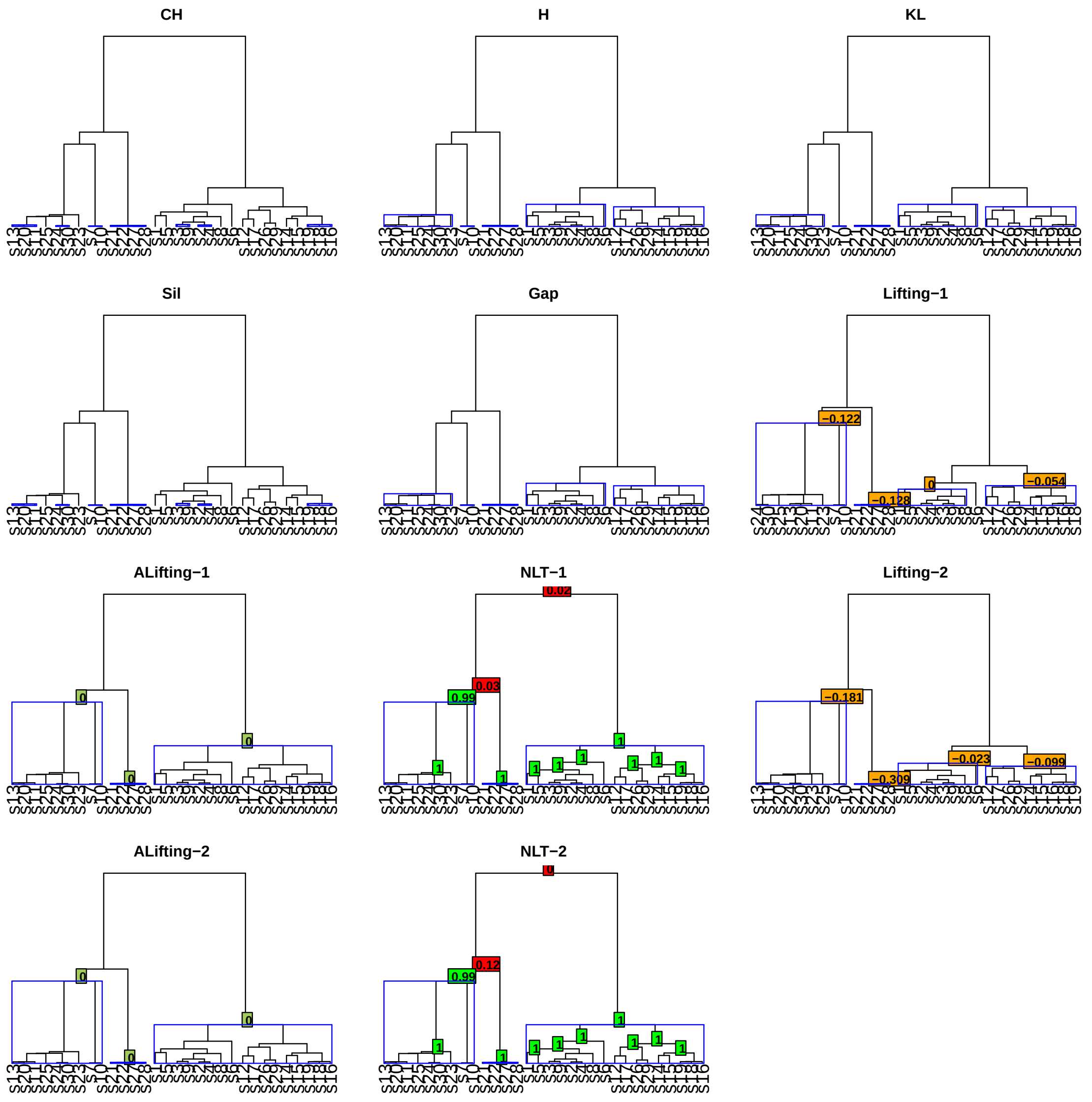


Figure 7.9: The comparison of clustering scheme found by different CVIs with phylogenetic distances for the simulated population with three sub-populations. Lifting-1 and Lifting-2: nodes are labelled with denoised detail coefficients. ALifting-1 and ALifting-2: nodes are labelled with denoised compactness and denoised dissimilarity scores, respectively. NLT-1 and NLT-2: nodes are labelled with the clustering probabilities colored with green, $p \geq \theta$, or red, $p < \theta$.

and the Sil statistic because of their failure of clustering, and we just provide one pairs plots for the ones find the same clustering pattern. Lifting-1 and Lifting-2 also behaves similarly, so we include the pairs plots for Lifting-1. This pairs plots comparison is given in Figure 7.11. The pairs plots clearly illustrate that two components are combined in one cluster by our ALifting/NLT methods, and Lifting-1 finds some small clusters because it does not allow any departure from the centroid of each cluster. Finally, Mclust and the Gap statistic show better performance than others by clustering different components into different clusters. They tend to cluster some species belong to one cluster into a separate

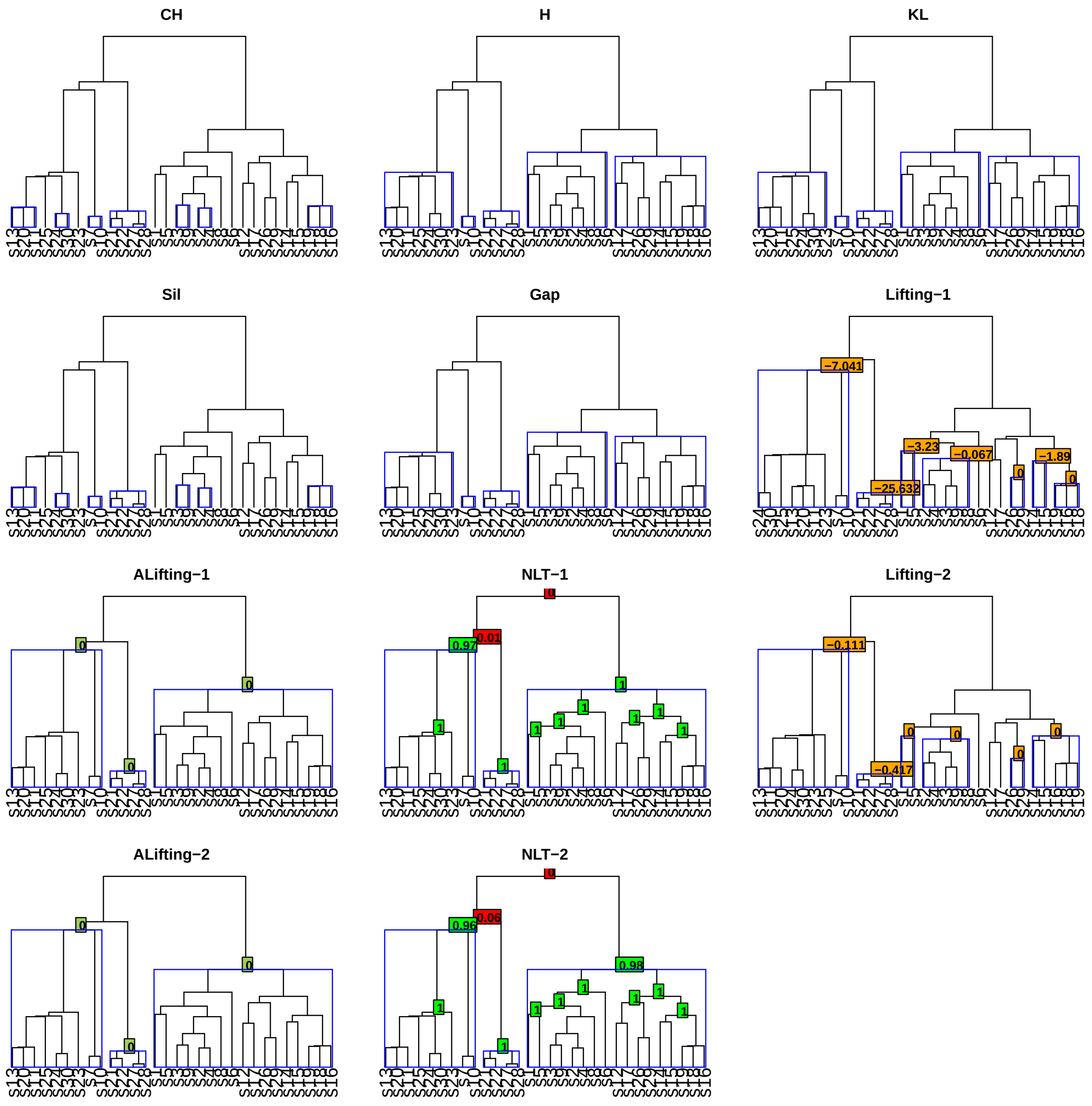


Figure 7.10: The comparison of clustering scheme found by different CVIs with matching distances for the simulated population with three sub-populations. Lifting-1 and Lifting-2: nodes are labelled with denoised detail coefficients. ALifting-1 and ALifting-2: nodes are labelled with denoised compactness and denoised dissimilarity scores, respectively. NLT-1 and NLT-2: nodes are labelled with the clustering probabilities colored with green, $p \geq \theta$, or red, $p < \theta$.

cluster if they locate further than the other species from their true cluster. To check the robustness of each method, box plots of external scores for each CVI are provided. This comparison is also done using two different distance measures in the clustering stage: Figures 7.12 and 7.13 illustrate the phylogenetic and matching distance results, respectively. The box plot comparison clearly shows that the CH index and Sil statistic always perform poorly, and the results of the KL index and Gap statistic have a high variation when we apply clustering with matching distances. Our Lifting-1 also has a high variation because of the threshold choice ($\lambda = 0$), but this variation disappears with the automatic threshold

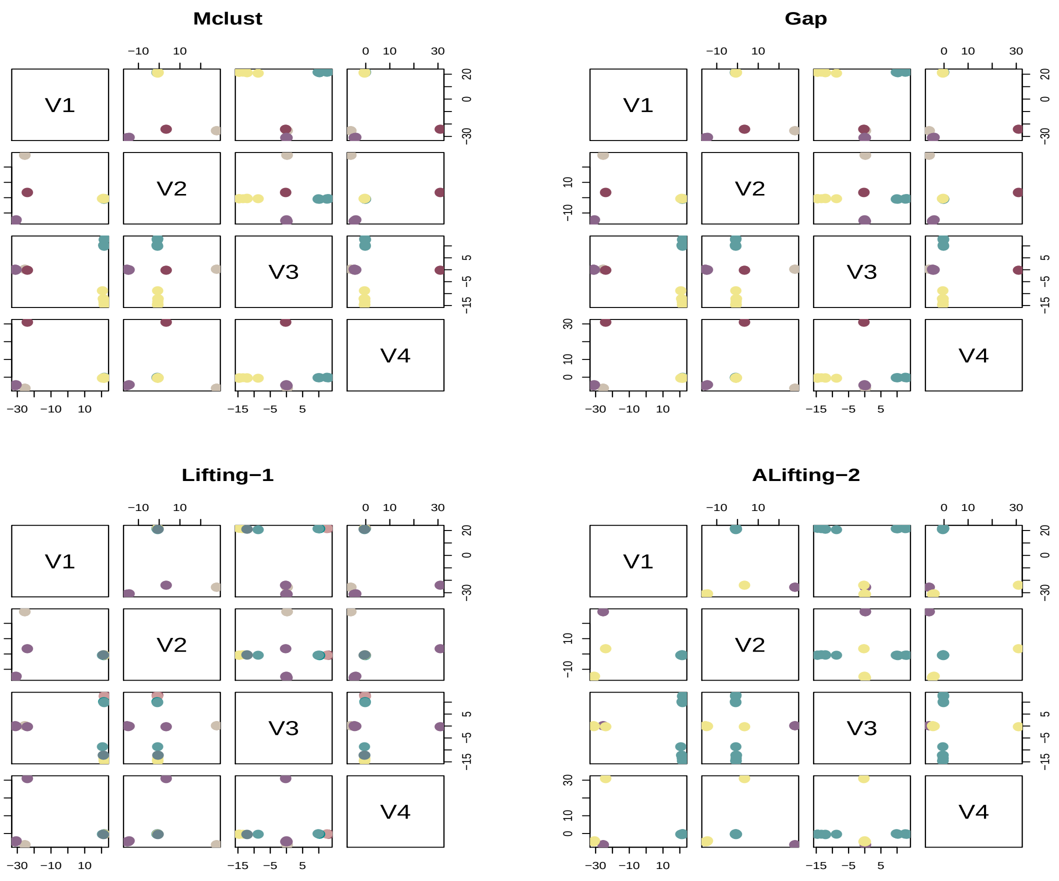


Figure 7.11: The comparison of pairs plots of the simulated population with three sub-populations. Matching distances are used in clustering. The one labelled as Gap is for the H index, the KL index and the Gap statistic, and ALifting-2 is for ALifting-1, ALifting-2, NLT-1 and NLT-2.

choice by ALifting/NLT.

The comparison study illustrates that Lifting-1 and Lifting-2 show the highest performance when we build the tree by phylogenetic distances, and ALifting/NLT has a low performance. This clearly shows that our variance estimate is not robust, so if we find a better variance estimate, we will have a better threshold rule for our ALifting/NLT algorithms. To see the choice of the threshold, we consider different variance estimates. The current version of our algorithm finds the variance estimate using sample variance defined in Equation (6.1) over all the nodes, including leaves, in the tree of interest. We compare the estimate of variance including and excluding leaves, and we also estimate the variance by MAD, given in Equation (2.41). We tabulate the results of ALifting-1 and ALifting-2 when we build the tree using phylogenetic distances in Table 7.8. The tabulated results clearly show that MAD estimates are close to zero, so if we use MAD estimate, the universal threshold will be close to zero which will create small clusters like Lifting-1 and Lifting-2. Thus, the sample variance estimate as we already use gives a better variance

		Including leaves		Excluding leaves	
	Methods	ALifting-1	ALifting-2	ALifting-1	ALifting-2
$\hat{\sigma}$	MAD	0	0	0.0008	0.045
	s	0.039	0.098	0.054	0.1265

Table 7.8: The comparison of different variance estimation settings.

Distance measure	ALifting-1	ALifting-2
Phylo.Dist.	$[0.106, 0.117]$	$[0.268, 0.289]$
Matching.Dist.	$[14.666, 15.268]$	$[0.267, 0.287]$

Table 7.9: The range of λ for different versions of ALifting algorithm. Phylo.Dist: phylogenetic distances, and Matching.Dist: matching distances.

estimate compared to MAD. We currently include leaves, and we discuss above that we overestimate variance in this way. However, excluding the leaves provides much higher variance. Thus, our way of finding variance estimate gives the best estimate within these different settings. To see the performance of our algorithm, we halve the variance estimate, and we notice that it slightly increases the performance of our algorithms. Thus, an estimate within the range $[s/2, s)$ may give a better threshold for our ALifting/NLT algorithms. The range of λ found by our ALifting algorithm with different distance measures and different node values are tabulated and given in Table 7.9. Thus, it is clearly seen that the threshold varies between each replicate, and finding the proper threshold estimate reduces the chance of small clusters.

Overall, the variation tends to be high in small data sets which means we overestimate the threshold, λ. The outcome of overestimation is to find an uninformative clustering pattern of the data of interest. Thus, we can consider using our Lifting method with zero threshold ($\lambda = 0$) which provides a better clustering pattern than our ALifting and NLT methods.

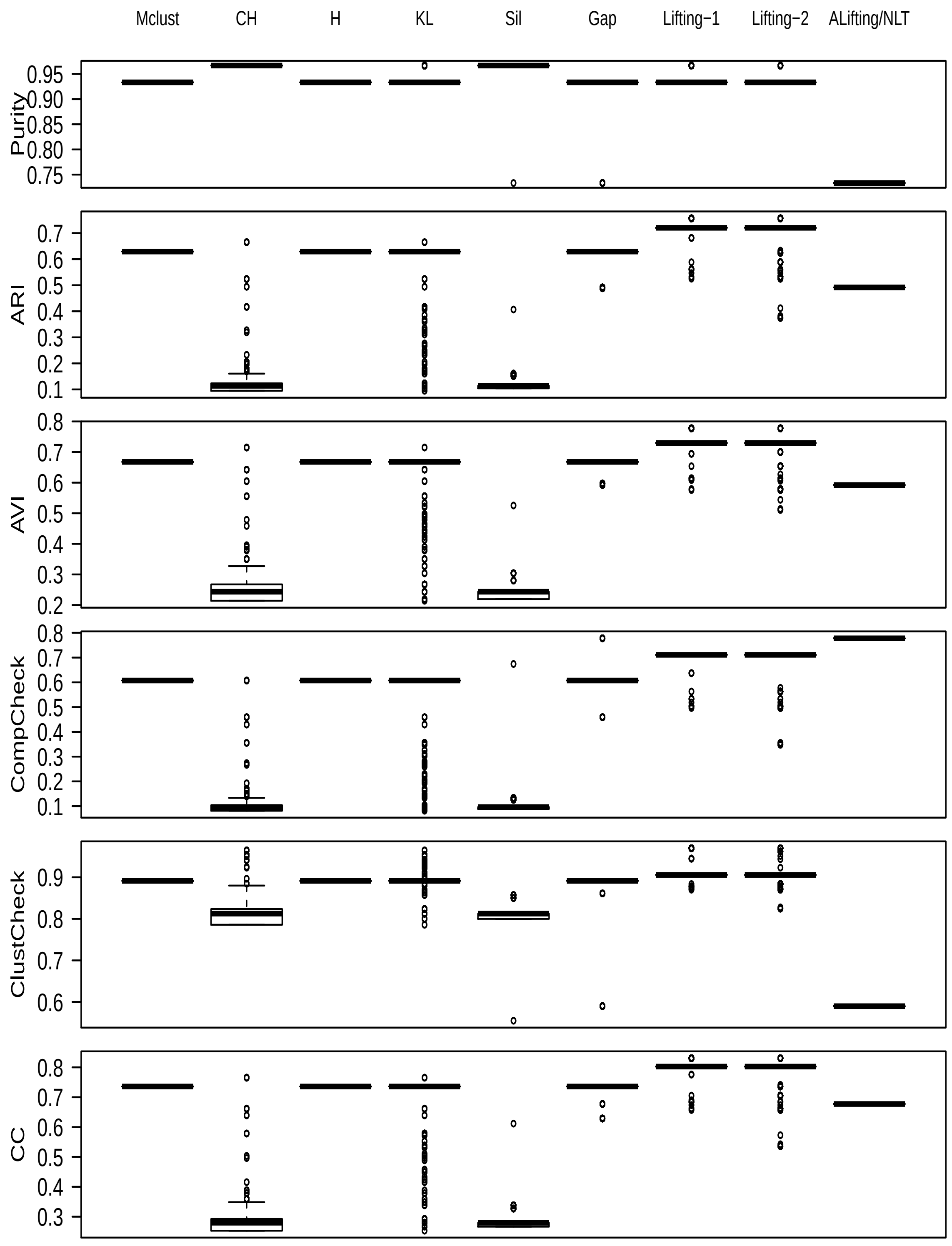


Figure 7.12: Box plot: the comparison of CVIs with phylogenetic distances for simulated population with three sub-populations. ALifting/NLT is for ALifting-1, ALifting-2, NLT-1 and NLT-2.

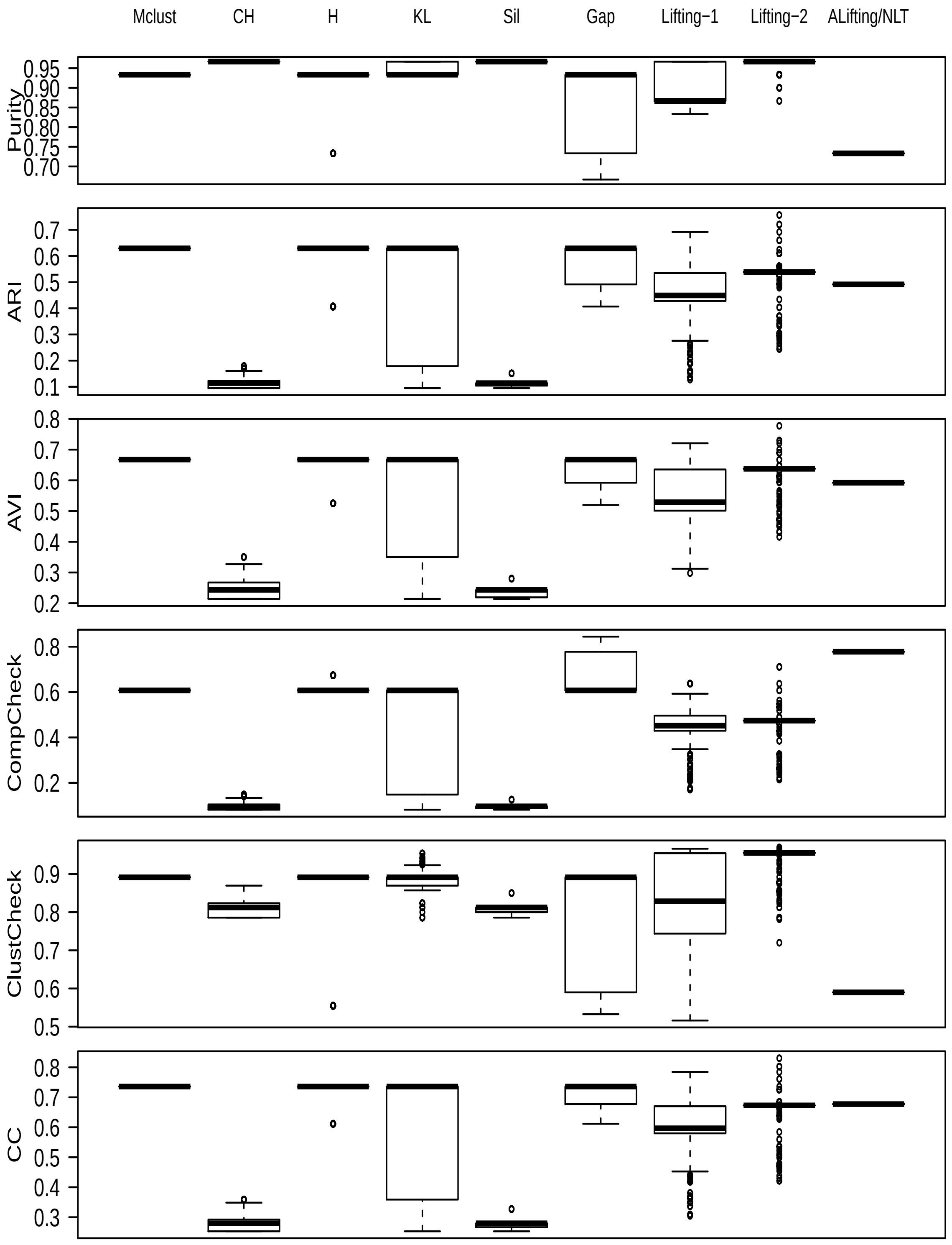


Figure 7.13: Box plot: the comparison of CVIs with matching distances for the simulated population with three sub-populations. ALifting/NLT is for ALifting-1, ALifting-2, NLT-1 and NLT-2.

Distance	Mclust	CH	KL	Gap	Lifting-1	ALifting-1	NLT-1	Lifting-2	NLT-2
Matching.Dist	9	17	39	13	14	11	10	13	8
Phylo.Dist	7	14	13	13	10	11	7	10	9

Table 7.10: The comparison of number of clusters found by different CVIs for the HIV-1 data. The column labelled as Gap is for the H index, Sil statistic and Gap statistic, and the column labelled as NLT-2 is for both NLT-2 and ALifting-2. Matching.Dist is for the matching distances, and Phylo.Dist is for the phylogenetic distances.

7.4 Real data

We also apply our Lifting algorithm to a real HIV-1 data set (Salazar-Gonzalez et al., 2009), which is available via GenBank (Benson et al., 2005) (accession numbers: FJ495818—FJ495826, FJ495937—FJ495943, FJ496000—FJ496007, FJ496072—FJ496085 and FJ496145—FJ496214). The data were collected longitudinally from 12 infected individuals, ten males and two females. Nine of the subjects come from the US (WITO, SUMA, WEAU, TRJO, 04013226, 04013396, CH40, CH58 and CH77) and three of them come from Zambia (ZM246F, ZM247F and ZM249M); the female subjects are from Zambia (ZM246F and ZM247F). In addition, the number of records for each subject varies with 108 full-length HIV-1 genome sequences in total. In the study, there are two types of viruses. While US subjects carry one type of the virus (type B), Zambian subjects have the second type of the HIV-1 virus (type C).

We analyze the HIV-1 data, using different CVIs and our Lifting algorithms. The HIV-1 sequences are not aligned, so we align using MUSCLE software before starting any analysis. In the MDS step, the dimension is set to 15, and the Kimura 2-parameter model is used in phylogenetic distance calculation. Note that when we build a dendrogram, we do not use exact branch lengths. Fixing the difference between each agglomeration stage at one unit helps us to see the clustering pattern easily.

Table 7.10 illustrates that the KL and CH indices find a large number of clusters when we build the tree with matching distances. The dendrogram (Figure 7.14) illustrates that these indices do not cluster some of the sequences.

The clustering scheme found by the H index, Gap statistic and Sil statistic is the same, and their behaviour does not change when we build the tree with phylogenetic distances. They cluster each subject in separate clusters, but they cluster one of the Zambian subjects (ZM247F) into two separate clusters. The behaviour of the CH and KL indices are the same as the H index, Gap statistic and Sil statistic when we build the tree with phylogenetic distances. Figure 7.15 illustrates that the only difference is that the CH index does not cluster one of the sequences for one of the US subjects (WITO). Mclust algorithm finds a different clustering scheme than other CVIs. As discussed in Section 5.3.2,

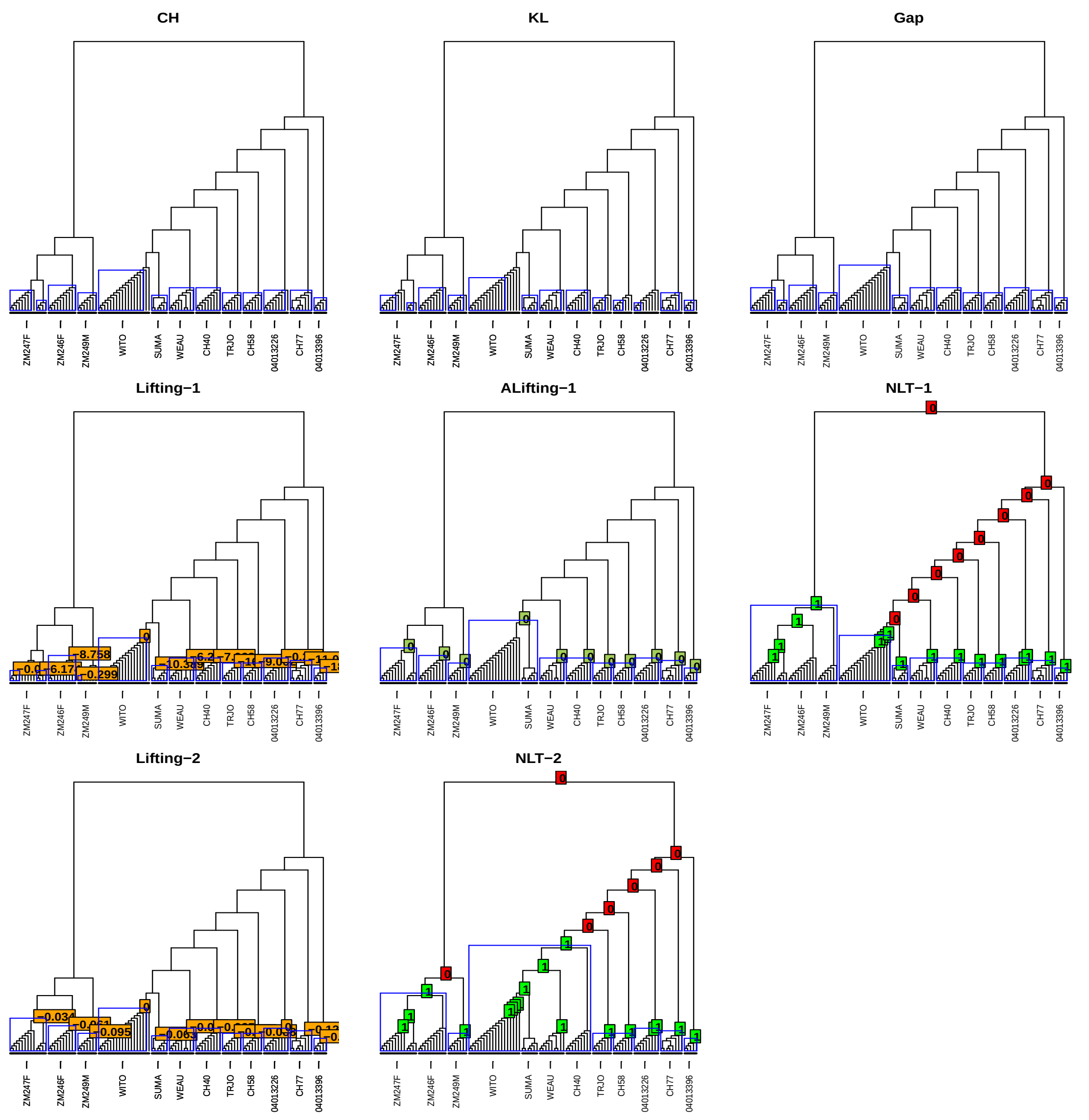


Figure 7.14: The comparison of clustering scheme found by different CVIs with matching distances for the HIV-1 data. Lifting-1 and Lifting-2: nodes are labelled with the denoised detail coefficients. ALifting-1: nodes are labelled with the denoised compactness values. NLT-1 and NLT-2: nodes are labelled with the clustering probability. The dendrogram titled as Gap is for the H index, Sil statistic and Gap statistic, and the one titled as NLT-2 is for both NLT-2 and ALifting-2.

Mclust is a model-based clustering algorithm while the other CVIs discussed in this study and our algorithm are based on hierarchical clustering. When we check the partitioning found by Mclust using matching distances, it clusters four of the US subjects together (CH58, 04013396, TRJO and SUMA) and clusters the other subjects individually. Mclust clusters six of the US subjects together (CH40, CH58, 04013396, TRJO, WEAU and SUMA) using phylogenetic distances.

Our Lifting-1 and Lifting-2 algorithms behave similarly to the H index and Sil statistic. Thus, they cluster each individual into separate clusters. Lifting-1 and Lifting-2 do

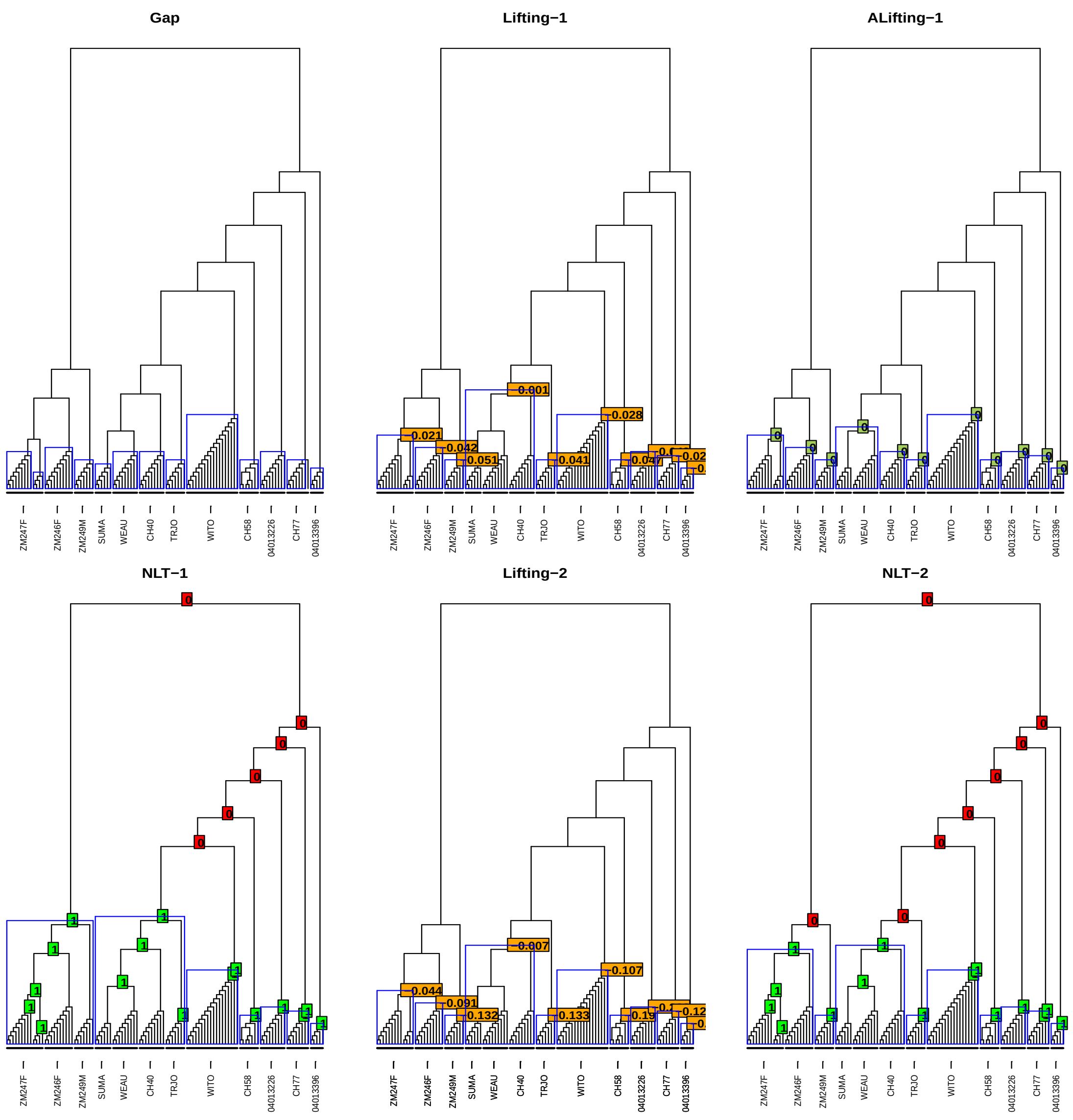


Figure 7.15: The comparison of clustering scheme found by different CVIs with phylogenetic distances for the HIV-1 data. Lifting-1 and Lifting-2: nodes are labelled with the denoised detail coefficients. ALifting-1: nodes are labelled with the denoised compactness values. NLT-1 and NLT-2: nodes are labelled with the clustering probability. The dendrogram titled as Gap is for the other CVIs (the CH index, H index, KL index, Sil statistic and Gap statistic), and the one titled as NLT-2 is for both NLT-2 and ALifting-2.

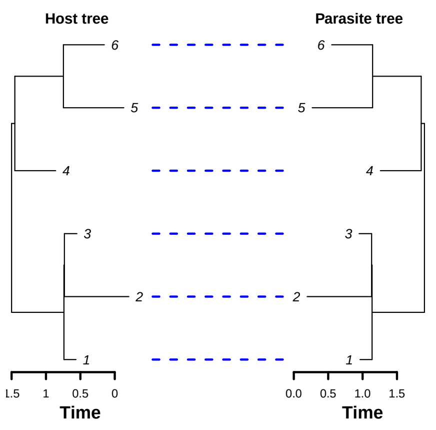
not cluster some of the sequences when we build the tree with matching distances, but our algorithm clusters all of the sequences by automatically picking its threshold. While ALifting-1 clusters subjects WITO and SUMA into one cluster and other subjects into separate clusters, the clustering scheme found by ALifting-2 and NLT-2 is slightly different. They cluster female Zambian subjects (ZM246F and ZM247F) together, and they cluster four US subjects together (WITO, SUMA, WEAU and CH40), and NLT-1 also behaves differently. It clusters all Zambian subjects into one cluster and each US subject into separate cluster. When we build the tree with phylogenetic distances, ALifting-1 clusters subjects SUMA and WEAU together. ALifting-2 and NLT-2 behave similar to the partitioning found by ALifting-2 with matching distances. The only difference is that three US subjects (SUMA, WEAU and CH40) are clustered together, so the subject WITO is also clustered separately in this version of the algorithm. The behaviour of NLT-1 is that next to clustering all Zambian subjects together, it also clusters four US subjects together (SUMA, WEAU, CH40 and TRJO). In addition, Figures 7.15 and 7.14 illustrate that our NLT algorithm always find the same clustering pattern whichever lifting order it follows, so the probability of placing a cluster at a node is either zero or one. When we use matching distances to build the tree, the thresholds found by ALifting-1 and ALifting-2 algorithms are $\lambda = 11.78$ and $\lambda = 0.12$, respectively. The thresholds for the tree built by phylogenetic distances for ALifting-1 and ALifting-2 algorithms vary around $\lambda = 0.04$ and $\lambda = 0.13$, respectively.

7.5 Applying Lifting to cophylogenetic data

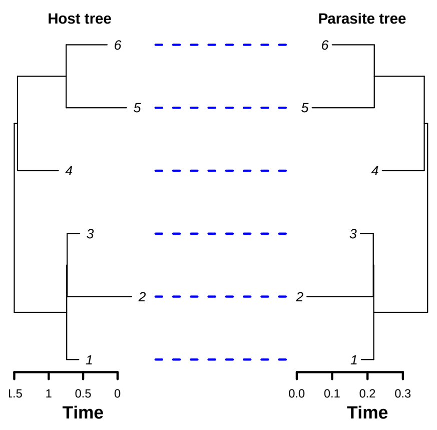
7.5.1 Introduction to cophylogeny

In previous sections, we worked on one phylogenetic tree, but one of the popular topics in evolutionary biology is to explore interaction between hosts and their parasites. Thus, hosts and parasites have interacting evolutionary trees showing cophylogenetic patterns. Fahrenholz (1913) proposed that the phylogeny of parasites mirrors their hosts, and this statement is known as Fahrenholz rule. Thus, it is assumed that there is synchronized speciation between parasites and their hosts (cospeciation) which means host and parasite trees are identical. However, there were no adequate methods to test the cospeciation of host-parasite relationships until Hafner & Nadler (1988) offered a test to identify if there is evidence of cospeciation between hosts and their parasites. There are also many other methods to test for cospeciation between hosts and their parasites. These methods are mainly divided into two groups: event-based methods and distance or topology-based methods.

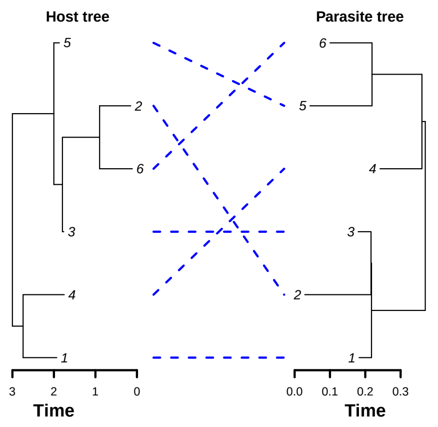
Event-based methods use different coevolutionary events to find the phylogenetic association between hosts and their parasites. The first event is cospeciation which occurrs when interacting trees have completely congruent phylogenies. Congruency means both hosts and parasites share an identical topology. Thus, two trees can be congruent even if the speciation of hosts and parasites occur at different times. There are, however, some conditions which reduce the congruency, and this reduction creates new coevolutionary events: parasites can connect with a new host (host-switch), a speciation can occur only on parasites (duplication), a speciation can occur only on hosts (inertia), or a parasite can go extinct (sorting). Before we review different event-based methods, we can illustrate



(a) Cospeciation, complete congruence.



(b) Congruence.



(c) Incongruence.

Figure 7.16: The illustration of coevolutionary events. Hosts are labelled from one to six, and their corresponding parasites share the same label as their hosts.

the different coevolutionary events on a toy data set. We randomly generate a host tree including six species, and for illustrative purposes, we create a copy of the host tree to act as the parasite tree. Thus, Figure 7.16a illustrates perfect cospeciation since both trees are identical in terms of both the topology and the time. We scale the branch lengths of the parasite tree and keep the same host tree. Figure 7.16b, hence, illustrates a congruence between host and parasites, but there is not a cospeciation event because they are not identical in terms of the time. There is also host-switch in this tree since parasites evolve much quicker than hosts. Finally, Figure 7.16c depicts the incongruence; they do not share the topology. If we carefully check this tree, we can observe each of the event separately in some part of the tree.

Event-based methods find which events have occurred in connected species. The first method in this group is the parsimony method proposed by Brooks (1988). Parasites are treated as character states of hosts, then the congruent and incongruent sections of the host-parasite phylogenies are determined. A second method, reconciliation analysis, was introduced by Page (1990), where the host-parasite association is taken as a linkage. This method is implemented in different tools; Component (Page, 1993) finds the best possible coevolutionary structure by minimizing the duplication, inertia and sorting events and maximizing the cospeciation event, but it ignores the host-switch event. Then TreeMap (Page, 1994) maximizes the cospeciation and minimizes the host-switch events, but it falsely models the events. Jungle (Charleston, 1998), implemented in TreeMap2, offers a better coevolutionary history of host-parasite taking the host-switch events into account. Another tool is Tarzan (Merkle & Middendorf, 2005) which allows the researcher to define the age of each node of the parasite tree, but it does not give the optimal coevolutionary history. Then Jane (Conow et al., 2010) is implemented which does not only consider the uncertainty of time on the parasite tree, but it also considers the uncertainty on the host tree. The next tool, TreeFitter (Ronquist, 1995), is the first implementation which gives a different cost function for each event, then it minimizes the total cost to find the optimal number of events which is occurred in the host-parasite interaction. The final method we describe in this group is a Bayesian approach (Huelsenbeck et al., 2000) which finds the coevolutionary relationship between hosts and parasites using their DNA sequences while the other methods we described assume that the phylogenies of the host and parasite trees are known. This Bayesian approach analyzes a mixture model for host-shift speciation and DNA substitution models to find a coevolutionary history of hosts and parasites.

The distance-based or topology-based methods test the hyphothesis that host and parasite trees are statistically independent. Distance-based methods were introduced by Hafner et al. (1994), who find the distance matrices of hosts and parasites using their aligned DNA sequences. Then they count the number of similar genetic events from these matrices. Finally, they apply a Mantel test to find if there is a significant correlation between host and parasite distance matrices. This method works for the statistically dependent case, but it can not deal with the phylogenetic dependency. Related species share the similar phenotypes with their last common ancestors or even with the earliest ancestor which take place in their evolutionary history, so this creates the phylogenetic dependency between species (Felsenstein, 1985b). Another method, ParaFit (Legendre et al., 2002), obtains the distance matrices from either DNA sequences or a built tree, and it can deal with any kind of coevolutionary events including the cases when a host has more than one parasite interaction. However, this method can not also deal with the phylogenetic dependency of a data set. Another method, introduced by Hommola et al. (2009), is a permutation test of the hypothesis that hosts and parasites evolve independently when host and parasite trees have an association linkage. They check the occurrence of cospeciation by calculating the Pearson correlation between host and parasite distances taking into account the host-parasite associations. Another distance based method, MRCAlink (MRCA for the most recent common ancestor; Schardl et al., 2008), evaluates the occurrence of cospeciation using the reduced distance matrices of parasites and hosts by removing the independent hosts and parasites from the matrices.

Next to bitrophic interacted trees, there are higher-order phylogenetic systems. Mramba et al. (2013) proposed a metod to explore the coevolutionary on three interacted phylogenetic trees. They only take the interactions which create a triangle into account, and they calculate p-values for each possible two trees. Recently, an improved version of their method was proposed by Nooney et al. (2017). Their method is applicable to higher-order systemes including phylogenetic networks, and they calculate one efficient p-value which is an improved version of the permutation statistic offered by Hommola et al. (2009).

Methods to evaluate evidence of coevolutionary history are summarized above. There are reviews which provide detailed comparison of different methods provided by Vienne et al. (2013) and Filipiak et al. (2016). Vienne et al. (2013) concluded that while event-based methods test the congruence between host and parasite trees, distance-based methods test the independence or similarity of trees. They also concluded that host-shift events are more likely to occur than cospeciation.

7.5.2 Application

In Section 7.2, we proposed the application of our Lifting method introduced in Section 5.4 to phylogenetic trees. Since our method is applicable to phylogenetic trees, we can apply it to interacting trees as an explanatory tool to investigate which clades of a phylogenetic system may show signs of cospeciation. In this study, we only focus on bitrophic trees. We assume that host and parasite trees are two independent trees, and apply our Lifting method separately to both trees using phylogenetic distances. When we built the trees with hierarchical clustering, the tree topologies were quite different from the ones generated in previous studies. Hence, we build the trees using the neighbour-joining (NJ) algorithm, described in Section 4.3.2 which is available in **ape** (Paradis et al., 2004) in **R** (**nj()** function). Using our algorithm, we aim to supply a visualization tool which shows where the related species of hosts and related parasites are located on their trees, and we link the clusters on the host tree with the clusters on the parasite tree if these clusters include parasites of the clustered hosts. Hence, we aim to check if there are any pattern between host and parasite trees in terms of congruency. The results of analyzing two well-known cophylogenetic data sets in the literature are presented in this section.

Pocket gophers and their chewing lice

One commonly used data set is pocket gophers and their chewing lice (Hafner et al., 1994). Pocket gophers live the most of their life alone in tunnels, so they do not interact

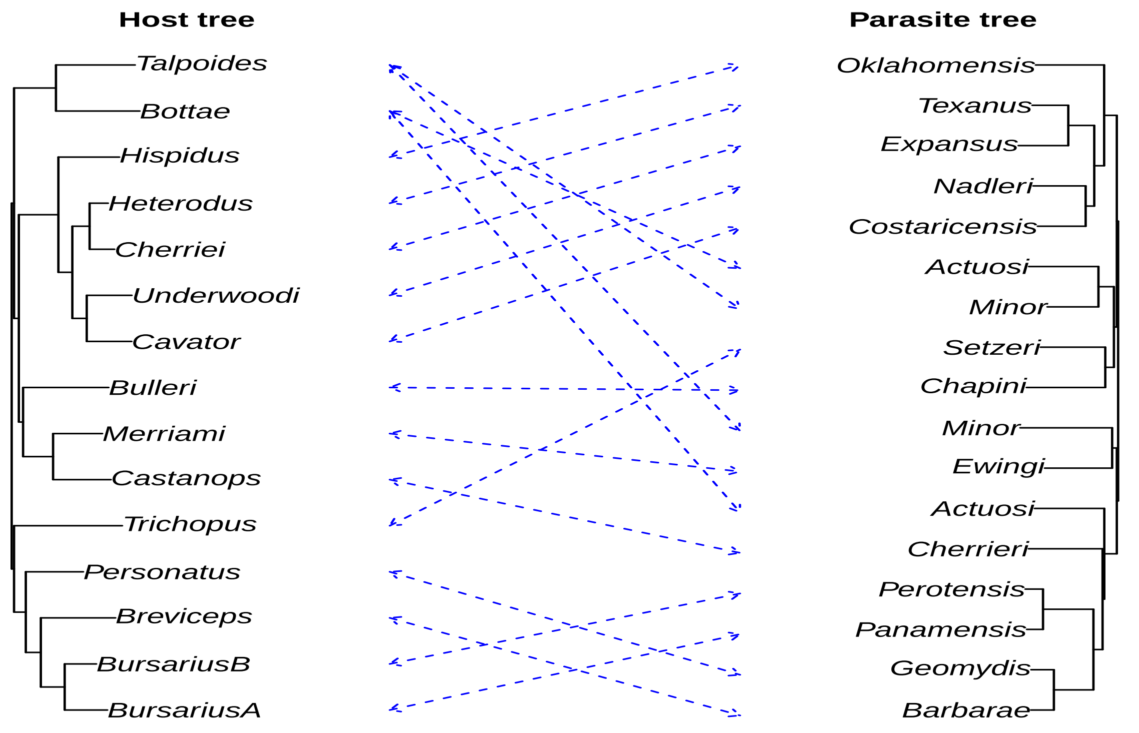


Figure 7.17: Phylogenetic trees of pocket gophers (hosts, left) and their chewing lice (parasites, right) with their interactions shown by blue dashed lines.

much with other gophers which decreases the chance of host-shift event. This data set played significant role in the development of cophylogenetic analysis, and it includes the DNA sequences of length 379 for a particular gene (mitochondrial cytochrome oxidaze subunit I (COI)) of 15 pocket gophers and their 17 chewing lice. This data is available via GenBank (Benson et al., 2005) (accession numbers for hosts and parasites: L32682.1 — L32696.1 and L32665.1 — L32681.1, respectively). Phylogenetic trees of pocket gophers and their chewing lice are given in Figure 7.17, and their interactions are shown by blue lines.

In this data set, we have only a portion of the DNA sequence of each specie, so variation between species is low. In this case, if we use our ALifting algorithm which automatically picks the threshold, we are likely to overestimate the threshold because of the lack of differences between sequences, which probably leads us to have one big cluster per tree. Thus, it is more appropriate not to allow any departure from the centroid of each cluster or any variation between sequences belonging to the same cluster after denoising if we define the function values as compactness values or dissimilarity scores, respectively. We, therefore, produce results for our Lifting algorithm by setting the threshold, λ, to zero and using phylogenetic distances with the Kimura-2 parameter model. Different function values, dissimilarity scores or compactness values, do not materially change the outcome of our Lifting algorithm in this data set. Phylogenetic trees of pocket gophers and their

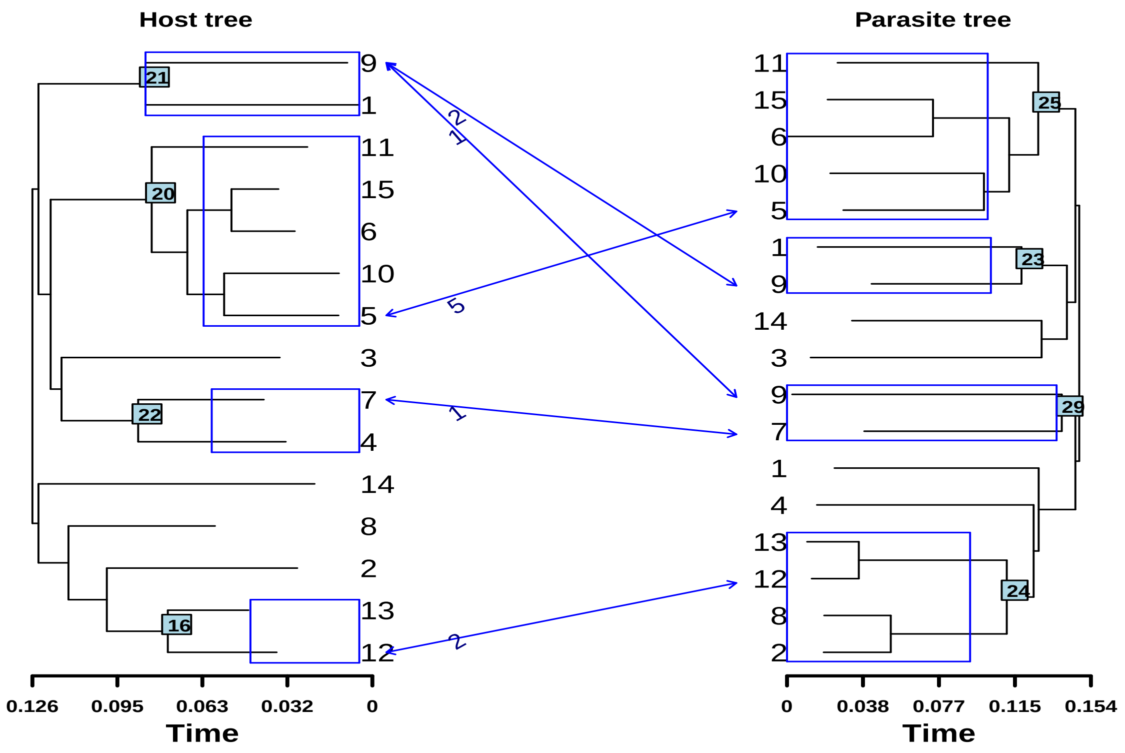


Figure 7.18: Clustered pocket gophers (hosts, left) and their chewing lice (parasites, right) by our Lifting algorithm. Clustered nodes are labelled with the agglomeration order starting from $n + 1$, where n is the number of species.

chewing lice with our Lifting results are presented in Figure 7.18. In this figure, instead of labelling tips with species' names, we number them. There are 15 pocket gophers, so we enumerate tips from 1 to 15. We also give the same numbers to the chewing lice which infest each pocket gopher. For example, the chewing lice which infest the pocket gopher numbered as one are labelled as one. Note that there are 17 chewing lice because two pocket gophers are infested by two different chewing lice. These are the ones labelled as "9" and "1", so two parasites are labelled as "1" and another two of them as "9". We link clusters between trees with a labelled blue arrow if the cluster on the parasite tree includes any of the parasites of the species in the cluster on the host tree, and the labels on the arrows illustrate how many species are shared in the connected clusters. If there are no shared species, those clusters remain unconnected. Our Lifting algorithm finds a similar clustering pattern on both trees. If we look carefully, we will see that if a node on the host tree has the same speciation structure as a node on the parasite tree, a cluster is located at those nodes. For example, nodes 20 and 21 on the host tree have the same speciation structure as nodes 25 and 23 on the parasite tree, respectively. Thus, there is congruence in these parts of the trees. In addition, another cluster located at node 16 on the host tree is connected to node 24 on the parasite tree. Even though the cluster on the parasite tree includes two more parasites, the parasites of the corresponding hosts are branched together

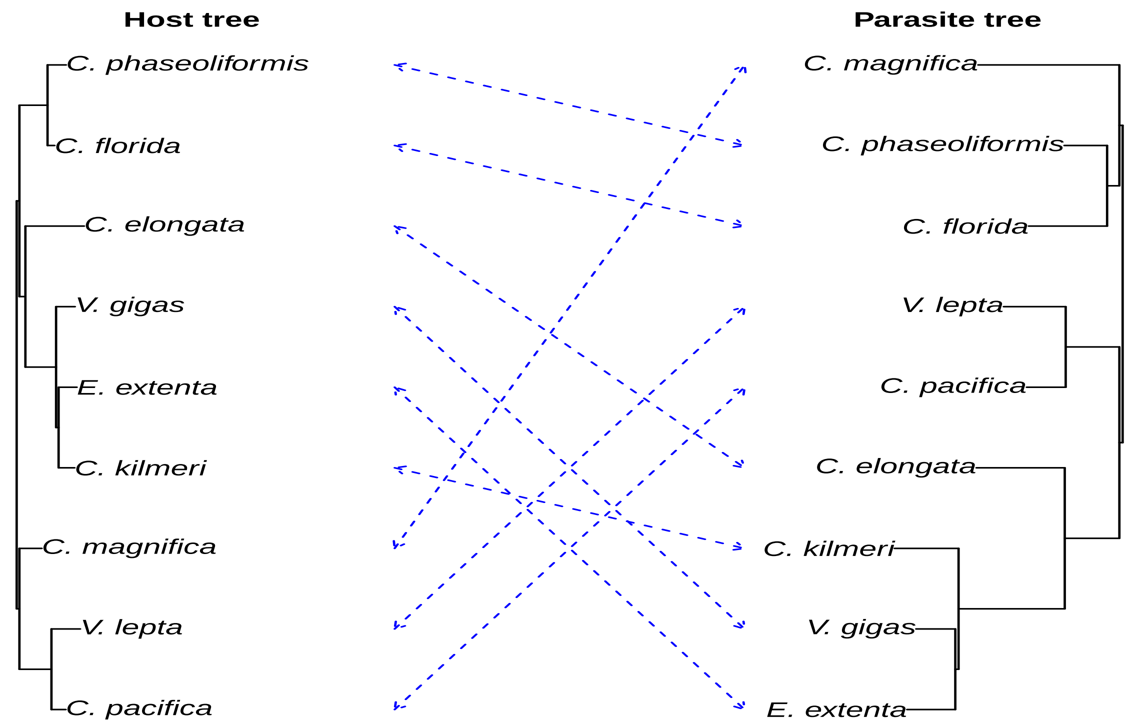


Figure 7.19: Phylogenetic trees of deep sea clams (hosts, left) and their bacteria (parasites, right) with their interactions shown by blue dashed lines.

and are clustered together. Overall our algorithm finds a clustering pattern on the parasite tree which highly mirrors the clustering pattern on the host tree, and identify the pasts of the tree which show evidence of cospeciation.

Deep sea clams and their sulfur-oxidizing bacteria

The second cophylogenetic data set is about deep sea clams and their sulfur-oxidizing endosymbiotic bacteria, published by Peek et al. (1998). Deep sea clams need sulfur-oxidizing bacteria for their nutrients which are transmitted via eggs. Thus, a strong cophylogeny pattern is expected. In this data set, there are 9 clams and their 9 bacteria, and the mitochondrial COI DNA sequences of length 516 for the clams and the small subunit (16S) sequences of length 1433 for the bacteria are provided which are available via GenBank (Benson et al., 2005) (accession numbers for hosts: AF008274, AF035941, AF008272, AF008281, AF008295, AF008283, AF008266, AF008264, AF035942, and for parasites: AF035719 — AF035727). Phylogenetic trees of the deep sea clams and their bacteria are given in Figure 7.19, where their interactions are shown by blue lines. We present the clustering pattern found by our Lifting algorithm with the same settings with the pocket gophers and their chewing lice data set for this cophylogenetic data set. Phylogenetic trees after applying our Lifting algorithm are given in Figure 7.20. We find that if the hosts and parasites share the same speciation structure, they are clustered to-

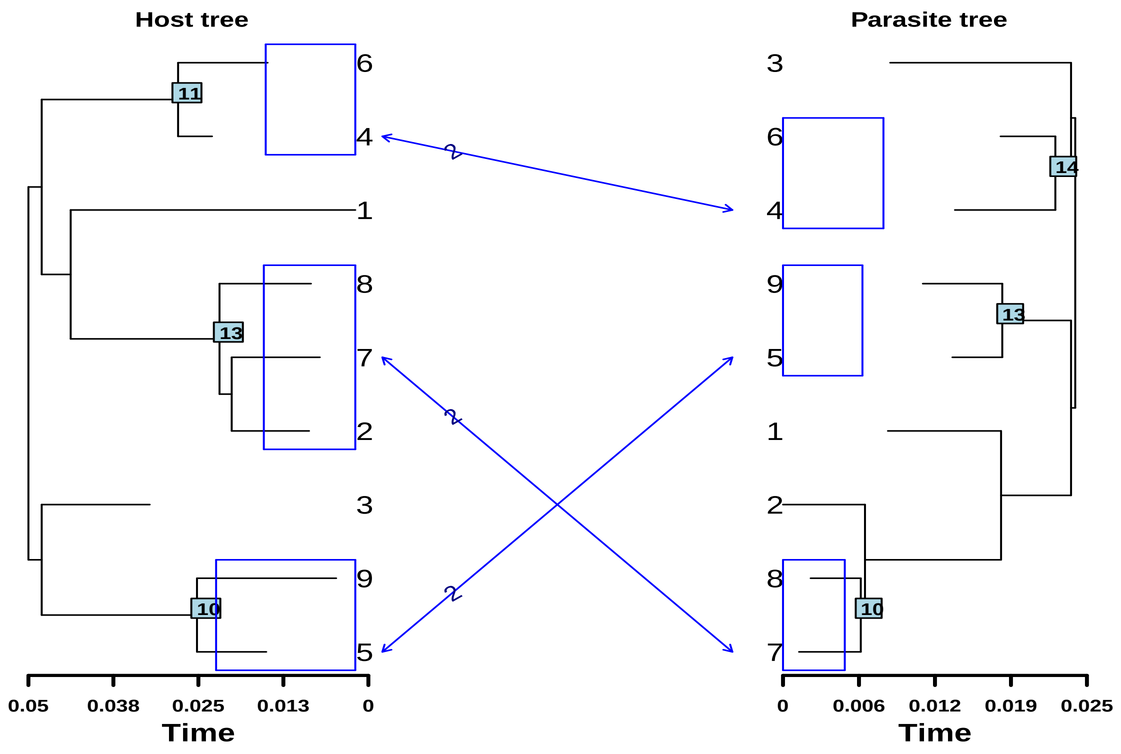


Figure 7.20: Clustered deep sea clams (hosts, left) and their bacteria (parasites, right) by our Lifting algorithm. Clustered nodes are labelled with the agglomeration order starting from $n+1$, where n is the number of species.

gether as it happened for the pocket gophers and their chewing lice data set. Nodes 11 and 10 cluster the hosts of parasites at nodes 14 and 13, respectively, where these hosts and parasites are branched in the same way as each other. The speciation which is happened at node 13 on the host tree is different than the one at node 10 on the parasite tree, but we find that there is not such a large difference between the clams labelled “7” and “8” and their bacteria. Thus, we also find a similar clustering pattern for the clams and their bacteria.

7.6 Summary

In Section 5.4, we discussed how we can apply the lifting algorithm to a clustering of a multidimensional data set, and in this chapter, we propose a way of applying our Lifting algorithm to phylogenetic data sets (DNA sequences). To apply our Lifting algorithm, we need three basic inputs: joined pairs, the distances between nodes, and node values. The first two features come from the hierarchical clustering step, and we proposed a node value called compactness in Section 5.4. For phylogenetic data sets, we can still use this type of node value after assigning coordinates for each sequence using MDS. We also introduce another node value for phylogenetic data sets: dissimilarity scores. Basically, we check each locus for all sequences under the node of interest in turn, and we count the number of loci having different nucleotides. We also examine the effect of different distance computing methods (matching and phylogenetic distances) to our Lifting algorithm in this chapter.

The application of our Lifting algorithm on phylogenetic data sets is illustrated using simulated data sets and a real data set. We simulate DNA sequences using **Seq-Gen** (Rambaut & Grassly, 1997), but we need a base tree to generate the sequences. We simulate a base tree using **ms** (Hudson, 2002). In the simulation study, we create three base trees. Two of them are generated using the same parameters with different seeds, and we just take mutation into account as an evolution history. The aim is to see how different trees can be generated with the same parameters, and how our Lifting algorithms behave on these trees. In the third tree, we consider migration along with mutation as an evolution history. The structure of these trees are given in Figure 7.3. We find that the choice of evolution model in the sequence generation step does not affect the performance of our Lifting algorithms. Thus, we only discuss the case when we generate sequences under the Kimura 2-parameter model.

For two trees with no sub-populations (having only mutation history), the comparison study is done using different internal cluster validity indices (CVIs) and Mclust explained in Sections 5.3.1 and 5.3.2, respectively. We observe that the CH, KL and H indices and Sil statistic fail to cluster the data. They find a high number of small clusters. The number of clusters found by the CH index and the Sil statistic are close to the upper boundary choice for these methods. Thus, these results clarify the importance of the upper boundary choice for these methods. The H and KL indices perform better when we build the tree with phylogenetic distances. While the KL index still finds small size clusters, the H index performs very differently, and it clusters diverged species into the same cluster (see Figure 7.6). The Gap statistic also attempts to cluster diverged species together, but when we build the second base tree having mutation history with phylogenetic distances, it finds the same clustering structure as ALifting-1. Our Lifting algorithms are not affected much by different distance computing methods. When we set the threshold to zero (Lifting-1 and Lifting-2), we find a high number of small clusters with some outliers. When the algorithm automatically picks the threshold (ALifting-1 and ALifting-2), our algorithm finds more compact clusters (see Figures 7.5 and 7.6). Our NLT-1 and NLT-2 algorithms find the same classification with high probabilities as our ALifting-1 and ALifting-2 algorithms, respectively.

The final simulated tree-structure illustrates a population which has migration and mutation history, and we use sub-populations as true components for this data structure. Thus, we can check the performance of each method using external scores described in Section 5.3.3. We notice that the choice of distance measure has a significant role for this data structure. When we build the tree with matching distances, results of the Gap statistic and KL index have a high variation. However, they cluster sub-populations with a high performance by building trees with phylogenetic distances. For the CH index and Sil statistic, results do not change; they fail to partition this data structure. Mclust always clusters the data with high performance (ARI$\approx$ 63%), but our Lifting-1 and Lifting-2 show the highest performance when we build the tree with phylogenetic distances. We notice that ALifting/NLT overestimate the threshold, λ, for small data sets because the variation between sequences are low, so our Lifting algorithms by setting λ to zero perform better.

Another comparison study is done for a real HIV-1 data set (Salazar-Gonzalez et al., 2009). While CVIs attempt to cluster each individual in separate clusters, Mclust and ALifting/NLT find different partitions. Here, the question was if we would like to partition the data in terms of their gender or nationality or different subjects, or if we would like to find where mutation happened in time. Using ALifting/NLT, we find the possible point in the time where speciation happens. Thus, we believe that this can be helpful for other scientists doing genetic based studies. This study can be helpful to track possible speciation in time.

Finally, we examine the behaviour of our algorithm on cophylogenetic data sets using two different real data sets: pocket gophers and their chewing lice (Hafner et al., 1994) and deep sea clams and their sulfur-oxidizing bacteria (Peek et al., 1998). In this part of the study, we demonstrate the clustering pattern found by both host and parasite trees. Our Lifting algorithm automatically clusters species together if any group of hosts shares the same branching pattern with their parasites. Thus, this visualization tool will be helpful for a researcher who wishes to investigate the congruency between hosts and parasites in a data set where cospeciation has been detected.

Chapter 8

Discussion

In this thesis, we have proposed a new algorithm which automatically detects where clustering happens in a dendrogram by denoising some generalized node values. We generalized the algorithm which could be used on any multidimensional data set by defining the node value as the average distance from the centroid of each possible cluster (compactness), or it could be applied to phylogenetic data sets (DNA sequences) either using compactness or dissimilarity score as a node value. The dissimilarity score was defined as the average number of loci in which at least one of them does not share the same nucleotide (excluding gaps) between sequences under the node of interest.

Our proposed algorithms were discussed in Chapters 5 , 6 and 7. In the first part of Chapter 5, we also discussed some internal cluster validity indices (CVIs) in the literature which find the number of clusters for hierarchically built trees. Even though our focus was hierarchically built trees, we also included mixture model-based clustering (Mclust) in our simulations to see the performance of another clustering method in comparison to our algorithms. We also presented some available similarity scores which check the performance of different methods if we know the true partitioning of a data set.

Within different CVIs, we noticed that the CH index had a higher performance than other indices in terms of capturing true components, but when there was a high variation in a data set or overlapping components, it showed a low performance. It tended to find a high number of clusters. Its performance in high dimensional data sets was also low since it found many small size clusters. Overall, all indices captured true components with a high performance if they were well separated, and the data were defined in low dimensions. In opposite cases, none of them easily partitioned the data: the Gap statistic tended to find one big cluster, and other CVIs tended to find a high number of clusters. Even though this behaviour of the Gap statistic is applicable to overlapping components, we observed that it does not even guarantee to show a high performance for well-separated data sets. In addition, all CVIs we are aware of find the number of clusters in a data set, so by cutting a tree from any height which gives the found number of clusters, we can reach the clustering pattern. We also noticed that during the implementation of CVIs, we need to give an upper boundary for the number of clusters. We recognized that the different upper boundary choices change the clustering pattern.

When we checked the performance of Mclust, we noticed that its performance was the highest one if the components could be explained by the normal distribution. However, it tended to cluster a high number of objects from different components together for uniquely shaped data sets such as concentric circles or the non-normally distributed data sets (see Section 5.5). Mixture model-based clustering may even deal with these data structures if a new model is defined, and if the clustering algorithm based on the new model is implemented which increases the computational complexity of the algorithm, and it requires a new model to be specified for each data set.

In the clustering literature, there are also many cluster validity scores, and we discussed some of them in Section 5.3.3. CompCheck and ClustCheck (Fowlkes & Mallows, 1983) had a poor performance alone because CompCheck only looked at if the objects from the same component were clustered together in any number of clusters. If all the objects are clustered together, CompCheck will be equal to one, so it can not be used alone to check the performance of any index. Thus, using CC, the product of CompCheck and ClustCheck, is a better choice. The benefit of CC is that it does not penalize small clusters, so if a component is divided into small clusters without mixing elements from different components, it will return a high score which is appropriate in some applications. The next score is Purity (Rendón et al., 2011), which can overestimate the performance since it does not check if any cluster combines objects from different components. We notice that other two indices, the adjusted Rand index (ARI; Hubert & Arabie, 1985) and adjusted variation information (AVI; Vinh et al., 2010), behave similarly to CC, but these indices include an adjustment process to allow for the degree of clustering which might occur by chance, which makes them have a slightly lower score than CC.

We proposed a new method (Lifting) which locates clusters in a dendrogram based on a denoising method, the lifting "one coefficient at a time" (LOCAAT) algorithm, in Section 5.4. The lifting algorithm on trees needs three features: joined pairs, branch lengths and a function value for each node. The first two features can be easily obtained during the process of building the trees. In this study, we built the trees using hierarchical clustering. Some data sets may come with a node value, but one is not always available. Thus, to obtain the third requirement of our algorithm, we proposed a generalized node value for multidimensional data sets called compactness, which was defined as the average distance from the centroid of each possible group in a tree. If the denoised detail coefficients of a node and all its child nodes are less than or equal to a threshold, we declare that node to be a cluster. We concluded a simulation study to compare our algorithm, other CVIs and Mclust in terms of various cluster validity scores using four different artificial data structures in two-dimensional space and a real data set. Two artificial data structures included five normally distributed components, where components had a low variation in one structure and had a larger variation in the other structure which created some overlaps between different components. The other two data structures were more complex; one of them consisted of three concentric circles, and the other one included six irregularly shaped components in total. Finally, we checked the performance of our Lifting algorithm using the well known crabs data set (Campbell & Mahon, 1974).

Simulation results and the results of crabs data set showed that our Lifting method partitioned the data well in terms of the cluster validity scores. When the data sets included normally distributed components, Mclust always showed the highest performance as expected, and some CVIs (the CH, H and KL indices) showed slightly better results than our Lifting method. When the variation increased within the components, there was some overlap between the components. In this case, the performance of other CVIs decreased, and our Lifting method performed better than them. For the concentric circle data set, Mclust failed to partition the data; it tended to combine objects from different components into the same cluster. The CH, H and KL indices performed similarly and captured the two main components into separate clusters while they divided one component into many clusters. Our Lifting method tended to divide two main components into small clusters which decreased its performance. The reason was we were setting a constant threshold which was used for each replicate. However, the variation changed in each replicate of the simulation study, so fixing a threshold at a certain value decreased the performance of our algorithm. Our findings for the final data structure, non-normally distributed six-component data set, showed that the CH index performed slightly better than our Lifting method, and other CVIs either failed to partition or had a high variation between replicates. Overall, Mclust showed a better performance than some of the indices, but it had a weak performance especially for capturing the tails of different components.

In Chapter 6, we presented an updated version of our Lifting method (ALifting). Our ALifting method estimated the threshold for each data set, so we omitted the process of artificial threshold setting. We proposed that if the denoised compactness values of a node and all its child nodes are less than or equal to the universal threshold (weighted variation), we place a cluster at this node. The performance of our ALifting algorithm for the artificial data structures was much higher than other methods. It showed the highest performance after Mclust for normally distributed data structures. Its performance for the six-component uniquely shaped data structure noticeably increased, and it captured the true components with the highest similarity scores. For the three-component concentric circle data structure, its performance did not change much. Thus, our ALifting algorithm underestimated the threshold for this specific data structure; we may need to consider different variance estimation methods.

In Chapter 6, we also proposed a method based on the non-decimated lifting (NLT) algorithm. In NLT, the LOCAAT algorithm is repeated P times using a different permutation order of nodes for lifting. Thus, in each repetition, if we apply our ALifting algorithm, we will have P different clustering results. We can summarize the P clustering results by the proportion of times the node in P repetitions. In this way, we proposed a method which gives a probability of placing a cluster at each node. If the clustering probability of a node and all its child nodes are higher than or equal to a chosen probability of acceptance, $\theta \in [0, 1]$, we place a cluster at this node. Its behaviour was similar to our ALifting algorithm when we set θ at 0.5 with high probabilities. In this version of our algorithm, we need to choose P carefully since high P is the only way to reduce the possible variation on clustering caused by the permutation stage of our NLT algorithm.

Our methods are also applicable to phylogenetic data sets, based on DNA sequences. In Chapter 7, we presented how to apply our algorithms to DNA sequences. Phylogenetic trees are binary trees, so we can apply our proposed algorithms for multidimensional data sets easily to phylogenetic data sets. We proposed that we can use two different distance measures to build trees: matching distances which count the number of loci having different nucleotides between a pair of DNA sequences, and phylogenetic distances using an evolutionary model. After building the tree hierarchically, we can still use compactness as a node value, but we need to apply multidimensional scaling to find the position of each sequence in $\mathbb{R}^p$, where we decide p by checking eigenvalues. We also suggested a second approach which calculates the dissimilarity score for each node in the tree. For all the sequences under a node, we count the number of loci for which any species has a different nucleotide in that position.

We produced three different artificial phylogenetic data sets. First two of them had only mutation history and included 100 species, and the only differences was the seed we used to simulate sequences. We aimed to see how different trees could be built using the same parameters. The third of them had 30 species having migration history in addition to the mutation history. We found that CVIs failed to cluster phylogenetic data sets with only mutation history with matching distance, but if we build the tree with phylogenetic distances, the H index and Gap statistic find a clustering pattern. Our algorithms always clustered the data, and when we checked the dendrogram illustrations, the groups identified looked like a good summary of these data sets. These two artificial data sets were not labelled, so we could not check the performance of different methods. However, the third data set included three sub-populations. We compared the performance of different methods using similarity scores. The CH index and Sil statistic failed to partition the data, and the partitions found by the KL index and Gap statistic when we built the tree were highly variable especially with matching distances. Mclust and the H index always performed well with little or no variation. For this data structure, we found that when we built the tree with phylogenetic distances, our method with zero threshold showed the best performance. However, there was variation between each replicate of the simulation study because of the manual choice of the threshold. Our ALifting and NLT methods always found the same pattern with any distance method and any node value. However, their performances were lower than others. This showed that for small data sets our ALifting and NLT methods tended to overestimate the threshold. Using a tuning parameter on the variance estimate, the performance of our algorithms noticeably increased. Hence, if a more robust variance estimate can be found, the performance of our algorithms can be high for any type of data structure.

Application of different methods including our algorithm to a HIV-1 data set (Salazar-Gonzalez et al., 2009) showed that while other CVIs tended to group different individuals into different clusters, our algorithms and Mclust clustered some individuals together. In this data set, we had the region information. However, we need to question whether we want to find a clustering pattern showing different regions or different individuals, or if there are any signs which show some individuals share the similar viruses. Our algorithm found a different clustering pattern which suggested that some individuals branched together.

In Chapter 7, we also investigated the behaviour of our method on cophylogenetic data sets. We analyzed two well known real data sets: pocket gophers and their chewing lice (Hafner et al., 1994) and deep sea clams and their sulfur-oxidizing bacteria (Peek et al., 1998). The aim was not to identify any speciation events. We would like to provide a visualization tool which may help other researchers to summarize these type of data sets. We applied our algorithm separately to both host and parasite trees, so interactions between these trees were not counted. Both data sets are small data sets, so our ALifting and NLT methods would find a single large cluster. Thus, we applied our algorithm with zero threshold. We found that our algorithm clustered the congruent part of parasites and hosts.

In this study, we only explored the usage of lifting in clustering on binary trees, but it will be also interesting to explore how a lifting-based clustering idea can be applied on a network. Binary trees branch from a root, so we can apply our algorithm starting from the root. If our method does not identify a cluster at the root, we check the next generation and so on. Thus, we have a starting point and a ruled branching pattern. However, in a network, we do not have any starting point to check where we can place a cluster, and we do not know in which order branching occurs.

Appendix A

Extra plots for Chapters 5 and 6

In Section 5.5, four different simulation studies are set. Some extra plots and tables are given with different settings in the following sections. We repeat the simulation study with 1000 replicates for the five-component normally distributed data sets with complete linkage, and we also repeat the simulation study for all four different data settings if we set the upper boundary at 15 for the CH, H and KL indices and Sil statistic. For the Gap statistic, we also set the upper boundary at 15, and ten reference data sets are generated over the range of the data as we applied in Section 5.5. We use four different versions of our proposed clustering algorithm. The one labelled as ZLifting is for our Lifting algorithm with zero threshold ($\lambda = 0$), introduced in Section 5.4. We set a different arbitrary λ for each data structure in Lifting while ALifting picks the threshold automatically (see Section 6.2). The final version of our algorithm is NLT, introduced in Section 6.3. In NLT, the probability of acceptance and the number of paths are set at $\theta = 0.5$ and $P = 100$, respectively. Each figure and table are labelled with the settings we used. Brief discussion for each setting can be found in Section 5.5.

A.1 Five-component normally distributed data with low variance

Index	Mclust	CH	H	KL	Sil	Gap	ZLifting	Lifting	ALifting	NLT
N	5	5	5	25	5	3	59	4	14	16
Purity	0.999	0.995	0.989	0.953	0.993	0.674	0.998	0.808	0.998	0.998
ARI	0.998	0.985	0.980	0.442	0.984	0.588	0.216	0.773	0.623	0.536
AVI	0.993	0.970	0.965	0.556	0.968	0.587	0.579	0.889	0.797	0.761
CompCheck	0.998	0.986	0.988	0.409	0.988	0.993	0.147	0.978	0.510	0.421
ClustCheck	0.998	0.991	0.982	0.951	0.987	0.670	0.999	0.722	0.997	0.998
CC	0.998	0.988	0.985	0.578	0.987	0.770	0.382	0.838	0.712	0.647

Table A.1: The comparison of CVIs for the five-component normally distributed data with low variance (complete linkage, limit=50). First row is for the average number of clusters, and others are for the average similarity scores. For Lifting, the threshold λ is set at 0.2.

Index	Mclust	CH	H	KL	Sil	Gap	ALifting	ALifting2	NLT
N	5	8	8	8	6	3	7	6	6
Purity	0.999	0.961	0.833	0.912	0.875	0.506	0.997	0.807	0.998
ARI	0.998	0.948	0.800	0.883	0.846	0.373	0.989	0.994	0.993
AVI	0.993	0.942	0.827	0.881	0.865	0.519	0.984	0.992	0.989
CompCheck	0.998	0.990	0.990	0.990	0.994	1.000	0.984	0.993	0.991
ClustCheck	0.998	0.945	0.762	0.894	0.830	0.430	0.998	0.998	0.998
CC	0.998	0.964	0.862	0.928	0.899	0.633	0.991	0.995	0.994

Table A.2: The comparison of CVIs for the five-component normally distributed data with low variance (single linkage, limit=15). First row is for the average number of clusters, and others are for the average similarity scores.

A.2 Five-component normally distributed data with larger variation

Index	Mclust	CH	H	KL	Sil	Gap	ZLifting	Lifting	ALifting	NLT
N	5	6	5	7	5	1	54	5	21	23
Purity	0.981	0.929	0.846	0.768	0.888	0.233	0.972	0.778	0.966	0.968
ARI	0.953	0.810	0.752	0.563	0.794	0.038	0.217	0.663	0.451	0.394
AVI	0.993	0.970	0.965	0.556	0.968	0.041	0.561	0.764	0.681	0.661
CompCheck	0.962	0.823	0.879	0.726	0.875	0.993	0.150	0.868	0.346	0.294
ClustCheck	0.962	0.880	0.764	0.723	0.810	0.229	0.969	0.662	0.962	0.966
CC	0.998	0.988	0.985	0.578	0.987	0.465	0.380	0.753	0.576	0.532

Table A.3: The comparison of CVIs for the five-component normally distributed data with larger variation (complete linkage, limit=50). First row is for the average number of clusters, and others are for the average similarity scores. For Lifting, the threshold λ is set at 0.3.

Index	Mclust	CH	H	KL	Sil	Gap	ALifting	ALifting2	NLT
N	5	7	11	12	2	1	21	20	21
Purity	0.981	0.248	0.236	0.252	0.201	0.200	0.925	0.762	0.936
ARI	0.953	0.044	0.027	0.043	0.000	0.000	0.831	0.856	0.842
AVI	0.879	0.059	0.037	0.058	0.000	0.001	0.818	0.853	0.823
CompCheck	0.962	0.983	0.974	0.972	0.998	1.000	0.835	0.873	0.835
ClustCheck	0.962	0.220	0.212	0.219	0.199	0.199	0.902	0.904	0.918
CC	0.962	0.462	0.453	0.459	0.446	0.447	0.866	0.887	0.875

Table A.4: The comparison of CVIs for the five-component normally distributed data with larger variation (single linkage, limit=15). First row is for the average number of clusters, and others are for the average similarity scores.

A.3 Three-component concentric circle data

Index	Mclust	CH	H	KL	Sil	Gap	ALifting	ALifting2	NLT
N	9	7	5	6	2	5	20	20	20
Purity	0.894	1.000	1.000	1.000	0.778	0.999	1.000	0.821	1.000
ARI	0.280	0.926	0.968	0.963	0.532	0.943	0.316	0.316	0.315
AVI	0.366	0.763	0.898	0.876	0.532	0.903	0.568	0.569	0.567
CompCheck	0.293	0.913	0.963	0.957	1.000	0.935	0.280	0.280	0.279
ClustCheck	0.832	1.000	1.000	1.000	0.622	0.999	1.000	1.000	1.000
CC	0.494	0.956	0.981	0.978	0.789	0.966	0.529	0.529	0.528

Table A.5: The comparison of CVIs for the three-component concentric circle data (single linkage, limit=15). First row is for the average number of clusters, and others are for the average similarity scores.

A.4 Six-component non-normally distributed data

Index	Mclust	CH	H	KL	Sil	Gap	ALifting	ALifting2	NLT
N	9	9	7	7	6	2	13	12	13
Purity	0.943	0.913	0.767	0.765	0.705	0.404	0.996	0.868	0.996
ARI	0.873	0.929	0.767	0.730	0.611	0.118	0.949	0.953	0.950
AVI	0.770	0.906	0.763	0.714	0.595	0.161	0.929	0.936	0.931
CompCheck	0.836	0.993	0.994	0.994	0.995	1.000	0.923	0.929	0.925
ClustCheck	0.968	0.907	0.721	0.731	0.657	0.273	0.997	0.997	0.997
CC	0.899	0.947	0.841	0.832	0.773	0.505	0.959	0.963	0.960

Table A.6: The comparison of CVIs for the six-component non-normally distributed data (single linkage, limit=15). First row is for the average number of clusters, and others are for the average similarity scores.

Bibliography

Arbelaitz, O., Gurrutxaga, I., Muguerza, J., PéRez, J. M., & Perona, I. (2013). An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1), pp. 243–256.

Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., & Wheeler, D. L. (2005). GenBank. *Nucleic Acids Research*, 33(Database issue), pp. D34–D38.

Brooks, D. R. (1988). Macroevolutionary comparisons of host and parasite phylogenies. *Annual Review of Ecology and Systematics*, 19(1), pp. 235–259.

Calinski, T. & Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, 3(1), pp. 1–27.

Campbell, N. A. & Mahon, R. J. (1974). A multivariate study of variation in two species of rock crab of genus *Leptograpsus*. *Australian Journal of Zoology*, 22, pp. 417–425.

Chalise, P., Raghavan, R., & Fridley, B. (2016). *IntNMF: Integrative Clustering of Multiple Genomic Dataset*. R package version 1.1.

Charleston, M. A. (1998). Jungles: a new solution to the host/parasite phylogeny reconciliation problem. *Mathematical Biosciences*, 149(2), pp. 191–223.

Charrad, M., Ghazzali, N., Boiteau, V., & Niknafs, A. (2014). NbClust: an R package for determining the relevant number of clusters in a data set. *Journal of Statistical Software*, 61(6), pp. 1–36.

Chen, W.-C. (2011). Overlapping codon model, phylogenetic clustering, and alternative partial expectation conditional maximization algorithm. *Ph.D. Dissertation, Iowa State University*.

Chui, C. K. (1997). *Wavelets: A Mathematical Tool for Signal Processing*. Philadelphia: Society for Industrial and Applied Mathematics.

Claypoole, R., Davis, G., Sweldens, W., & Baraniuk, R. (2003). Nonlinear wavelet transforms for image coding via lifting. *IEEE Transactions on Image Processing*, 12(12), pp. 1449–1459.

Coifman, R. & Donoho, D. (1995). Translation-invariant de-noising. In A. Antoniadis & G. Oppenheim (Eds.), *Wavelets and Statistics: Lecture Notes in Statistics*, volume 103 (pp. 125–150). New York: Springer.

Conow, C., Fielder, D., Ovadia, Y., & Libeskind-Hadas, R. (2010). Jane: a new tool for the cophylogeny reconstruction problem. *Algorithms for Molecular Biology*, 5(1), 16.

Daubechies, I. (1992). *Ten Lectures on Wavelets*. Philadelphia: Society for Industrial and Applied Mathematics.

Donoho, D. L. & Johnstone, I. M. (1994). Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3), pp. 425–455.

Donoho, D. L., Johnstone, I. M., Kerkyacharian, G., & Picard, D. (1995). Wavelet shrinkage: Asymptopia? *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(2), pp. 301–369.

Durbin, R. D. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge: Cambridge University Press.

Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5), pp. 1792–1797.

Efron, B. (1979). Bootstrap methods: another look at the jackknife. *The Annuals of Statistics*, 7(1), pp. 1–26.

Efron, B., Halloran, E., & Holmes, S. (1996). Bootstrap confidence levels for phylogenetic trees. *Proceedings of the National Academy of Sciences of the United States of America*, 93(23), pp. 13429–13434.

Ewens, W. (1979). *Mathematical Population Genetics*, volume 9 of *Biomathematics*. Berlin: Springer.

Fahrenholz, H. (1913). Ectoparasiten und abstammungslehre. *Zoologischer Anzeiger*, 41, pp. 371–374.

Felsenstein, J. (1981). Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6), pp. 368–376.

Felsenstein, J. (1985a). Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 39(4), pp. 783–791.

Felsenstein, J. (1985b). Phylogenies and the comparative method. *The American Naturalist*, 125(1), pp. 1–15.

Feng, D. & Doolittle, R. F. (1987). Progressive sequence alignment as a prerequisitetto correct phylogenetic trees. *Journal of Molecular Evolution*, 25(4), pp. 351–360.

Filipiak, A., Zając, K., Kübler, D., & Kramarz, P. (2016). Coevolution of host-parasite associations and methods for studying their cophylogeny. *Invertebrate Survival Journal*, 13, pp. 56–65.

Fitch, W. M. (1971). Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology*, 20(4), pp. 406–416.

Fowlkes, E. B. & Mallows, C. L. (1983). A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383), pp. 553–569.

Fraley, C. & Raftery, A. E. (2002). Model-based clustering, discriminant analysis and density estimation. *Journal of the American Statistical Association*, 97, pp. 611–631.

Fraley, C., Raftery, A. E., Murphy, T. B., & Scrucca, L. (2012). *mclust Version 4 for R: Normal Mixture Modeling for Model-Based Clustering, Classification, and Density Estimation*.

Gotoh, O. (1996). Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology*, 264(4), pp. 823–838.

Hafner, M. S. & Nadler, S. A. (1988). Phylogenetic trees support the coevolution of parasites and their hosts. *Nature*, 332, pp. 258–259.

Hafner, M. S., Sudman, P., Villablanca, F., Spradling, T., Demastes, J. W., & Nadler, S. A. (1994). Disparate rates of molecular evolution in cospeciating hosts and parasites. *Science*, 265, pp. 1087–1090.

Hartigan, J. A. (1975). *Clustering Algorithms*. New York: J. Wiley & Sons.

Hasegawa, M., Kishino, H., & Yano, T. (1985). Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, 22(2), pp. 160–174.

Hogeweg, P. & Hesper, B. (1984). The alignment of sets of sequences and the construction of phyletic trees: an integrated method. *Journal of Molecular Evolution*, 20(2), pp. 175–186.

Hommola, K., Smith, J. E., Qiu, Y., & Gilks, W. R. (2009). A permutation test of host-parasite cospeciation. *Molecular Biology and Evolution*, 26(7), pp. 1457–1468.

Hubert, L. & Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2(1), pp. 193–218.

Hudson, R. R. (2002). Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18(2), pp. 337–338.

Huelsenbeck, J. P., Rannala, B., & Larget, B. (2000). A Bayesian framework for the analysis of cospeciation. *Evolution*, 54(2), pp. 352–364.

Isaev, A. (2006). *Introduction to Mathematical Methods in Bioinformatics*. Berlin: Springer.

Jansen, M., Nason, G. P., & Silverman, B. W. (2001). Scattered data smoothing by empirical Bayesian shrinkage of second-generation wavelet coefficients. In *International Symposium on Optical Science and Technology*, volume 4478 (pp. 87–97).

Jansen, M., Nason, G. P., & Silverman, B. W. (2009). Multiscale methods for data on graphs and irregular multidimensional situations. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 71(1), pp. 97–125.

Johnstone, I. M. & Silverman, B. W. (2005). Ebayesthresh: R programs for empirical Bayes thresholding. *Journal of Statistical Software*, 12(8), pp. 1–38.

Jukes, T. H. & Cantor, C. R. (1969). Evolution of protein molecules. In H. Munro (Ed.), *Mammalian Protein Metabolism* (pp. 21–132). Academic Press.

Katoh, K., Misawa, K., Kuma, K., & Miyata, T. (2002). MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, 30(14), pp. 3059–3066.

Kimura, M. (1980). A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16(2), pp. 111–120.

Knight, M. & Nunes, M. (2012). *nlt: A Nondecimated Lifting Transform for Signal Denoising*. R package version 2.1-3.

Knight, M. I. & Nason, G. P. (2009). A nondecimated lifting transform. *Statistics and Computing*, 19(1), pp. 1–16.

Krzanowski, W. J. & Lai, Y. T. (1988). A criterion for determining the number of groups in a data set using sum-of-squares clustering. *Biometrics*, 44(1), pp. 23–34.

Legendre, P., Desdevises, Y., & Bazin, E. (2002). A statistical test for hostparasite coevolution. *Systematic Biology*, 51(2), pp. 217–234.

Mallat, S. (1998). *A Wavelet Tour of Signal Processing*. San Diego: Academic Press.

Manly, B. F. J. (2004). *Multivariate Statistical Methods: A Primer*. Boca Raton, FL: Chapman & Hall/CRC Press, third edition.

Mardia, K. V., Kent, J. T., & Bibby, J. M. (1979). *Multivariate Analysis*. London: Academic Press.

Meilă, M. (2007). Comparing clusterings—an information based distance. *Journal of Multivariate Analysis*, 98(5), pp. 873–895.

Merkle, D. & Middendorf, M. (2005). Reconstruction of the cophylogenetic history of related phylogenetic trees with divergence timing information. *Theory in Biosciences*, 123(4), pp. 277–299.

Milligan, G. & Cooper, M. (1985). An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50, pp. 159–179.

Mramba, L., Barber, S., Hommola, K., Dyer, L., Wilson, J., Forister, M., & Gilks, W. R. (2013). Permutation tests for analyzing cospeciation in multiple phylogenies: applications in tri-trophic ecology. *Statistical Applications in Genetics and Molecular Biology*, 12, pp. 1–23.

Nason, G. (2016). *wavethresh: Wavelets Statistics and Transforms*. R package version 4.6.8.

Nason, G. P. (2008). *Wavelet Methods in Statistics with R*. New York: Springer Science+Business Media.

Nason, G. P. & Silverman, B. W. (1994). The discrete wavelet transform in S. *Journal of Computational and Graphical Statistics*, 3(2), pp. 163–191.

Nason, G. P. & Silverman, B. W. (1995). The stationary wavelet transform and some statistical applications. In A. Antoniadis & G. Oppenheim (Eds.), *Wavelets and Statistics: Lecture Notes in Statistics*, volume 103 (pp. 281–300). New York: Springer-Verlag.

Nilsen, G. & Lingjaerde, O. C. (2013). *clusterGenomics: Identifying Clusters in Genomics Data by Recursive Partitioning*. R package version 1.0.

Nooney, C., Barber, S., Gusnanto, A., & Gilks, W. R. (2017). A statistical method for analysing cospeciation in tritrophic ecology using electrical circuit theory. *Statistical Applications in Genetics and Molecular Biology*, 16(5-6), pp. 349–365.

Notredame, C., Higgins, D. G., & Heringa, J. (2000). T-Coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1), pp. 205–217.

Nunes, M. & Knight, M. (2017). *adlift: An Adaptive Lifting Scheme Algorithm*. R package version 1.3-3.

Nunes, M. A., Knight, M. I., & Nason, G. P. (2006). Adaptive lifting for nonparametric regression. *Statistics and Computing*, 16(2), pp. 143–159.

Nunes, M. A. & Nason, G. P. (2004). *Stopping times in adaptive lifting*. Technical Report 05: 15, Department of Mathematics, University of Bristol, UK.

Page, R. D. M. (1990). Component analysis: a valiant failure? *Cladistics*, 6(2), pp. 119–136.

Page, R. D. M. (1993). *Users's Manual for Component, version 2.0*. The Natural History Museum, London, UK.

Page, R. D. M. (1994). Parallel phylogenies: reconstructing the history of host-parasite assemblages. *Cladistics*, 10, pp. 155–173.

Paradis, E., Claude, J., & Strimmer, K. (2004). APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, 20, pp. 289–290.

Peek, A. S., Feldman, R. A., Lutz, R. A., & Vrijenhoek, R. C. (1998). Cospeciation of chemoautotrophic bacteria and deep sea clams. *Proceedings of the National Academy of Sciences of the United States of America*, 95(17), pp. 9962–9966.

Pickands, James, I. (1967). Maxima of stationary Gaussian processes. *Zeitschrift fr Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 7(3), pp. 190–223.

Piella, G. & Heijmans, H. (2002). Adaptive lifting schemes with perfect reconstruction. *IEEE Transactions on Signal Processing*, 50(7), pp. 1620–1630.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rambaut, A. & Grassly, N. C. (1997). Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Computer Applications in the Biosciences*, 13(3), pp. 235–238.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336), pp. 846–850.

Rendón, E., Abundez, I., Arizmendi, A., & Quiroz, E. (2011). Internal versus external cluster validation indexes. *International Journal of Computers and Communications*, 5(1), pp. 27–34.

Ronquist, F. (1995). Reconstructing the history of host-parasite associations using generalised parsimony. *Cladistics*, 11, pp. 73–89.

Rousseeuw, P. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(1), pp. 53–65.

Saitou, N. & Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4), pp. 406–425.

Salazar-Gonzalez, J. F., Salazar, M. G., Keele, B. F., Learn, G. H., Giorgi, E. E., Li, H., Decker, J. M., Wang, S., Baalwa, J., Kraus, M. H., Parrish, N. F., Shaw, K. S., Guffey, M., Bar, K. J., Davis, K. L., Ochsenbauer-Jambor, C., Kappes, J. C., Saag, M. S., Cohen, M. S., Mulenga, J., Derdeyn, C. A., Allen, S., Hunter, E., Markowitz, M., Hraber, P., Perelson, A. S., Bhattacharya, T., Haynes, B. F., Korber, B. T., Hahn, B. H., & Shaw, G. M. (2009). Genetic identity, biological phenotype, and evolutionary pathways of transmitted/founder viruses in acute and early HIV-1 infection. *Journal of Experimental Medicine*, 206(6), pp. 1273–1289.

Schardl, C. L., Craven, K. D., Speakman, S., Stromberg, A., Lindstrom, A., & Yoshida, R. (2008). A novel test for host-symbiont codivergence indicates ancient origin of fungal endophytes in grasses. *Systematic Biology*, 57(3), pp. 483–498.

Schoöniger, M. & Haeseler, A. v. (1995). Simulating efficiently the evolution of DNA sequences. *Bioinformatics*, 11(1), pp. 111–115.

Shimodaira, H. (2002). An approximately unbiased test of phylogenetic tree selection. *Systematic Biology*, 51(3), pp. 492–508.

Shimodaira, H. (2004). Approximately unbiased tests of regions using multistep-multiscale bootstrap resampling. *The Annuals of Statistics*, 32(6), pp. 2616–2641.

Shotwell, M. S. (2013). profdpm: an R package for MAP estimation in a class of conjugate product partition models. *Journal of Statistical Software*, 53(8), pp. 1–18.

Silverman, B. W. (2012). *EbayesThresh: Empirical Bayes Thresholding and Related Methods*. R package version 1.3.2.

Sokal, R. R. & Michener, C. D. (1958). A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28, pp. 1409–1438.

Studier, J. A. & Keppler, K. J. (1988). A note on the neighbor-joining algorithm of Saitou and Nei. *Molecular Biology and Evolution*, 5(6), pp. 729–731.

Sugar, C. A. & James, G. M. (2003). Finding the number of clusters in a dataset: An information-theoretic approach. *Journal of the American Statistical Association*, 98(463), pp. 750–763.

Suzuki, R. & Shimodaira, H. (2006). Pvclust: an R package for assessing the uncertainty in hierarchical clustering. *Bioinformatics*, 22(12), pp. 1540–1542.

Sweldens, W. (1998). The lifting scheme: A construction of second generation wavelets. *SIAM Journal Mathematical Analysis*, 29(2), pp. 511–546.

Taylor, W. R. (1988). A flexible method to align large numbers of biological sequences. *Journal of Molecular Evolution*, 28(1), pp. 161–169.

Thompson, J., Higgins, D., & Gibson, T. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22), pp. 4673–4680.

Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2), pp. 411–423.

Trappe, W. K. & Liu, K. J. R. (2000). Denoising via adaptive lifting schemes. In A. Aldroubi, A. F. Laine, & M. A. Unser (Eds.), *Proceedings of SPIE, Wavelet Applications in Signal and Image Processing VIII*, volume 4119 (pp. 302–312).

Venables, W. N. & Ripley, B. D. (2002). *Modern Applied Statistics with S*. New York: Springer, fourth edition.

Vidakovic, B. (1999). *Statistical Modeling by Wavelets*. New York: J. Wiley.

Vienne, D. M., Refrégier, G., López-Villavicencio, M., Tellier, A., Hood, M. E., & Giraud, T. (2013). Cospeciation vs host-shift speciation: methods for testing, evidence from natural associations and relation to coevolution. *New Phytologist*, 198(2), pp. 347–385.

Vinh, N. X., Epps, J., & Bailey, J. (2009). Information theoretic measures for clusterings comparison: Is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09 (pp. 1073–1080). New York, NY, USA: ACM.

Vinh, N. X., Epps, J., & Bailey, J. (2010). Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11, pp. 2837–2854.

Wallace, D. L. (1983). A method for comparing two hierarchical clusterings: Comment. *Journal of the American Statistical Association*, 78(383), pp. 569–576.

Wright, S. (1943). Isolation by distance. *Genetics*, 28(2), pp. 114–138.

Yang, Z. (1993). Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. *Molecular Biology and Evolution*, 10(6), pp. 1396–1401.