Dendrogram seriation in data visualisation: algorithms and applications

Denise Earle

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

National University of Ireland Maynooth
Department of Mathematics
October 2010

Head of Department: Professor Stephen Buckley
Thesis Supervisor: Doctor Catherine Hurley

Contents

Abstract

Seriation is a data analytic tool for obtaining a permutation of a set of objects with the goal of revealing structural information within the set of objects. The purpose of this thesis is to investigate and develop tools for seriation with the goal of using these tools to enhance data visualisation.

The particular focus of this thesis is on "dendrogram seriation" algorithms. A dendrogram is a tree-like structure used for visualising the results of a hierarchical clustering and the order of the leaves in a dendrogram provides a permutation of a set of objects. Dendrogram seriation algorithms rearrange the leaves of a dendrogram in order to find a permutation that optimises a given criterion.

Dendrogram seriation algorithms are widely used, however, the research in this area is often confusing because of inconsistent or inadequate terminology. This thesis proposes new notation and terminology with the goal of better understanding and comparing dendrogram seriation algorithms.

Seriation criteria measure the goodness of a permutation of a set of objects. Popular seriation criteria include the path length of a permutation and measuring anti-Robinson form in a symmetric matrix. This thesis proposes two new seriation criteria, "lazy path length" and "banded anti-Robinson" form, and demonstrates their effectiveness in improving a variety of visualisations.

The main contribution of this thesis is a new dendrogram seriation algorithm. This algorithm improves on other dendrogram seriation algorithms and is also flexible because it allows the user to either choose from a variety of seriation criteria, including the new criteria mentioned above, or to input their own criteria.

Finally, this thesis performs a comparison of several seriation algorithms, the results of which show that the proposed algorithm performs competitively against other algorithms. This leads to a set of general guidelines for choosing the most appropriate seriation algorithm for different seriation interests and visualisation settings.

Acknowledgements

I wish to thank my supervisor, Dr. Catherine Hurley, for introducing me to the area of visualisation and seriation, and for the tremendous help and guidance over the past few years.

I wish to acknowledge the financial support provided by a Research Frontiers grant from Science Foundation Ireland.

I also wish to give my thanks to the staff and research students in the Department of Mathematics at NUI Maynooth. Thank you to those who attended my various seminars and gave great advice and encouragement over many cups of tea and coffee.

Thank you to my parents, Pat and Larry, for their constant support and bewilderment at my academic accomplishments. Thank you also for providing an escape house, where I could run to and forget all about work. To my brothers, grandparents, extended family and my husband's family, thank you for supporting me and putting up with me. To my five "sisters", a special thank you for keeping me involved in your lives and for keeping me sane.

Finally, the most important person I wish to thank is Cathal. I am certain that I would not have accomplished all that I have today were it not for you completely spoiling me with love and support. Thank you with all my heart.

List of Figures

List of Tables

Chapter 1

Introduction

Every day, data analysts are faced with the task of understanding, extracting and presenting useful information from data. Crucial to this task are data visualisation tools because they reveal patterns, trends and other features that are simply not shown in tables of data.

Given the importance of data visualisation, it is surprising how often one sees poorly constructed graphics. Graphics often suffer from misleading or inconsistent scales on axes, missing text, unnecessary "jazzing-up" of graphics using 3-D or shading effects, inappropriate use of colour or poorly ordered data. To combat these and other graphical errors many books and papers provide general guidelines for constructing graphics (see, for example, Cleveland 1985, 1993, Chambers et al. 1983, Tufte 2001, Brewer 1994 and Zeileis et al. 2008).

Reordering data is another technique for improving data visualisation. Graphics are generally constructed using the default ordering of variables, cases or categories corresponding to the order in which they are listed in the data. However, graphics are greatly improved when the data is systematically reordered. For example, Cleveland (1993) ordered categories by their median in multi-panel dot plots, Friendly (1994) ordered categories in mosaic displays by their score on the first correspondence analysis direction and Hurley (2004) ordered variables in scatterplot matrices so that interesting panels were positioned close to the main diagonal. In each of these situations, reordering revealed patterns in data that were not obvious when the default order of variables or categories was used.

Seriation is a term used for describing the systematic reordering of data and is defined as follows:

Definition 1.1. *Seriation is a data analytic tool for obtaining a permutation of a set of objects with the goal of revealing structural information within the set of objects.*

There are other terms and techniques related to seriation. For example, in archaeology, relative dating is a method for determining the chronological order of artifacts or past events. In ecology, ordination is the collective term for multivariate techniques that arrange objects along axes.

This chapter continues with a simple example of seriation and then describes the main issues and history of seriation. This chapter also outlines the specific areas of seriation to which this thesis contributes and gives the organisation of the rest of the thesis.

1.1 Seriation example

Der and Everitt (2001, pg. 307) described a dataset, where a large number of people in the UK were surveyed and asked which of thirteen characteristics they would associate with seven European countries. For demonstration purposes, this example uses only nine of the characteristics. Table 1.1 contains the data, where the ij^{th} entry is the percentage of people who think that characteristic i matches country j. One topic of interest for this dataset is finding out which countries are considered similar or dissimilar by the surveyed people.

Table 1.1: This table shows a subset of the data described in Der and Everitt (2001, pg. 307). The ij^{th} entry is the percentage of surveyed people who think characteristic i matches country j.

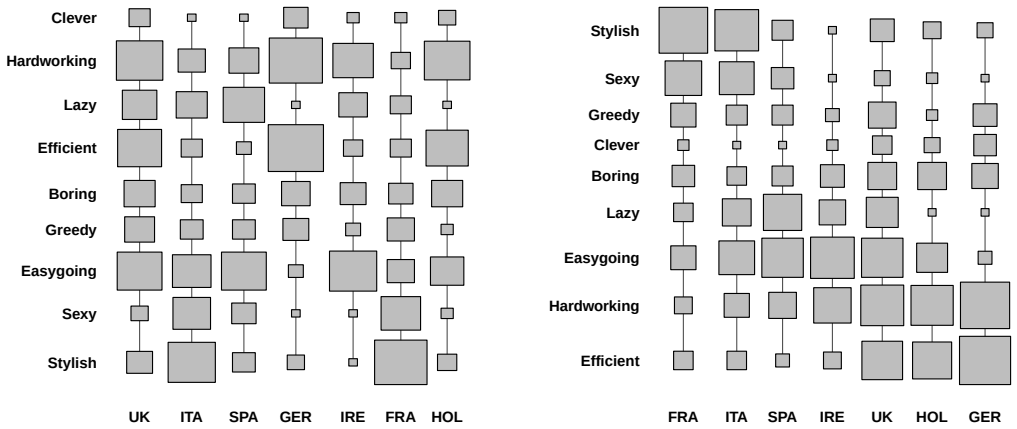
	UK	Italy	Spain	Germany	Ireland	France	Holland
Clever	6	1	1	8	2	2	4
Hardworking	29	10	12	38	22	5	28
Lazy	16	13	23	1	11	6	1
Efficient	26	6	3	41	5	6	24
Boring	13	6	7	11	9	8	13
Greedy	12	7	7	9	3	10	2
Easygoing	27	20	27	3	30	10	15
Sexy	4	19	8	1	1	21	2
Stylish	9	30	7	4	1	37	5

People are more adept at perceiving patterns using shapes and sizes than they are using actual numbers. Therefore, tables or matrices are often better represented using a so called table plot (also known as a fluctuation plot, see, for example, Unwin et al. 2006, §5.4), where each entry in the table is visualised by a box with area proportional to the value of the entry.

Figure 1.1.(a) shows a table plot of the data, where the rows and columns are ordered as they appear in Table 1.1. The vertical lines through the boxes

are drawn in order to assist the reader in comparing the countries.

Even for such a small table plot, it is quite difficult to see relationships between the objects, or identify groups and trends in the data. Performing these tasks is generally much simpler after seriating the rows and columns in the data.



(a) Arbitrary ordering (b) Seriated rows and columns

Figure 1.1: The table plot in (a) visualises the data in Table 1.1, where the rows and columns are ordered as in Table 1.1. The table plot in (b) visualises the same data, except the rows and columns are seriated.

Figure 1.1.(b) shows a table plot of the same data, except the rows and columns are ordered according to the "best" permutation of the countries and the "best" permutation of the characteristics. For this example, the "best" permutation is the one that optimises an objective function (namely the BAR objective function of Section 3.4.2) that rewards permutations placing similar objects close together, where similarity is measured using Euclidean distance.

Seriating the rows and the columns positions similarly sized boxes (i.e. values) close together in the table plot in Figure 1.1.(b), which makes it easier to see groups in the data and to see which countries are perceived as similar by the surveyed people. Using Figure 1.1.(b), it is easy to see that France and Italy scored similarly on all characteristics and were considered the two most stylish and sexy countries. On the other hand, Germany, Holland and the UK were not considered sexy or stylish but were considered more hardworking and efficient. This information is not as easily obtained from the table plot in Figure 1.1.(a) or the raw percentages in Table 1.1.

This example treated the task of reordering the characteristics and countries in Table 1.1 as two separate seriation problems. An alternative seriation problem involves *simultaneously* seriating both the characteristics and countries. However, seriation problems of this type are not considered in this thesis.

1.2 Challenges of seriation

Consider again the problem of seriating the characteristics in Table 1.1 (ignoring the problem of seriating the countries). This appears to be a small seriation problem because there are only nine characteristics. However, for n objects there are $n!$ possible permutations, which means there are in fact 362,880 permutations of the characteristics in Table 1.1. So, what appeared to be a small seriation problem is actually quite big!

Some of the 362,880 permutations of characteristics will improve the table plot in Figure 1.1.(a), while other permutations will not. Therefore, one challenge in seriation is evaluating the "goodness" of a seriation. Robinson (1951) presented the first formal method for evaluating a seriation and since 1951 many other methods or functions have been developed (see, for example, Hahsler et al. 2008). These functions are collectively known as "seriation criteria" and are described in more detail in Section 2.3.

Although time consuming, it is possible to examine all 362,880 permutations of the characteristics in Table 1.1 and determine the best permutation for a given seriation criterion. However, for much larger seriation problems, an exhaustive search of all possible permutations is infeasible. Therefore, the second challenge in seriation is obtaining a good permutation as efficiently as possible.

Many techniques are available for finding a good permutation of a set of objects. Generally, these algorithms are heuristics and produce good but not necessarily optimal seriations. Chapter 2 gives an overview of different seriation methods and algorithms.

1.3 History of seriation

Kendall (1971) credited Sir W.M. Flinders Petrie (1899), an English Egyptologist, as being the first to use formal seriation methods. Hundreds of graves were excavated in the Nile area and Petrie ordered these graves in chronological order based on the objects he found in the graves. Unfortunately, according to Kendall (1971), Petrie's writings are not easy to follow and most of his notes and records are now destroyed. As a result, Petrie's seriation method is not fully understood.

Unlike Petrie, Robinson (1951) provided a mathematical framework for the problem of seriation. He proposed a desired form for a symmetric matrix, whose elements are agreement indexes for a set of archaeological deposits. This suggested form, now called Robinson form, has become an important concept

in seriation.

Robinson (1951) was also the first to introduce a formal method for assessing the goodness of seriation results. He used an agreement coefficient between rows of a matrix to quantify the "goodness" of a seriation result. Prior to Robinson's agreement coefficient, researchers assessed seriation results subjectively using intuitive judgement. Section 2.3.2 gives an account of Robinson form and various functions for measuring it.

The Semiology of Graphics (Bertin 1983) is another highly influential piece of work. One of the many ideas Bertin described in this monograph is the "reorderable matrix", where he showed that reordering the rows and columns of a data matrix makes information easier to understand. Since then, reordering the rows and columns of a matrix is a topic that has received much attention in the literature (see, for example, Gale et al. 1984, Eisen et al. 1998 and Wu et al. 2010).

Petrie (1899), Robinson (1951) and Bertin (1983) built the foundations for many of today's developments in seriation and visualisation. Since 1967 many papers have contributed significantly to the combined areas of seriation and visualisation including Eisen et al. (1998), Chen (2002), Friendly and Kwan (2003) and Hurley (2004).

Software advancements have also influenced the development of seriation and visualisation tools, particularly the software program `R` (R Development Core Team 2010). `R` is a powerful software environment that provides the analyst with free, easily accessible visualisation and seriation software. `R` also provides the researcher a platform on which to share their seriation and visualisation tools with a worldwide community.

1.4 Contributions of the thesis

The main motivation for the research in this thesis is the development of flexible seriation tools to help in extracting information from data. More specifically, this thesis contributes to the following areas:

Understanding dendrogram seriation.

Dendrogram seriation is a popular seriation method that is based on hierarchical clustering and several such algorithms have been developed (see, for example, Gruvaeus and Wainer 1972). However, due to inconsistent terminology or lack of information, it is often difficult to understand exactly how a dendrogram seriation algorithm works and how it differs from other algorithms. This thesis focusses on dendrogram seriation and

develops notation and terminology for describing dendrogram seriation algorithms.

A new dendrogram seriation algorithm.

The new algorithm allows the user to choose from many different seriation criteria. Existing seriation algorithms focus on one seriation criterion only and so the proposed algorithm is more flexible. This algorithm also contains improvements on currently available dendrogram seriation algorithms.

New seriation criteria.

This thesis proposes new criteria for evaluating the goodness of a seriation and describes applications of these criteria to a variety of visualisation settings.

A comparison of seriation algorithms.

This thesis compares the performance of the proposed algorithm with several other seriation algorithms, and provides guidelines for choosing suitable seriation algorithms for different seriation interests and visualisation settings.

This thesis concerns seriation based on a symmetric dissimilarity matrix $D = [d_{i,j}]$, where $d_{i,j}$ represents the dissimilarity between objects i and j, and $d_{i,i} = 0$, for $1 \leqslant i, j \leqslant n$. Carroll and Arabie (1980) refer to such a matrix D as "two-way" because D has two dimensions, and "one-mode" because the rows and columns of D refer to the same set of objects. Seriation is also possible for "two-mode" dissimilarity data (see, for example, the data in Table 1.1), where the rows and columns refer to two different sets of objects (see, for example, Hubert et al. 2006).

A dissimilarity measure, for example correlation or Euclidean distance, measures the dissimilarity of two objects. However, it is also possible to use other measures for seriation which are not dissimilarity measures, for example visualisation based measures. Wilkinson et al. (2005) described several "scagnostic" indexes, which measure the "interestingness" of a scatterplot produced by two variables. Peng et al. (2004) also designed visualisation based measures, which measure "clutter" in components of various statistical graphics including the panels in parallel coordinates plots and scatterplot matrices.

Note that, for convenience, this thesis uses the term "dissimilarity' to refer to all measures, regardless of whether or not the measure is in fact a dissimilarity measure.

1.5 Organisation of the thesis

This chapter has set the scene for the importance of seriation in data visualisation and gave a short account of the beginnings of seriation. It also discussed the main issues with seriation and outlined the specific areas to which this thesis contributes. The rest of the thesis is organised as follows.

Chapter 2 provides an overview of dendrogram seriation, which includes an account of the first dendrogram seriation algorithm (Gruvaeus and Wainer 1972) and a discussion of various reasons for the increasing popularity of dendrogram seriation. The chapter then focusses on different seriation criteria and goes on to survey many other seriation methods and algorithms.

Chapter 3 introduces a new dendrogram seriation algorithm called DendSer. DendSer is more flexible than other seriation algorithms because it allows the user to choose from a variety of seriation criteria including two new criteria, which are described in Section 3.4. Chapter 3 also develops notation and terminology for describing an important parameter of dendrogram seriation that is generally not discussed in the literature. This parameter tells DendSer which permutations to examine and Section 3.6 shows that the most suitable setting for this parameter depends on the choice of seriation criterion.

Chapter 4 presents many visualisation applications of DendSer. The variety of examples highlights not only the flexibility of DendSer but also the application of the new seriation criteria.

Chapter 5 compares the performance of DendSer with several other seriation algorithms. The algorithms are assessed based on how well they recover hidden patterns in data and how well they optimise various seriation criteria. This comparison study results in a set of guidelines for helping the user choose the most suitable seriation algorithm for their specific seriation applications.

Finally, Chapter 6 summarises and discusses the main contributions of the thesis.

Chapter 2

Dendrogram seriation and other seriation methods

2.1 Introduction

This chapter provides an overview of seriation, with particular emphasis on dendrogram seriation.

Section 2.2 introduces dendrogram seriation and gives an account of the first dendrogram seriation algorithm (Gruvaeus and Wainer 1972). This section also discusses some reasons for the increasing popularity of dendrogram seriation.

Typically, seriation algorithms try to find a permutation of a set of objects that optimises a specific goodness of seriation criterion. Two popular seriation criteria are minimising the path length of a permutation and optimising a function that measures how well a symmetric matrix follows anti-Robinson form (Robinson 1951). Section 2.3 gives an account of path length and anti-Robinson form, and also mentions other available seriation criteria.

The example in Section 2.4 uses heatmaps and different seriation criteria to demonstrate the effectiveness of the dendrogram seriation algorithm described in Gruvaeus and Wainer (1972). This example also uses modified versions of icicle plots (Kruskal and Landwehr 1983) to visualise how the algorithm works.

Researchers have used various methods to tackle the problem of seriation. Section 2.5 gives a brief account of many of these methods including Travelling Salesperson heuristics and dimension reduction techniques. Figure 2.5 shows a two-way categorisation of seriation algorithms, the first according to method and the second according to the criterion that the algorithm tries to optimise.

The chapter concludes with a brief summary.

2.2 Dendrogram seriation algorithms

This section describes one branch of seriation algorithms called "dendrogram seriation" algorithms.

Cluster analysis is a general term for a range of statistical techniques that aim to classify objects into groups based on their similarity, where objects in the same group are more similar than objects from different groups. Many different clustering methods are available including hierarchical clustering, k-means and model-based clustering (Fraley and Raftery 1999, 2002). This section focusses on hierarchical clustering, specifically agglomerative hierarchical clustering.

Agglomerative hierarchical clustering begins with all objects in individual clusters and then iteratively merges the two most similar clusters until all objects are in the same cluster. The linkage used in the hierarchical clustering defines the similarity of two clusters. Single linkage defines the similarity of two clusters to be the minimum distance between elements of each cluster, whereas complete linkage defines the similarity of clusters to be the maximum distance between elements of each cluster. Average linkage falls in between these two extremes and defines the similarity of two clusters to be the mean distance between elements of each cluster.

A simple graphic for visualising the results of a hierarchical clustering is a binary tree-like structure called a dendrogram shown in Figure 2.1. The

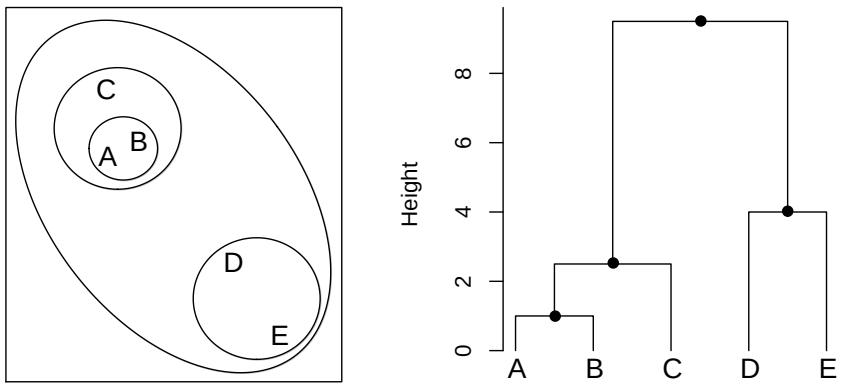


Figure 2.1: Example of a dendrogram visualising a hierarchical clustering of five random data objects.

root of a dendrogram represents a single cluster containing all objects and the leaves (at the bottom of the dendrogram in Figure 2.1 represent the individual objects. A dendrogram visualising a hierarchical clustering of n objects has $n-$

1 nodes (represented by black dots in Figure 2.1), which represent the clusters merged at each step of the hierarchical clustering process. Dendrograms are usually drawn vertically with the root node at the top. The vertical axis represents the level of similarity, or height at which clusters are merged.

2.2.1 Dendrograms and seriation

The order of the leaves in a dendrogram provides a permutation of a set of objects. However, the leaf ordering is not unique. A dendrogram with n leaves has $n-1$ nodes and each of these nodes can be rearranged resulting in a total of 2^{n-1} possible permutations of the objects. "Dendrogram seriation" algorithms rearrange the nodes in a dendrogram in order to obtain a permutation of the leaves (i.e. objects) that optimises some seriation criterion.

Dendrogram seriation dates back to Gruvaeus and Wainer (1972), whose motivation was to obtain a unique ordering of the objects from a hierarchical clustering. Their algorithm examines each node N in a dendrogram and rearranges the left and right sub-nodes of N so that the two most similar objects at the edges of the sub-nodes are placed adjacently. Figure 2.2 illustrates the four comparisons made by the algorithm from Gruvaeus and Wainer (1972), where nodes marked with a circle indicate that the leaves in these nodes are reversed. For example, if $d_{BD} = \min(d_{BC}, d_{BD}, d_{AC}, d_{AD})$, where d is some dissimilarity measure, then the algorithm selects the second dendrogram.

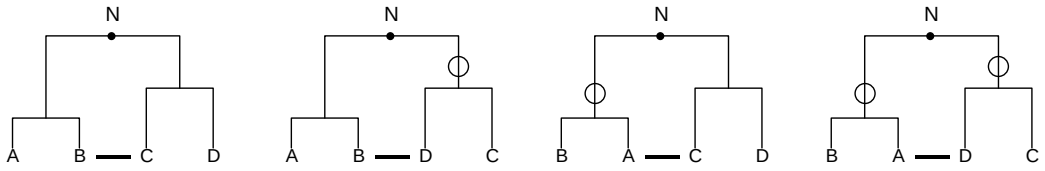


Figure 2.2: Illustration of the dendrogram seriation algorithm described in Gruvaeus and Wainer (1972)

Dendrogram seriation has become increasingly popular since 1972 with several papers describing new algorithms and applications (Degerman 1982, Gale et al. 1984, Eisen et al. 1998, Alon et al. 1999, Wishart 1999, Bar-Joseph et al. 2001, Morris et al. 2003, Forina et al. 2007, Tien et al. 2008 and Wu et al. 2010). These algorithms use various node operations for rearranging the nodes in a dendrogram: some exchange the branches of a node and others reverse the leaves in a node. However, the descriptions of these node operations are often quite vague or even omitted.

2.2.2 Advantages of dendrogram seriation

There are several reasons for the increasing popularity of dendrogram seriation. First of all, dendrogram seriation is based on hierarchical clustering, which is generally well-known and well-understood. Dendrogram seriation also simplifies the problem of reordering n objects by reducing the size of the search space from $n!$ possible permutations to 2^{n-1} possible permutations.

Dendrogram seriation is also very flexible compared to other seriation methods. Many algorithms have been developed, which optimise different seriation criteria such as the path length criterion (Bar-Joseph et al. 2001, Forina et al. 2007) and anti-Robinson criteria (Wishart 1999, Morris et al. 2003). Dendrogram seriation has also been successfully applied to many visualisation settings including scatterplot matrices, parallel coordinates plots (Hurley 2004) and heatmaps (see, for example, Gale et al. 1984, Eisen et al. 1999).

Also, if the user is interested in clustering their data as well as seriating their data, then dendrogram seriation algorithms are a convenient tool. However, Chapter 5 shows that dendrogram seriation algorithms produce results that are competitive with other seriation algorithms that are not based on hierarchical clustering. Therefore, dendrogram seriation algorithms are very useful, regardless of the user's clustering interests.

2.3 Seriation criteria

Two seriation criteria that frequently feature in the literature are path length and anti-Robinson form. This section provides a formal definition of these two seriation criteria and also gives a brief account of their visualisation applications.

2.3.1 Path length

The shortest path problem is a variation of the well-known Travelling Salesperson Problem (TSP). Given n cities, the goal of the TSP is to find the shortest tour that starts from a selected city, visits each city once and returns to the starting city. With the shortest path problem, there is no return to the starting city.

In relation to seriation, the following cost function measures the path length of a permutation:

Definition 2.1. *Take a set of n objects, where $d_{i,j}$ is the dissimilarity value between objects i and j. For a permutation π of the objects, the path length*

cost function is defined as:

$$PL(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)}, \tag{2.1}$$

where $\pi(i)$ is the object in the ith position of π.

A permutation minimising the PL cost function aims to place similar objects adjacently, which makes minimising the PL cost function a suitable goal for seriation.

The path length criterion has been successfully applied to a variety of statistical visualisations. Hurley (2004) described why the path length criterion is suitable when seriating variables in a parallel coordinates plot. Bar-Joseph et al. (2001) showed that minimising the path length of a permutation helps to reveal biological structure in heatmaps of gene expression data. Hahsler and Hornik (2007) used the path length criterion to seriate a heatmap of a dissimilarity matrix.

2.3.2 Robinson form

Consider a symmetric matrix where the values in the matrix are non-increasing as one moves away from the diagonal. A matrix with this pattern is said to have "Robinson" form after the statistician W.S. Robinson, who first described this pattern in Robinson (1951). Similarly, a matrix where the values are non-decreasing as one moves away from the diagonal is said to have "anti-Robinson" form.

Definition 2.2. *A symmetric matrix $D = [d_{i,j}]$, for $1 \leqslant i, j \leqslant n$, has anti-Robinson form if $d_{i,k} \leqslant d_{i,j}$ and $d_{k,j} \leqslant d_{i,j}$, for $i < k < j$.*

The following is an example of a symmetric matrix that follows anti-Robinson form:

$$\text{AR matrix} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 1 & 2 & 3 \\ 2 & 1 & 0 & 1 & 2 \\ 3 & 2 & 1 & 0 & 1 \\ 4 & 3 & 2 & 1 & 0 \end{pmatrix} \tag{2.2}$$

Anti-Robinson form is a natural concept for seriation. If a dissimilarity matrix has anti-Robinson form, then the smallest dissimilarity values are close

to the main diagonal and the largest values are far away from the main diagonal. This means that objects with low dissimilarity are close together in the corresponding permutation and objects with high dissimilarity are far apart.

Anti-Robinson form has several applications in seriating statistical graphics. Gale et al. (1984) and Wishart (1999) aimed to seriate dissimilarity matrices so that they were as close as possible to anti-Robinson form, in order to produce more visually appealing heatmaps. Morris et al. (2003) applied anti-Robinson form to improving visualisations of networks. Hurley (2004) described how anti-Robinson form is a desirable pattern for a scatterplot matrix.

Several algorithms exist that uncover anti-Robinson form if it is present in a matrix (see, for example, Hubert 1974). However, in most cases, anti-Robinson form is not present in symmetric matrices and so it is generally only possible to reorder a matrix so that it follows approximate anti-Robinson form. Therefore, it is useful to be able to measure how close a matrix is to following anti-Robinson form.

Hubert et al. (2001, pg. 55) defined two merit functions which measure how well a matrix follows anti-Robinson form. The first function computes the number of pairs of values that satisfy the conditions in Definition 2.2 minus the number of pairs that violate those conditions (such a pair is referred to, in this thesis, as an "anti-Robinson violation"). The second function is a weighted version of the first. Chen (2002) also described a loss function for measuring anti-Robinson form, which counts only violations of anti-Robinson form. He also described a weighted version of this measure. These four anti-Robinson functions are implemented in the `R` package `seriation` (Hahsler et al. 2010).

Hubert et al. (2006) described another anti-Robinson merit function, which sums the values of the element-wise product of a dissimilarity matrix and a target matrix. The target matrix is a dissimilarity matrix between unit spaced objects on a straight line and so follows anti-Robinson form. For example, Equation 2.2 gives such a target matrix for $n = 5$. Formally, for a permutation π of n objects, the merit function from Hubert et al. (2006) is defined as follows:

$$\text{ARc}(\pi) = \sum_{|i-j| \leqslant n-1} |i-j| d_{\pi(i),\pi(j)}. \tag{2.3}$$

Maximising the function in Equation 2.3 is equivalent to maximising the Pearson correlation between the dissimilarities $d_{i,j}$ and their corresponding $|i-j|$ values. Therefore, this thesis refers to the anti-Robinson function in Equation 2.3 as ARc, where c denotes correlation.

There are many functions related to Equation 2.3, two of which Caraux and Pinloche (2005) call the "least squares" and "inertia" functions. The least

squares loss function is defined as follows:

$$\mathrm{LS}(\pi) = \sum_{|i-j| \leqslant n-1} (|i-j| - d_{\pi(i),\pi(j)})^2. \tag{2.4}$$

Expanding Equation 2.4 gives the following:

$$\mathrm{LS}(\pi) = \sum_{|i-j| \leqslant n-1} |i-j|^2 - 2 \sum_{|i-j| \leqslant n-1} |i-j| d_{\pi(i),\pi(j)} + \sum_{|i-j| \leqslant n-1} (d_{\pi(i),\pi(j)})^2. \tag{2.5}$$

In the expansion of $LS(\pi)$ in Equation 2.5, disregarding the constant first and third terms and also the scalar 2 in the middle term, it is clear that minimising Equation 2.4 is equivalent to maximising Equation 2.3.

The inertia merit function places emphasis on pushing large dissimilarities away from the main diagonal, as opposed to pulling small dissimilarities close to the main diagonal:

$$\mathrm{IN}(\pi) = \sum_{|i-j| \leqslant n-1} d_{\pi(i),\pi(j)} |i-j|^2. \tag{2.6}$$

Equation 2.6 differs from Equation 2.3 by multiplying the dissimilarities by $|i-j|^2$ instead of $|i-j|$.

2.3.3 Other seriation criteria

Path length, anti-Robinson form, least squares and inertia are examples of seriation criteria that are based on dissimilarities between objects (as are the two new seriation criteria described in Sections 3.4.1 and 3.4.2). The following criteria may be based on either a dissimilarity matrix or the data matrix itself.

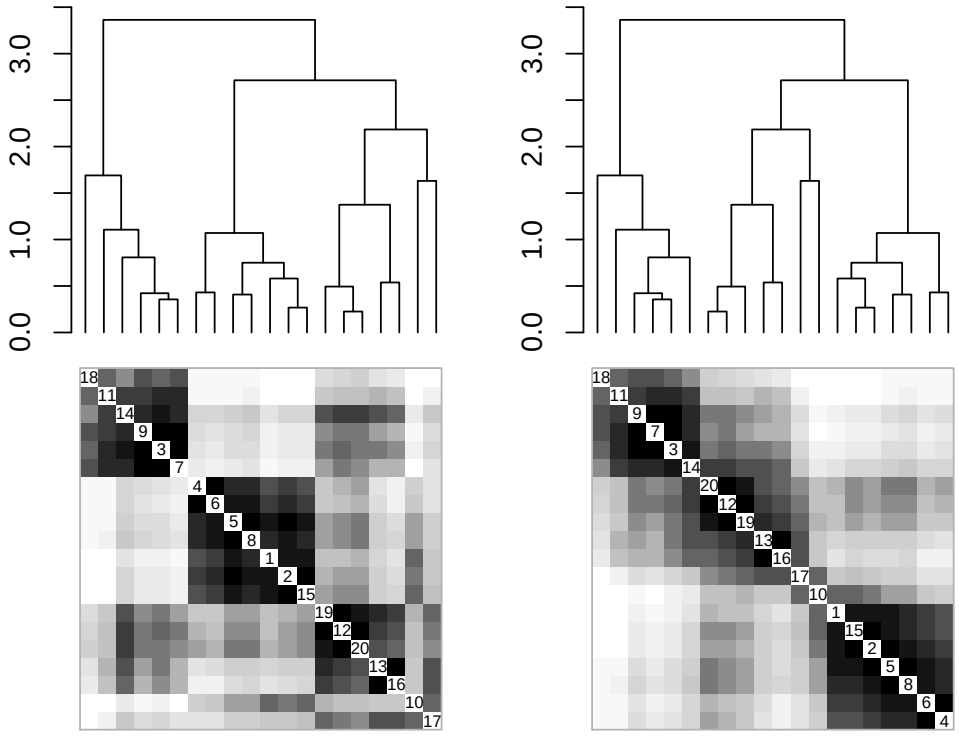
Niermann (2005) defined two stress measures called Moore stress and Neumann stress, which compare the entries in a matrix with their neighbours. The smaller the stress value of a matrix, the more similar the entries are to their neighbours. McCormick et al. (1972) defined a similar measure called the "measure of effectiveness", which is described in Section 2.5.4. These three criteria are implemented in the `R` package `seriation` (Hahsler et al. 2010).

2.4 Dendrogram seriation example

This example demonstrates the use of the dendrogram seriation algorithm described in Gruvaeus and Wainer (1972) (referred to here as the "GW" algorithm).

The well-known Iris dataset (Fisher 1936) contains four measurements of 150 irises. For demonstration purposes, this example uses a random sample of twenty irises. After standardising the variables, the irises are hierarchically clustered using Euclidean distance and average linkage. (Note that throughout this thesis standardising is done in the conventional way, i.e. standardising to zero mean and unit standard deviation.)

Figure 2.3 shows two dendrograms visualising the results of the hierarchical clustering and two heatmaps of the Euclidean distance matrix, where the colour scale black to white represents low to high Euclidean distance. The permutation of the irises in the dendrogram in Figure 2.3.(a) is used to order the rows/columns of the corresponding heatmap. The dendrogram in Figure 2.3.(b) is rearranged by the GW algorithm and the new permutation of irises is used to order the rows/columns in the corresponding heatmap.



(a) Initial leaf order (b) GW leaf order

Figure 2.3: Dendrograms and heatmaps of the Euclidean distance matrix for a sample of irises from the Iris dataset. The heatmap in (a) is constructed using the permutation of irises from the initial dendrogram. The heatmap in (b) is constructed using the permutation of irises returned from the algorithm described in Gruvaeus and Wainer (1972). The colour scale black to white represents low to high Euclidean distance.

The heatmap in Figure 2.3.(a) shows three dark blocks around the main diagonal, which indicate three clusters of similar irises. The first and third clus-

ters of irises (blocks in the north-west and south-east corners of the heatmap in Figure 2.3.(a)) are also similar to each other, which is indicated by the dark patch at their intersection in the north-east corner of the heatmap. The iris labelled as 10 (second last on the main diagonal in Figure 2.3.(a)) appears to be misplaced because this iris is not similar to the other irises in the third block but is similar to some of the irises in the middle block.

The GW algorithm rearranges the initial dendrogram so that similar irises are placed closer together. This results in the two clusters of similar irises that are separated in Figure 2.3.(a) being placed adjacently in the heatmap in Figure 2.3.(b). These two clusters roughly correspond to two similar species of iris and the third cluster, placed in the south-east corner of the heatmap in Figure 2.3.(b) roughly corresponds to a third different species of iris.

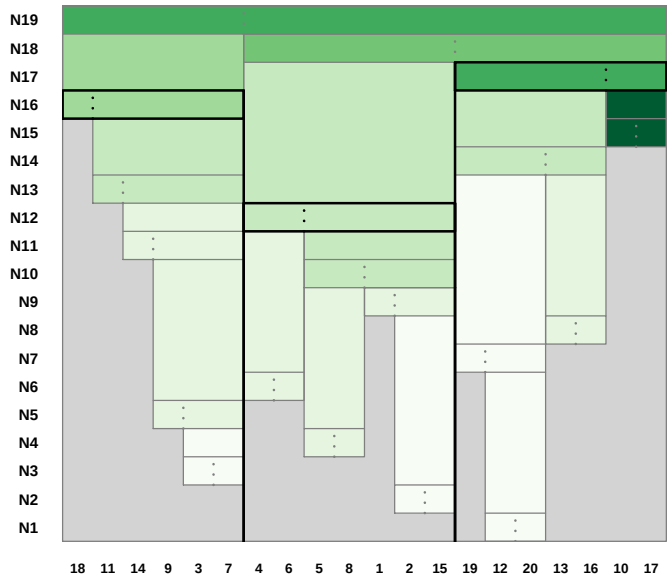
The GW algorithm produces a more visually appealing heatmap. It also reduces the path length of the initial permutation by approximately 28% and almost halves the number of anti-Robinson violations in the Euclidean distance matrix visualised by the heatmap in Figure 2.3.(a).

Icicle plots (Kruskal and Landwehr 1983) allow an alternative examination of the effect of the GW algorithm. Figure 2.4 contains two icicle plots visualising the hierarchical clustering of the sample of twenty irises from the Iris dataset. The columns in the icicle plots represent the objects with the object labels written at the bottom of the columns. The ith row in the icicle plots indicates which pair of objects or clusters are merged at the ith step of the hierarchical clustering process (i.e. the ith row corresponds to the ith node in the dendrogram). The node labels are written on the left hand side of the icicle plots. The merge point of the clusters is represented by a vertical dashed line. The entire hierarchical clustering process is observed by reading the icicle plot from the bottom row to the top.

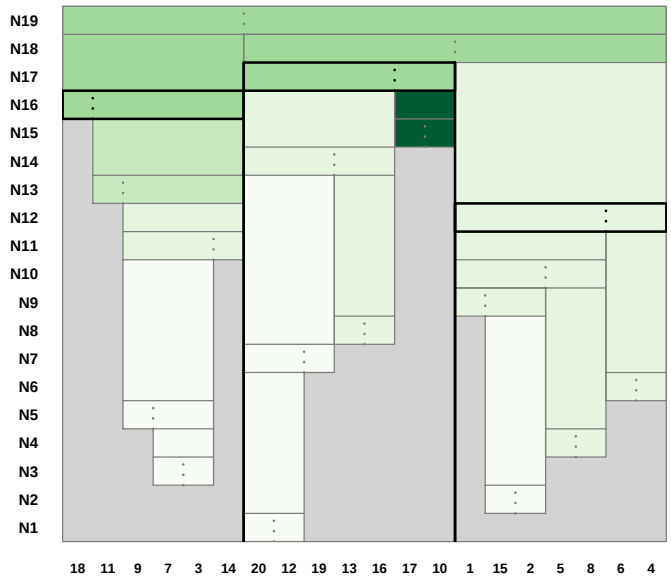
The objects (i.e. columns) in the icicle plots in Figures 2.4.(a) and (b) are ordered according to the initial permutation from the hierarchical clustering and the permutation returned from the GW algorithm respectively. The icicle plots are also modified from those described in Kruskal and Landwehr (1983) so that each node is coloured according to the mean of the Euclidean distances between adjacent objects in the node. The colour scale, light to dark green, represents low to high mean Euclidean distance.

In Figure 2.4.(a), the nodes N_{16} (left section), N_{12} (middle section) and N_{17} (right section) are highlighted using black lines. These three nodes correspond to the three clusters of irises shown in the heatmap in Figure 2.3.(a), where N_{16} and N_{17} correspond to the two more similar clusters of irises.

The icicle plot in Figure 2.4.(b) shows that the GW algorithm swaps N_{12}



(a) Initial order



(b) GW leaf order

Mean adjacent Euclidean distance

Figure 2.4: Icicle plots visualising a hierarchical clustering of a sample of the cases from the Iris dataset. The objects (columns) in the icicle plot in (a) are ordered according to the initial permutation from the dendrogram in Figure 2.3.(a). The objects (columns) in the icicle plot in (b) are ordered according to the dendrogram seriation algorithm described in Gruvaeus and Wainer (1972). The nodes are coloured according to the mean Euclidean distance between adjacent objects in the nodes. Light to dark green indicates low to high mean Euclidean distance between adjacent objects respectively. The same colour scale is used for both icicle plots.

and N_{17}. Comparing the layout of the "blocks" that make up N_{12} and N_{17} shows that the GW algorithm also rearranges various sub-nodes of N_{12} and N_{17}, which leads to noticeably lower mean adjacent Euclidean distance values for N_{12} and N_{17} (indicated by the lighter green colour of N_{12} and N_{17} in Figure 2.4.(b)). All of these node rearrangements lead to a noticeably lower mean adjacent Euclidean distance value for the node N_{18}, which merges the objects in N_{12} and N_{17}.

The GW algorithm also rearranges various sub-nodes of N_{16}, which is again observed by comparing the "blocks" that make up N_{16} in Figures 2.4.(a) and (b). These node rearrangements, combined with the node rearrangements of N_{12}, N_{17} and their sub-nodes, lead to the lighter green colour of the final node N_{19} in the icicle plot in Figure 2.4.(b), which indicates that the permutation returned by the GW algorithm has a shorter path length than the initial permutation from the hierarchical clustering.

2.5 Other seriation methods

Many different algorithms have been used for seriation. Figure 2.5 provides an overview and classification of several of these algorithms.

The rows in Figure 2.5 categorise the algorithms according to the seriation criterion (path length, anti-Robinson form or other) that they aim to optimise. The columns classify the algorithms into five categories of seriation method: (i) Dendrogram seriation, (ii) Travelling Salesperson Problem (TSP) heuristics, (iii) Partial enumeration, (iv) Dimension reduction and (v) a category containing algorithms that do not fit into the previous four categories.

Section 2.2 already discussed the Dendrogram seriation category of algorithms. This section gives a brief account of the other four categories of seriation methods in Figure 2.5.

2.5.1 Heuristics for the Travelling Salesperson Problem

The second column in Figure 2.5 gives an overview of heuristic algorithms for the Travelling Salesperson Problem (TSP). The TSP is a well known and well researched combinatorial optimisation problem (see, for example, Lawler et al. 1985). Given n cities, the goal of the TSP is to find the shortest tour that starts from a selected city, visits each city once and returns to the starting city.

The shortest path problem is similar to the TSP except there is no return to the starting city. In other words, the goal of the shortest path problem is to find a permutation of the cities that minimises the path length criterion.

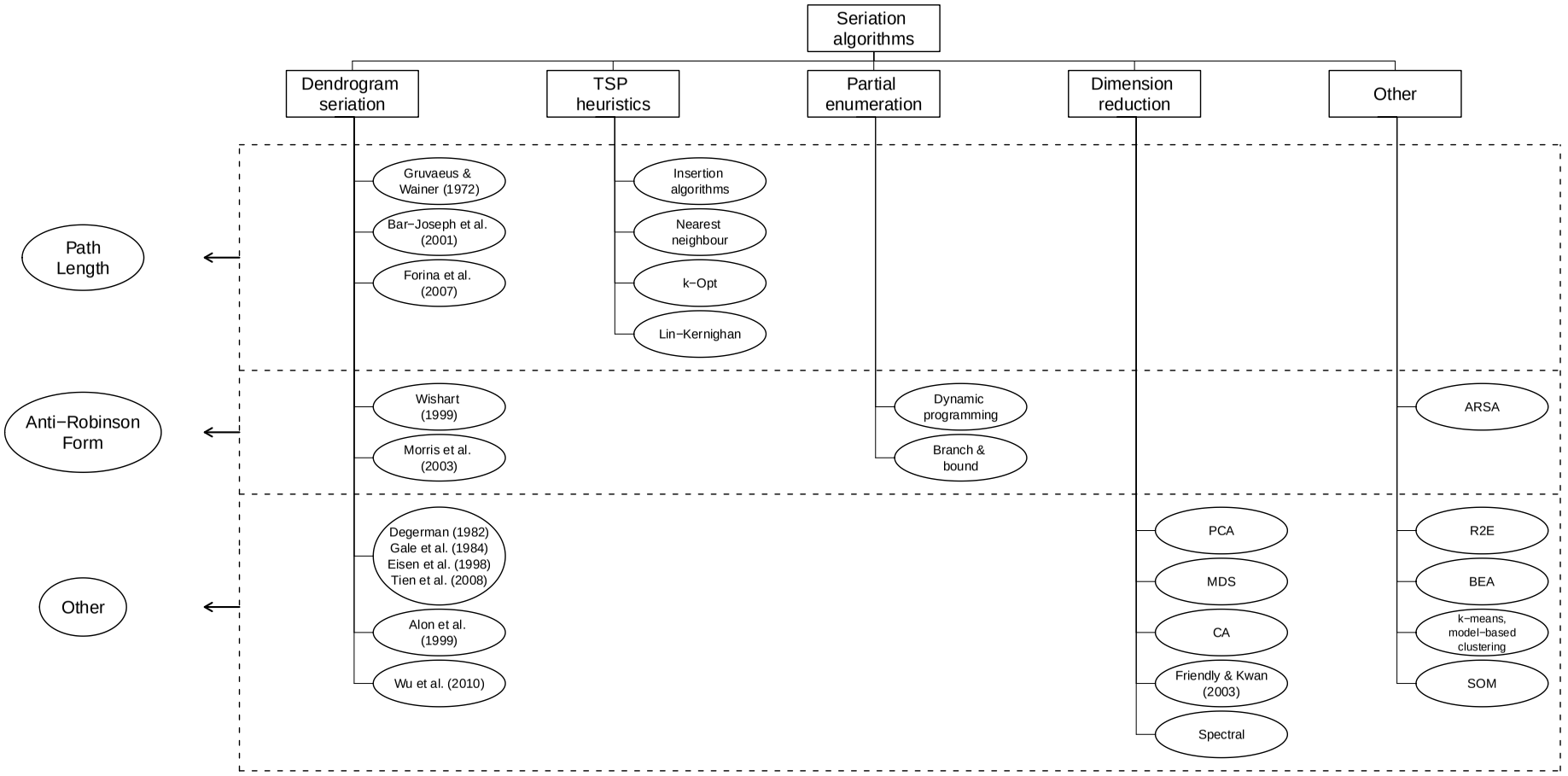


Figure 2.5: overview of seriation algorithms.

The TSP can be transformed into the shortest path problem by inserting a "dummy" city, which has zero dissimilarity to all other cities (Garfinkel 1985). The position of the dummy city in the TSP solution represents the cutting point for finding the shortest path. Through this simple transformation, algorithms designed for solving the TSP may also be used to solve the shortest path problem.

Finding the exact solution to a TSP with n cities is an NP-hard problem (see, for example, Garey and Johnson 1979). Small TSPs can be optimally solved using dynamic programming or branch-and-bound methods. However, for larger TSPs, it is more efficient to use heuristic algorithms to find good but not necessarily optimal solutions.

TSP heuristics divide into two groups: tour construction heuristics and tour improvement heuristics. Construction heuristics include nearest neighbour, repetitive nearest neighbour and various insertion heuristics (see, for example, Johnson and Papadimitriou 1985). Two improvement heuristics are k-Opt (see, for example, Croes 1958) and the Lin-Kernighan heuristic (Lin and Kernighan 1973). A general strategy for solving a TSP is to use a construction heuristic to create an initial tour and then improve the tour using an improvement heuristic.

The `R` package `TSP` (Hahsler and Hornik 2009) implements the TSP heuristic algorithms listed above.

2.5.2 Partial enumeration methods

The third column in Figure 2.5 lists two seriation algorithms, which are based on partial enumeration techniques. Both of these algorithms aim to optimise some anti-Robinson function.

For small numbers of objects, the optimal seriation solution can be found by listing and checking all possible permutations. However, Brusco and Stahl (2005) stated that this brute-force approach is currently infeasible for more than thirteen objects. Hubert et al. (2001) used dynamic programming methods to optimally seriate twenty or so objects, and Brusco and Stahl (2005) claimed that depending on the data, branch-and-bound methods can optimally seriate up to 40 objects.

Hubert et al. (2001, §2) described a seriation algorithm based on dynamic programming, where the goal is to optimise an anti-Robinson function. Dynamic programming is a method for efficiently solving optimisation problems. The method involves storing results of sub-problem calculations, which the algorithm then re-uses. Avoiding the re-calculation of the sub-problems sig-

nificantly reduces the time required to search for the optimal solution. However, even with this short-cut, dynamic programming is only suitable for small seriation problems because of its large memory requirements for storing the sub-problem results (Hubert et al. 2001, pg. 88).

Branch-and-bound is an alternative approach to dynamic programming. Brusco and Stahl (2005) described a forward branching algorithm for finding a permutation that optimises an anti-Robinson function. Their algorithm builds permutations using an iterative selection of objects for the next position in the permutation. At each step, the algorithm tests a partial permutation to see if it can lead to a better solution than a current best solution. When a partial permutation fails a particular test, the algorithm abandons the partial permutation and investigates a new partial permutation.

The branch-and-bound algorithm from Brusco and Stahl (2005) is available in the `R` `seriation` package (Hahsler et al. 2010)

2.5.3 Dimension reduction techniques

The fourth column in Figure 2.5 lists several seriation algorithms that are based on dimension reduction techniques such as principal components analysis (PCA), correspondence analysis (CA) and multidimensional scaling (MDS). For example, Kendall (1971) used MDS techniques to seriate tombs in a cemetery at Münsingen-Rain.

Friendly and Kwan (2003) used algorithms based on eigen decompositions to demonstrate their idea of "effect-ordered data displays". They used CA to seriate the categories in a mosaic display and they seriated the variables in parallel coordinates plots according to their weight on the first principal component.

Friendly and Kwan (2003) also described a "correlation ordering" algorithm, which seriates variables according to the angles formed by the first two eigenvectors of the correlation matrix. They used this algorithm to seriate the rays in star glyphs and also to seriate corrgrams (Friendly 2002), which are color-coded mappings of correlation matrices. The correlation ordering algorithm is implemented in the `R` package `corrgram` (Wright 2006).

Atkins et al. (1998) described a seriation algorithm based on spectral analysis. They constructed the Laplacian of a dissimilarity matrix and found the eigenvector corresponding to the smallest non-zero eigenvalue. They then sorted the elements in the eigenvector to get a seriation of the objects.

Hastie et al. (2009) discussed clustering methods based on spectral analysis. Standard clustering methods have proven useful in seriation and so spec-

tral clustering and its near relative kernal principal components could also be investigated as seriation algorithms.

2.5.4 Other seriation algorithms

This section summarises the seriation algorithms listed in the fifth column in Figure 2.5, which are algorithms that do not fall into the previous categories.

Simulated annealing

Brusco et al. (2007) described a simulated annealing algorithm called ARSA, which tries to maximise a merit function that measures anti-Robinson form (namely the ARc function in Equation 2.3). The algorithm begins with an arbitrary permutation and creates a new permutation by randomly deciding to either swap two objects or relocate a single object in the permutation. If the new permutation increases the merit function then it is accepted and the algorithm continues by creating a new permutation from the accepted permutation. If the new permutation decreases the merit function then it is accepted with probability inversely proportional to the amount by which the merit function is decreased. The algorithm continues until it reaches an upper limit for the amount of permutations that it can examine or reject.

The logic for allowing the algorithm to accept permutations that decrease the merit function is to avoid a solution that is a local optima, which is a permutation that is worse than the global maximum but better than any of its neighbours.

The ARSA algorithm is available in the `R` package `seriation` (Hahsler et al. 2010).

Rank-two ellipse seriation

Consider a sequence of correlation matrices $R = (R^{(1)}, R^{(2)}, \ldots)$, where $R^{(1)}$ is the correlation matrix of a symmetric dissimilarity matrix D and $R^{(i)}$ is the correlation matrix of $R^{(i-1)}$, $i > 1$. Chen (2002) presented a "rank-two ellipse" (R2E) seriation algorithm based on this sequence of correlation matrices. He described how the rank of $R^{(i)}$ reduces as i increases and when $R^{(i)}$ reaches rank two, the objects fall on an ellipse in two-dimensional space. The R2E algorithm then orders the objects based on their position on the ellipse. Chen used his R2E seriation algorithm to rearrange heatmaps and presented an example in which the R2E algorithm is useful for finding approximate anti-Robinson form in dissimilarity matrices.

The R2E seriation algorithm is implemented in the software package GAP (Wu et al. 2010) and also in the `R` package `seriation` (Hahsler et al. 2010).

Bond energy algorithm

The Bond Energy Algorithm (BEA) (McCormick et al. 1972) is a greedy heuristic for separately seriating the rows and columns of a two-way two-mode $n \times p$ matrix A. The goal of the BEA is to group large elements of A together and McCormick et al. (1972) proposed to measure this using the so called measure of effectiveness (ME):

$$\text{ME}(A) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{p} a_{i,j}[a_{i,j+1} + a_{i,j-1} + a_{i+1,j} + a_{i-1,j}]. \tag{2.7}$$

McCormick et al. (1972) used the BEA to seriate binary and non-binary data. However, Arabie and Hubert (1990) questioned the use of the BEA on non-binary data.

The BEA is available in the `R` package `seriation` (Hahsler et al. 2010). This package also implements the use of TSP heuristics for maximising the ME.

Clustering based methods

Section 2.2 discussed seriation algorithms based on hierarchical clustering, however seriation methods can also be based on other clustering algorithms such as k-means or model-based clustering (Fraley and Raftery 1999, 2002). For example, Hahsler et al. (2008) used the ARSA (Brusco et al. 2008) seriation algorithm to reorder within the clusters and also between the clusters of a k-means solution in order to obtain a seriation of a set of objects.

Self organising maps (SOMs; Kohonen 1984) can also be used in conjunction with seriation algorithms. SOMs arrange a set of objects into sections of a k dimensional array, where k is usually one or two. One may view SOMs as a clustering method similar to k-means because SOMs place similar objects in the same section and so the sections are equivalent to clusters. Given the results of a SOM, one may reorder the objects within each section of the array and reorder the sections themselves using some seriation algorithm such as ARSA in order to obtain a permutation of a set of objects.

2.6 Summary

This chapter presented an overview of seriation methods and criteria with Figure 2.5 providing a visual overview of several seriation algorithms.

Section 2.2 focussed on dendrogram seriation, including a brief account of the first dendrogram seriation algorithm (Gruvaeus and Wainer 1972) and a discussion of various reasons for the increasing popularity of dendrogram seriation. Section 2.4 also described an example demonstrating the use of dendrogram seriation algorithms.

Section 2.3 concerned seriation criteria and gave a detailed description of path length and anti-Robinson form, which are two of the more frequently used seriation criteria.

Section 2.5 continued the categorisation of seriation methods and gave a brief summary of several algorithms. Later in this thesis, Chapter 5 compares the performance of many of the seriation algorithms included in Figure 2.5.

Chapter 3

DendSer: a new dendrogram seriation algorithm

3.1 Introduction

This chapter presents a new dendrogram seriation algorithm called DendSer.

DendSer allows the user to choose different methods for rearranging the nodes in a dendrogram. Rearranging nodes, whether exchanging branches or reversing the order of the leaves in a node, is a key aspect of dendrogram seriation, however it receives little attention in the literature. Section 3.3 develops notation and terminology for describing how to rearrange a node and defines several node operations, which rearrange or operate on nodes in different ways.

Existing seriation algorithms focus on optimising one seriation criterion only. DendSer is more flexible because it allows the user to choose from a variety of seriation criteria or to input their own criteria. Section 3.4 describes the different criteria that the user may choose from, which include path length, anti-Robinson form and two new seriation criteria called "lazy path length" and "banded anti-Robinson" form.

Section 3.5 highlights how DendSer provides a general framework for implementing several other dendrogram seriation algorithms (Gruvaeus and Wainer 1972, Degerman 1982, Gale et al. 1984, Eisen et al. 1999, Tien et al. 2008, Alon et al. 1999, Wishart 1999 and Wu et al. 2010). However, other dendrogram seriation algorithms (Bar-Joseph et al. 2001, Morris et al. 2003 and Forina et al. 2007) do not fit into the DendSer framework.

Section 3.6 investigates which of the node operations defined in Section 3.3 is the most suitable for use in DendSer when optimising different seriation criteria. For each of four criteria, Section 3.6.2 performs a simulation study in

order to examine both the goodness of the permutations returned by DendSer when used with different node operations and the amount of work each node operation creates for DendSer.

The chapter ends with a brief summary.

3.2 A new dendrogram seriation algorithm

The pseudo-code for a new dendrogram seriation algorithm called DendSer is given below.

Algorithm 1 DendSer: Dendrogram seriation algorithm

Require: Dendrogram Δ, cost function F, node operation T
  $n \leftarrow$ # leaves in Δ
  maxloops $\leftarrow$ maximum # iterations
  nloops $\leftarrow 1$
  $\pi^* \leftarrow$ initial permutation of leaves in Δ
  change $\leftarrow$ TRUE
  while change **do**
    $\pi_{cur} \leftarrow \pi^*$
    for $i = 1$ **to** $n - 1$ **do**
      $N_i \leftarrow i^{th}$ node in Δ
      $\{\pi_1, \ldots, \pi_k\} \leftarrow$ set of permutations returned by $T(N_i; \Delta)$
      $(F_1, \ldots, F_k) \leftarrow F(\pi_1), \ldots, F(\pi_k)$
      $\pi_{new} \leftarrow \text{argmin}(F_1, \ldots, F_k)$
      if $F(\pi_{new}) < F(\pi_{cur})$ **then**
        update Δ according to π_{new}
        $\pi_{cur} \leftarrow \pi_{new}$
      end if
    end for
    if ($\pi_{cur} = \pi^*$ **or** nloops = maxloops) **then**
      change $\leftarrow$ FALSE
    else
      $\pi^* \leftarrow \pi_{cur}$
      nloops $\leftarrow$ nloops $+ 1$
    end if
  end while
  return π^*

DendSer takes in a dendrogram Δ, a cost function F and a node operation T. The cost function F takes in a permutation and either a symmetric dissimilarity matrix or a vector of weights and evaluates the permutation using some seriation criterion (see Section 3.4). A node operation T takes in a dendrogram Δ and a node N and returns a set containing permutations of the leaves in Δ corresponding to some operation on N (see Section 3.3).

DendSer selects each node N in turn, starting at the first node formed by the hierarchical clustering process and ending at the root node. The algorithm then evaluates the original permutation of the leaves and the permutations returned by $T(N; \Delta)$. The permutation minimising the cost function F is kept and Δ is updated according to this permutation. The algorithm then moves onto the next node. One iteration is complete when the algorithm has applied T to all of the $n-1$ nodes. If an iteration results in an improved permutation of the leaves, then the algorithm goes through another iteration. The algorithm stops when a full iteration fails to improve the permutation of the leaves or the maximum number of iterations is reached.

DendSer provides a number of choices for the node operation T and the seriation criterion F, which Sections 3.3 and 3.4 now describe.

3.3 Node operations

Dendrogram seriation algorithms are generally greedy algorithms and work by examining one or two nodes at a time, with the goal of finding permutations of the leaves that improve some criterion. For example, as shown in Figure 2.2, the algorithm described in Gruvaeus and Wainer (1972) examines two nodes at a time in order to place the most similar edge leaves adjacently.

There are many ways of rearranging the nodes in a dendrogram and existing dendrogram seriation algorithms use different methods that offer different permutations. However, researchers rarely provide much information about their chosen method. Different papers also use different terminology for describing dendrogram seriation. Due to both the lack of information and inconsistent terminology, the reader sometimes finds it difficult to understand how a particular algorithm works and also how it differs from other algorithms.

This section develops unifying terminology for describing dendrogram seriation algorithms and defines several node operations, which rearrange or operate on nodes in different ways. Each of these node operations provides a different choice for the node operation T in DendSer.

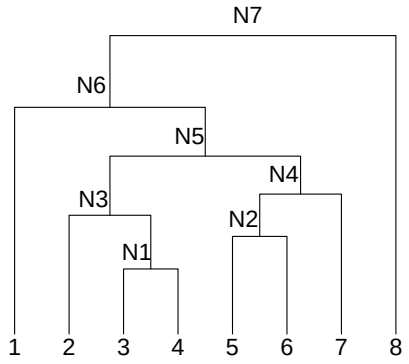
3.3.1 Reflection and translation

The following are two ways of rearranging or operating on a node in a dendrogram:

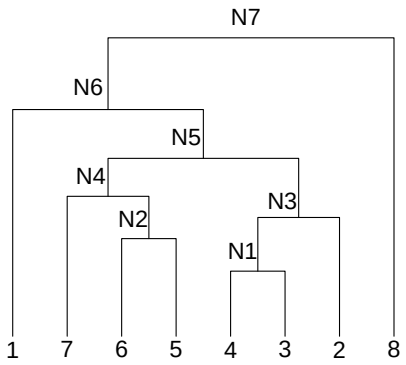
Definition 3.1. *The reflection of a node N in a dendrogram reverses the order of the leaves in N.*

Definition 3.2. *Let N_l and N_r be the left and right sub-nodes of a node N in a dendrogram. The translation of N swaps the positions of N_l and N_r but does not change the order of the leaves in N_l and N_r.*

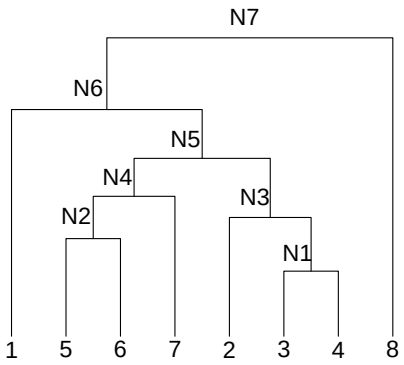
Forina et al. (2007) first used the term translation. Reflecting or translating a node N does not change the hierarchy represented by N, only the order of the leaves in N. For example, consider the dendrogram Δ in Figure 3.1.(a) with the nodes labelled N_1, N_2,..., N_7 according to their height. Figure 3.1.(b) shows the dendrogram Δ with the node N_5 reflected and Figure 3.1.(c) shows the dendrogram Δ with the node N_5 translated.



(a) Δ: Initial dendrogram



(b) Reflect N_5



(c) Translate N_5

Figure 3.1: The node N_5 in the dendrogram in (a) is reflected in the dendrogram in (b) and translated in the dendrogram in (c).

For a node N in a dendrogram Δ, the node operations R_0 and T_0 return the following sets of permutations:

$$R_0(N;\Delta) = \{\text{permutation of leaves in } \Delta \text{ after reflecting } N\}, \quad (3.1)$$

$$T_0(N;\Delta) = \{\text{permutation of leaves in } \Delta \text{ after translating } N\}. \quad (3.2)$$

For example, consider again the dendrogram Δ in Figure 3.1.(a). $R_0(N_5;\Delta)$

returns a set containing the permutation of leaves shown in the dendrogram in Figure 3.1.(b) and $T_0(N_5;\Delta)$ returns a set containing the permutation of leaves shown in the dendrogram in Figure 3.1.(c), i.e.

$$\begin{aligned} R_0(N_5;\Delta) &= \{(1,\underline{7,6,5,4,3,2},8)\}, \\ T_0(N_5;\Delta) &= \{(1,\underline{5,6,7,2,3,4},8)\}. \end{aligned}$$

Note that if a node N in a dendrogram Δ has only two leaves, then $R_0(N;\Delta) = T_0(N;\Delta)$. For convenience, $R_0(N;\Delta)$ also refers to the dendrogram Δ with N reflected. Similarly, $T_0(N;\Delta)$ also refers to the dendrogram Δ with N translated.

3.3.2 Extensions of R_0 and T_0

R_0 and T_0 operate directly on a node N and so they operate on N at a depth of zero. The following defines two node operations that operate on N at a depth of one, i.e. the left and right sub-nodes of N.

For a node N with three or more leaves and sub-nodes N_l and N_r in a dendrogram Δ, $R_1(N;\Delta)$ returns a set containing up to three unique permutations of the leaves in Δ: one corresponding to the reflection of N_l, one corresponding to the reflection of N_r and one corresponding to the reflection of both N_l and N_r. Similarly, $T_1(N;\Delta)$ returns a set containing up to three unique permutations of the leaves in Δ: one corresponding to the translation of N_l, one corresponding to the translation of N_r and one corresponding to the translation of both N_l and N_r. These node operations may be written as follows:

$$R_1(N;\Delta) = R_0(N_l;\Delta) \cup R_0(N_r;\Delta) \cup R_0(N_l,N_r;\Delta), \tag{3.3}$$

$$T_1(N;\Delta) = T_0(N_l;\Delta) \cup T_0(N_r;\Delta) \cup T_0(N_l,N_r;\Delta). \tag{3.4}$$

In Equation 3.3, $R_0(N_l,N_r;\Delta)$ returns a set containing the permutation of the leaves in Δ after first reflecting N_l and then reflecting N_r. Similarly, in Equation 3.4, $T_0(N_l,N_r;\Delta)$ returns a set containing the permutation of the leaves in Δ after first translating N_l and then translating N_r. Note that, although $R_0(N_l,N_r;\Delta)$ and $T_0(N_l,N_r;\Delta)$ are defined to operate on the nodes N_l and N_r in a specific order, changing this order does not affect the returned permutation (see Properties 1 and 2 in Section 3.3.3).

Note that, if either of the left and right sub-nodes of a node N has only one leaf, then $R_1(N,\Delta)$ and $T_1(N,\Delta)$ each return a set containing only two unique

permutations of the leaves in a dendrogram Δ. This is because reflecting or translating a node with only one leaf does not rearrange the node.

Applying R_1 and T_1 to N_5 in the dendrogram Δ in Figure 3.1.(a) returns the following sets of permutations of the leaves in Δ:

$$\begin{aligned} R_1(N_5;\Delta) &= R_0(N_3;\Delta) \cup R_0(N_4;\Delta) \cup R_0(N_3,N_4;\Delta) \\ &= \{(1,\underline{4,3,2},5,6,7,8),\ (1,2,3,4,\underline{7,6,5},8),\ (1,\underline{4,3,2},\underline{7,6,5},8)\}, \end{aligned}$$

$$\begin{aligned} T_1(N_5;\Delta) &= T_0(N_3;\Delta) \cup T_0(N_4;\Delta) \cup T_0(N_3,N_4;\Delta) \\ &= \{(1,\underline{3,4,2},5,6,7,8),\ (1,2,3,4,\underline{7,5,6},8),\ (1,\underline{3,4,2},\underline{7,5,6},8)\}. \end{aligned}$$

The following two node operations operate on a node N at both depths of zero and one. $R_{01}(N;\Delta)$ returns a set containing up to seven unique permutations of the leaves in Δ corresponding to all possible combinations of reflecting N, N_l and N_r. Similarly, $T_{01}(N;\Delta)$ returns a set containing up to seven unique permutations of the leaves in Δ corresponding to all possible combinations of translating N, N_l and N_r. These node operations may be written as follows:

$$R_{01}(N;\Delta) = R_0(N;\Delta) \cup R_1(N;\Delta) \cup R_1(N;R_0(N;\Delta)), \tag{3.5}$$

$$T_{01}(N;\Delta) = T_0(N;\Delta) \cup T_1(N;\Delta) \cup T_1(N;T_0(N;\Delta)). \tag{3.6}$$

In Equation 3.5, the notation $R_1(N;R_0(N;\Delta))$ means apply R_1 to the node N in the dendrogram represented by $R_0(N;\Delta)$, which is the dendrogram Δ with N reflected. Similarly, in Equation 3.6, $T_1(N;T_0(N;\Delta))$ means apply T_1 to the node N in the dendrogram represented by $T_0(N;\Delta)$, which is the dendrogam Δ with N translated. Note that if the sub-nodes, N_l and N_r, of a node N in a dendrogram Δ have one or two leaves each, then $R_{01}(N;\Delta) = T_{01}(N;\Delta)$.

Applying R_{01} and T_{01} to N_5 in the dendrogram Δ in Figure 3.1.(a) returns the following sets of permutations of the leaves in Δ:

$$\begin{aligned} R_{01}(N_5;\Delta) &= R_0(N_5;\Delta) \cup R_1(N_5;\Delta) \cup R_1(N_5;R_0(N_5;\Delta)) \\ &= \{(1,\underline{7,6,5,4,3,2},8), \\ &\quad (1,\underline{4,3,2},5,6,7,8),\ (1,2,3,4,\underline{7,6,5},8),\ (1,\underline{4,3,2},\underline{7,6,5},8), \\ &\quad (1,\underline{5,6,7},4,3,2,8),\ (1,7,6,5,\underline{2,3,4},8),\ (1,\underline{5,6,7},\underline{2,3,4},8)\}, \end{aligned}$$

$$\begin{aligned} T_{01}(N_5;\Delta) &= T_0(N_5;\Delta) \cup T_1(N_5;\Delta) \cup T_1(N_5;T_0(N_5;\Delta)) \\ &= \{(1,\underline{5,6,7,2,3,4},8), \end{aligned}$$

$$(1, \underline{3,4,2}, 5, 6, 7, 8),\ (1, 2, 3, 4, \underline{7,5,6}, 8),\ (1, \underline{3,4,2}, \underline{7,5,6}, 8),$$
$$(1, \underline{7,5,6}, 2, 3, 4, 8),\ (1, 5, 6, 7, \underline{3,4,2}, 8),\ (1, \underline{7,5,6}, \underline{3,4,2}, 8)\}.$$

Node operations may be extended to operate on nodes at any combination of depths, not just zero and one. The most extreme node operation is one that returns a set containing unique permutations of the leaves in a dendrogram corresponding to all possible combinations of reflecting or translating the nodes at *all* depths of a node N.

The final node operation defined in this section uses both reflection and translation and is simply the union of the results of R_0 and T_0:

$$C_0(N;\Delta) \quad = \quad R_0(N;\Delta) \cup T_0(N;\Delta). \tag{3.7}$$

Applying C_0 to N_5 in the dendrogram Δ in Figure 3.1.(a) returns the following set of permutations of the leaves in Δ:

$$\begin{aligned} C_0(N_5;\Delta) &= R_0(N_5;\Delta) \cup T_0(N_5;\Delta) \\ &= \{(1, \underline{7,6,5,4,3,2}, 8),\ (1, \underline{5,6,7,2,3,4}, 8)\}. \end{aligned}$$

DendSer allows the user to choose the node operation T from all seven node operations defined in this section: R_0, T_0, R_1, T_1, R_{01}, T_{01} and C_0. Some of these node operations are new and some are used in other dendrogram seriation algorithms:

- The algorithm described in Gruvaeus and Wainer (1972) used the node operation R_1.
- The algorithms described in Degerman (1982), Gale et al. (1984), Eisen et al. (1998) and Tien et al. (1998) used the node operation T_0.
- The algorithms described in Wishart (1999) and Morris et al. (2003) used the node operation R_0.

Section 3.6 investigates the suitability of these node operations when using DendSer to optimise different seriation criteria.

3.3.3 Properties of node operations

The following is a list of properties of the node operations defined in Sections 3.3.1 and 3.3.2. In each of the following, N is a node in a dendrogram Δ. See Appendix A for proofs of these properties.

Property 1. The order in which R_0 operates on two nodes N_a and N_b does not affect the returned permutation, i.e.

$$R_0(N_b; R_0(N_a; \Delta)) = R_0(N_a; R_0(N_b; \Delta)). \tag{3.8}$$

Equation 3.8 may be written using the following simpler notation:

$$R_0(N_a, N_b; \Delta) = R_0(N_b, N_a; \Delta). \tag{3.9}$$

Equation 3.9 extends to any number of nodes.

Property 2. The order in which T_0 operates on two nodes N_a and N_b does not affect the returned permutation, i.e.

$$T_0(N_b; T_0(N_a; \Delta)) = T_0(N_a; T_0(N_b; \Delta)). \tag{3.10}$$

Equation 3.10 may be written using the following simpler notation:

$$T_0(N_a, N_b; \Delta) = T_0(N_b, N_a; \Delta). \tag{3.11}$$

Equation 3.11 extends to any number of nodes.

Property 3. The following relationships hold between R_0 and T_0:

$$T_0(N; \Delta) = R_0(N, N_l, N_r; \Delta), \tag{3.12}$$

where N_l and N_r are the left and right sub-nodes of N.

$$R_0(N; \Delta) = T_0(N_a, \ldots, N_k, N; \Delta), \tag{3.13}$$

where $N_a, \ldots, N_k$ are all of the descendant sub-nodes of N.

Property 4. The sets of permutations returned by some node operations are subsets of the sets of permutations returned by other node operations:

(a) $T_0(N; \Delta) \subseteq R_{01}(N; \Delta)$.

(b) $R_0(N; \Delta) \subseteq R_{01}(N; \Delta)$.

(c) $R_1(N; \Delta) \subseteq R_{01}(N; \Delta)$.

(d) $T_0(N; \Delta) \subseteq T_{01}(N; \Delta)$.

(e) $T_1(N; \Delta) \subseteq T_{01}(N; \Delta)$.

(f) $C_0(N; \Delta) \subseteq R_{01}(N; \Delta)$.

(g) $R_0(N;\Delta) \subseteq C_0(N;\Delta)$.

(h) $T_0(N;\Delta) \subseteq C_0(N;\Delta)$.

Property 5. The node operations R_0 and R_1 are commutative, i.e.

$$R_1(N;R_0(N;\Delta)) = R_0(N;R_1(N;\Delta)). \tag{3.14}$$

Property 6. The node operations T_0 and T_1 are commutative, i.e.

$$T_1(N;T_0(N;\Delta)) = T_0(N;T_1(N;\Delta)). \tag{3.15}$$

3.4 Seriation criteria

Existing seriation algorithms focus on one seriation criterion. For example, the algorithms from Bar-Joseph et al. (2001), Forina et al. (2007) and TSP heuristics (see Section 2.5.1) minimise the path length of permutations, while the algorithms in Brusco et al. (2008), Wishart (1999) and Morris et al. (2003) focus on anti-Robinson form.

DendSer is more flexible and provides a number of choices for the seriation criterion F. These choices include the path length and anti-Robinson criteria and also new criteria described in the following subsections. This thesis uses the following cost function for the path length criterion:

$$\mathrm{PL}(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)}. \tag{3.16}$$

For measuring anti-Robinson form, this thesis uses the following cost function version of the ARc merit function described in Hubert et al. (2006) (see Equation 2.3):

$$\begin{aligned} \mathrm{ARc}(\pi) &= \sum_{|i-j|\leqslant n-1} n d_{\pi(i),\pi(j)} - \sum_{|i-j|\leqslant n-1} |i-j| d_{\pi(i),\pi(j)} && (3.17)\\ &= \sum_{|i-j|\leqslant n-1} (n-|i-j|) d_{\pi(i),\pi(j)}. && (3.18) \end{aligned}$$

Alternatively, the user may choose from the criteria provided by the `R` package `seriation` (Hahsler et al. 2010) or they may input their own user-defined measure.

3.4.1 Lazy path length

The lazy path length criterion, as the name suggests, is a variation of the path length criterion. Given a set of objects with dissimilarities between them, the goal is to find a permutation of the objects that:

1. has a short path length and

2. has the dissimilarities between adjacent objects generally increasing.

The following cost function measures how well a permutation satisfies this desired criterion:

Definition 3.3. *Consider a set of n objects, where $d_{i,j}$ is the dissimilarity value for objects i and j. For a permutation π of the objects, the lazy path length cost function is defined as follows:*

$$LPL(\pi) = \sum_{i=1}^{n-1}(n-i)d_{\pi(i),\pi(i+1)}. \tag{3.19}$$

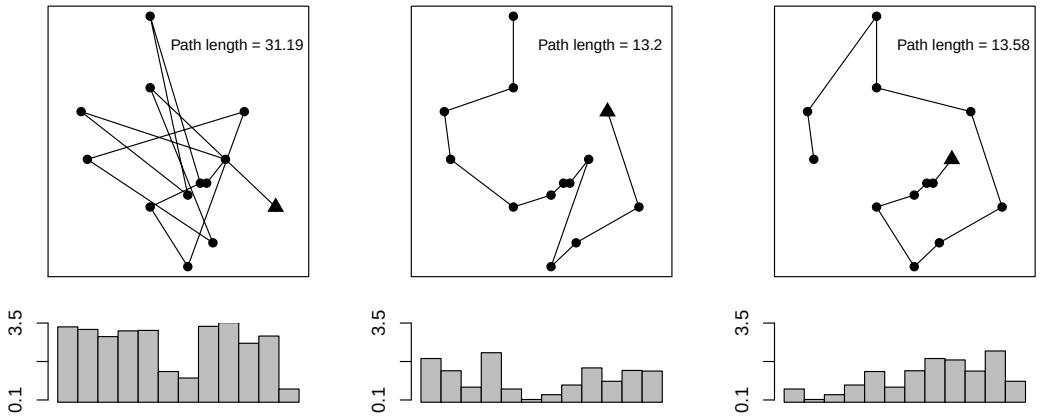
The LPL cost function is a weighted measure of the path length of a permutation, where dissimilarities at the beginning of the permutation are given more weight than dissimilarities near the end of the permutation. The LPL cost function is $O(n)$ and so is fast to compute.

Figure 3.2 illustrates an application of the lazy path length criterion. The path through the points in Figure 3.2.(a) corresponds to an arbitrary permutation of thirteen randomly generated points. The triangle indicates where the permutation begins and the heights of the bars beneath the scatterplot represent the Euclidean distances between adjacent points in the permutation.

The scatterplot in Figure 3.2.(b) shows the path corresponding to the permutation obtained by using DendSer with the PL cost function and the node operation R_{01}. The path length of this permutation is 13.2 and the bars show that the smallest Euclidean distances occur in the middle of the permutation.

Applying DendSer with the LPL cost function and the node operation R_{01} returns the path through the points shown in Figure 3.2.(c). The bars show that the smallest Euclidean distances are positioned close to the beginning of the permutation and the path length of the permutation is still short with a value of 13.58.

The lazy path length criterion may also be described in terms of the Travelling Salesperson Problem. The goal of the Travelling Salesperson Problem is for a salesperson to visit a set of cities by travelling as short a distance as possible. With the lazy path length criterion (ignoring the return to the starting



(a) Arbitrary (b) Shortest path (c) Lazy path

Figure 3.2: The paths through the points in (a), (b) and (c) correspond to an arbitrary permutation, a permutation returned by using DendSer with the PL cost function and a permutation returned by using DendSer with the LPL cost function, respectively. The triangles in the scatterplots indicate where the permutations begin and the bars beneath the scatterplots represent the Euclidean distances between adjacent objects in the corresponding permutations.

city), the salesperson still wants to travel as short a distance as possible but yet he is somewhat lazy in that he tries to visit the next closest city at each step. Hence the name *lazy* path length.

Note that the lazy path length criterion is related to some variations of the TSP, which also aim to satisfy two goals (see, for example, Gutin and Punnen 2002). However, the proposed lazy path length criterion described here is not discussed elsewhere.

The motivation for developing the lazy path length criterion comes from Hurley (2004), who discussed the visualisation concept of placing interesting features in prominent positions of statistical graphics. For example, Hurley (2004) seriated variables in scatterplot matrices so that interesting panels were positioned close to the main diagonal, which made it easier to observe trends and groups in data. Sections 4.1 and 4.3 present examples where seriating variables using the lazy path length criterion is effective in making interesting features more prominent in parallel coordinates plots and scatterplot matrices.

3.4.2 Banded anti-Robinson form

This section introduces a new seriation criterion called banded anti-Robinson form, which is a variation of anti-Robinson form.

Section 2.3.2 described anti-Robinson (AR) form and briefly outlined sev-

eral functions for measuring AR form in dissimilarity matrices. Optimising AR form in a dissimilarity matrix aims to fit *every* element of the matrix into a specific pattern. However, this pattern is quite strict and may be too rigid for some dissimilarity matrices. Therefore, this section proposes to "relax" AR form and instead use *banded* anti-Robinson form, which may be described as a hybrid of the path length and anti-Robinson criteria.

As with AR form, a matrix following banded AR form has small values close to the main diagonal, but only the values within a band of width w around the main diagonal satisfy AR form.

Definition 3.4. *A symmetric matrix $D = [d_{i,j}]$, for $1 \leqslant i, j \leqslant n$, has banded anti-Robinson form if for a band-width w with $w < n$:*

- $d_{i,k} \leqslant d_{i,j}$ *and* $d_{k,j} \leqslant d_{i,j}$, *for* $i < k < j$ *and* $|i - j| \leqslant w$.
- $d_{i,j} \leqslant d_{i',j'}$, *for* $|i - j| \leqslant w$ *and* $|i' - j'| > w$.

The following cost function measures how well a permutation satisfies banded AR form:

Definition 3.5. *Consider a set of n objects, where $d_{i,j}$ is the dissimilarity value between objects i and j and let w be the band-width, where $w < n$. For a permutation π of the objects, the banded anti-Robinson cost function is defined as:*

$$BAR(\pi) = \sum_{|i-j| \leqslant w} (w + 1 - |i - j|) d_{\pi(i),\pi(j)}. \tag{3.20}$$

The BAR cost function may be computed in $O(nw)$ time. Tien et al. (2008) also defined a function for measuring local AR form, which counts the number of AR violations in a band of width w around the main diagonal of a dissimilarity matrix.

This thesis uses $w = \lfloor \frac{n}{5} \rfloor$ for the BAR cost function because this value seems to work well in practice. Note that if $w = 1$, then the BAR cost function is equal to the PL cost function in Equation 3.16 because if $w = 1$, then

$$\text{BAR}(\pi) = \sum_{|i-j| \leqslant 1} (1 + 1 - 1) d_{\pi(i),\pi(j)}, \tag{3.21}$$

which is written more simply as:

$$\text{BAR}(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)}. \tag{3.22}$$

If the band-width covers the entire dissimilarity matrix, then $w = n - 1$ and so

$$\text{BAR}(\pi) = \sum_{|i-j| \leqslant n-1} (n - 1 + 1 - |i - j|) d_{\pi(i),\pi(j)}, \tag{3.23}$$

which simplifies to

$$\text{BAR}(\pi) = \sum_{|i-j| \leqslant n-1} (n - |i - j|) d_{\pi(i),\pi(j)}. \tag{3.24}$$

Therefore, when $w = n - 1$ the BAR cost function is equal to the ARc cost function in Equation 3.18.

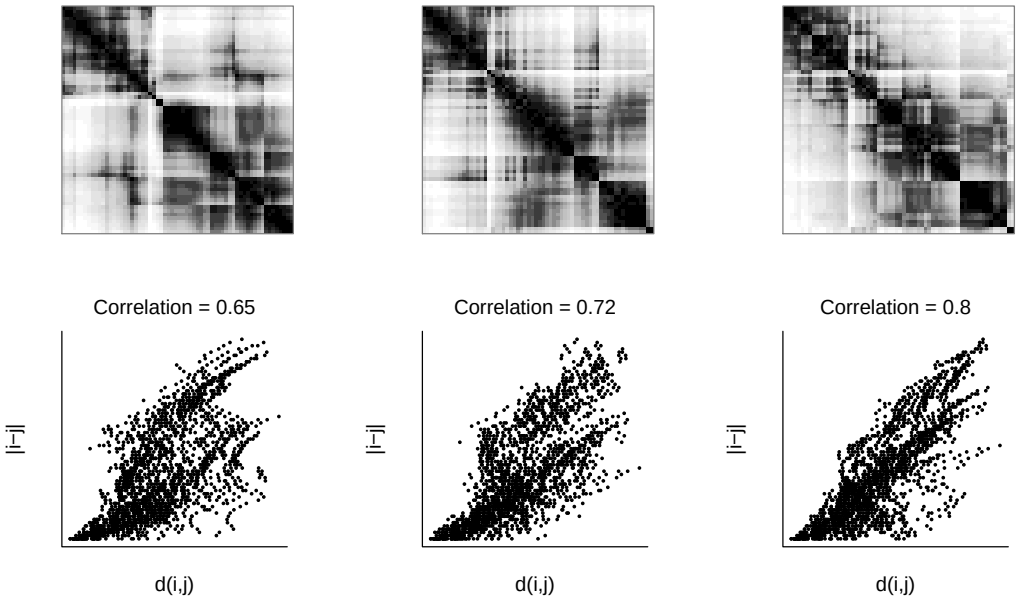
The following example demonstrates how the banded anti-Robinson criterion is a compromise between the path length and anti-Robinson criteria.

The Laser dataset is available in the `R` package `tourr` (Cook and Wickham 2010) and comes from an experiment at Bellcore, where physicists investigated the performance of a laser. The dataset contains 64 rows and four variables, which are the current applied to the laser (front and back), and the power and wavelength output of the laser.

After standardising the variables, the rows of the Laser dataset are hierarchically clustered using Euclidean distance and average linkage. The dendrogram is then seriated using DendSer with each of the cost functions PL, BAR and ARc. The node operations used in DendSer are R_{01} for PL and BAR, and T_0 for ARc. Section 3.6 justifies these choices of node operations.

Figures 3.3.(a), (b) and (c) show heatmaps of the Euclidean distance matrix, whose rows/columns are ordered according to the permutations returned by DendSer with each of PL, BAR and ARc respectively. The colour scale, black to white, represents low to high Euclidean distances. Beneath each of the heatmaps is a corresponding scatterplot of $d_{i,j}$ versus $|i - j|$ with the Pearson correlation of $d_{i,j}$ and $|i - j|$ written above the scatterplots. These scatterplots are referred to as Shepard plots (see, for example, Cox and Cox 1994, pg. 54). For each seriation, this example reports the Pearson correlation value instead of the ARc value because the correlation values are more meaningful.

The Shepard plots and correlations show that, of the three criteria, DendSer with ARc results in the Euclidean distance matrix (Figure 3.3.(c)) being the closest to anti-Robinson form, while DendSer with PL results in the Euclidean distance matrix (Figure 3.3.(a)) being the furthest from anti-Robinson form. DendSer with BAR results in the Euclidean distance matrix (Figure 3.3.(b)) being closer to anti-Robinson form than the PL seriation of the matrix but not as close as the ARc seriation of the matrix.



(a) DendSer + PL (b) DendSer + BAR (c) DendSer + ARc

Figure 3.3: Heatmaps of the Euclidean distance matrix for the rows in the Laser dataset. The heatmaps in (a), (b) and (c) are ordered according to the permutations returned by using DendSer with each of PL, BAR and ARc, respectively. The colour scale black to white represents low to high Euclidean distances. Beneath each heatmap is a corresponding scatterplot (Shepard plot) of $d_{i,j}$ versus $|i-j|$ with the Pearson correlation of $d_{i,j}$ and $|i-j|$ written above the scatterplots.

This example also suggests that anti-Robinson form may be too rigid a structure for some dissimilarity matrices. DendSer with ARc results in the heatmap of the Euclidean distance matrix (Figure 3.3.(c)) displaying lots of white squares close to the main diagonal. This "chess-board" effect makes it difficult to discern any structure in the Euclidean distance matrix. On the other hand, DendSer with BAR results in a visually "smoother" heatmap of the Euclidean distance matrix (Figure 3.3.(b)).

Section 5.3.7 revisits the Laser dataset and discusses how DendSer with BAR results a more interpretable heatmap of the Euclidean distance matrix for the rows in the Laser dataset than the heatmap resulting from DendSer with ARc and also the heatmaps resulting from some of the other seriation algorithms listed in Figure 2.5. Sections 4.5 and 4.6 discuss other datasets where minimising the BAR cost function produces more informative seriation results than minimising the ARc cost function.

3.4.3 LeafSort

This section introduces a cost function which measures how well weights on the leaves in a dendrogram increase as one reads from left to right.

Consider a set of objects with weights and visualise a hierarchical clustering of the objects using a dendrogram. The leaves in the dendrogram represent the objects and so each leaf has a corresponding weight. Given this situation, the following dendrogram seriation algorithm (described in Degerman 1982, Gale et al. 1984, Eisen et al. 1998 and Tien et al. 2008) examines each node N in the dendrogram and proceeds as follows:

1. Compute the mean weight of the leaves for the left and right sub-nodes of N. Denote these weights by $\bar{w}_L$ and $\bar{w}_R$ respectively.

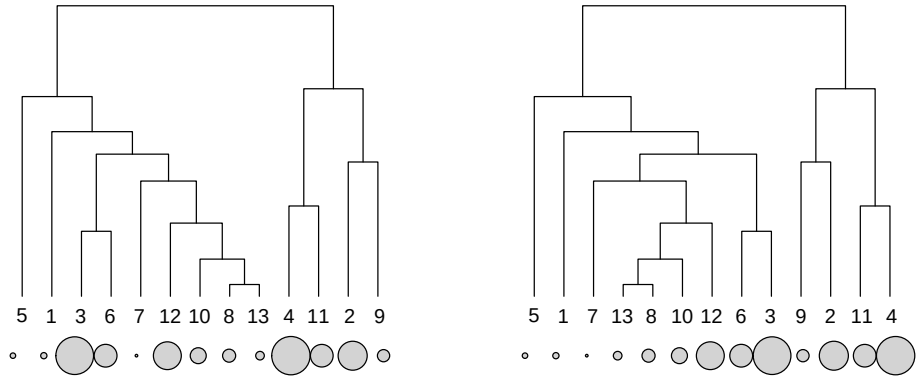
2. If $\bar{w}_R < \bar{w}_L$ then N is translated, otherwise N remains unchanged.

The end result of this "leaf sorting" algorithm is a rearrangement of the dendrogram, where the leaf weights generally increase as one reads from left to right.

Figure 3.4 illustrates a simple application of the leaf sorting algorithm. The dendrogram in Figure 3.4.(a) visualises a hierarchical clustering of thirteen randomly generated data objects. Each of these objects has a weight, which is represented by a circle beneath the leaves of the dendrogram. The leaf weights in Figure 3.4.(a) are in no particular order. However, through a series of node translations, the leaf weights can be ordered so that they are generally increasing as one reads from left to right, as shown in the dendrogram in Figure 3.4.(b).

The leaf sorting algorithm has been proposed by a number of authors, who differ in their method of weighting the leaves. Degerman (1982) computed a hierarchical clustering of eight body parts and weighted the body parts according to their anatomical position. He then rearranged the dendrogram so that the permutation of the leaves (i.e. body parts) were ordered according to their anatomical position. Eisen et al. (1998) weighted genes using some value, for example their mean gene expression level. They then computed a hierarchical clustering of the genes and used the leaf sorting algorithm to rearrange the leaves of the dendrogram so that leaves (i.e. genes) with smaller weights were placed earlier in the permutation.

Gale et al. (1984) generated an "external" permutation of a set of objects using an unspecified seriation algorithm aimed at optimising anti-Robinson form. They weighted each object by their position in the external permutation, i.e. the object in the ith position has a weight of i. They then computed



(a) Initial dendrogram (b) Increasing leaf weights

Figure 3.4: The dendrogram in (a) represents a hierarchical clustering of thirteen random data objects. Each of these objects has a corresponding weight, which is represented by a circle beneath the leaves of the dendrogram. The leaf weights in (a) are in no particular order. The dendrogram in (b) is a rearranged version of (a), which now has the leaf weights generally increasing from left to right.

a hierarchical clustering of the same set of objects and rearranged the dendrogram so that the resulting permutation of the leaves was as close as possible to the external permutation. Tien et al. (2008) used the same method as Gale et al. (1984) where the external permutation was generated using either the R2E seriation algorithm (Chen 2002) or the results of a one-dimensional self-organising map (Kohonen 1984).

Applying DendSer with the following cost function and the node operation T_0 performs the leaf sorting algorithm:

Definition 3.6. *Consider a set of n objects with corresponding weights w_i, for $1 \leqslant i \leqslant n$. For a permutation π of the objects, the LeafSort cost function is defined as:*

$$LS(\pi) = -\sum_{i=1}^{n} i w_{\pi(i)}. \tag{3.25}$$

To see why DendSer with the LS cost function and the node operation T_0 performs the leaf sorting algorithm, consider, for example, the node N_4 in the dendrogram in Figure 3.5. DendSer with LS and T_0 translates N_4 if

$$-(w_4 + 2w_5 + 3w_1 + 4w_2 + 5w_3) < -(w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5). \tag{3.26}$$

This condition simplifies to

$$\frac{w_1 + w_2 + w_3}{3} > \frac{w_4 + w_5}{2}. \tag{3.27}$$

Therefore, DendSer with LS and T_0 translates N_4 if the mean weight of the leaves in N_2 is less than the mean weight of the leaves in N_3. This is true for any number of leaves in N_2 and N_3.

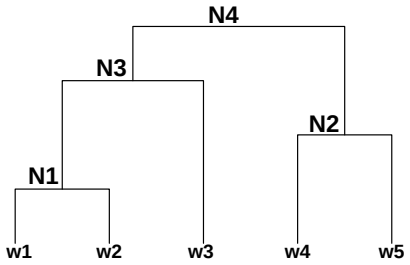


Figure 3.5: Dendrogram visualising a hierarchical clustering of five random data objects. The values w_i, for $1 \leqslant i \leqslant 5$, represent the leaf weights.

The LS cost function is $O(n)$ and so is fast to compute. It is clear that if one considers all $n!$ permutations, then the permutation minimising the LS cost function is the same permutation obtained by sorting the objects by their weight. However, this permutation may not be one of the 2^{n-1} permutations permitted by a dendrogram. Minimising the LS cost function over the 2^{n-1} permutations permitted by a dendrogram combines the benefits of clustering with the benefits of sorting.

The following explains why DendSer with LS and T_0 converges in one iteration. Let N be a node with sub-nodes N_a and N_b. During the first iteration of DendSer with LS and T_0, if the mean weight of the leaves in N_a is less than the mean weight of the leaves in N_b, then DendSer places N_a to the left of N_b. The mean weight of the leaves in N_a and N_b is unaffected by any node translations that DendSer may perform elsewhere in the dendrogram during the first iteration. This means that if DendSer examines the node N during a second iteration, then DendSer would still keep N_a on the left of N_b and so DendSer would make no changes during a second iteration. Therefore, DendSer with LS and T_0 only requires one iteration to converge.

According to its documentation, the `reorder.dendrogram` function in the `R` package `stats` (R Development Core Team 2010) appears to be equivalent to using DendSer with LS and T_0. However, the following shows that this is not the case.

Consider again the dendrogram in Figure 3.5. The `reorder.dendrogram` function, when used with `agglo.fun=mean`, translates N_4 if the weight of N_2 is less than the weight of N_3. However, `reorder.dendrogram` computes the weight of a node using the previously calculated weights of the sub-nodes. For example, the `reorder.dendrogram` function computes the weight of N_3 in

Figure 3.5 as

$$\text{Weight of } N_3 = \frac{\text{Weight of } N_1 + w_3}{2}, \tag{3.28}$$

where the weight of N_1 is $\frac{w_1 + w_2}{2}$. This means that the weight of N_3 simplifies to:

$$\text{Weight of } N_3 = \frac{w_1 + w_2 + 2w_3}{4}. \tag{3.29}$$

These calculations show that `reorder.dendrogram` may not weight a node N by the mean weight of the leaves in N. Therefore, the `reorder.dendrogram` function is not equivalent to DendSer with LS and T_0, and does not accurately perform the leaf sorting algorithm described above.

3.4.4 EdgeDist

Applying DendSer with the following cost function and the R_1 node operation performs the algorithm described in Gruvaeus and Wainer (1972) (referred to here as the GW algorithm):

Definition 3.7. *Consider a set of n objects, where $d_{i,j}$ is the dissimilarity between objects i and j. Let N be a node in a dendrogram and let $\pi(k), \ldots, \pi(k+m)$ be the ordered leaves in N. Then the EdgeDist cost function takes in a permutation π and a node N, and computes the path length of the leaves in N, i.e.*

$$ED(\pi, N) = \sum_{i=k}^{k+m-1} d_{\pi(i),\pi(i+1)}. \tag{3.30}$$

The following explains why DendSer with ED and R_1 performs the GW algorithm (see the description of the GW algorithm in Section 2.2.1). Consider the first dendrogram in Figure 3.6 with leaves A, B, C and D, where the ED value for the node N is $d_{A,B} + d_{B,C} + d_{C,D}$. Applying the node operation R_1 to N produces the remaining three dendrograms in Figure 3.6, where the circles over the branches indicate which nodes are reflected in order to produce each dendrogram. Computing the ED value for N in each of these three dendrograms still involves $d_{A,B}$ and $d_{C,D}$ but the middle term $d_{B,C}$ changes to $d_{B,D}$, $d_{A,C}$ and $d_{A,D}$ respectively. This means that choosing the permutation giving the lowest ED value for N is equivalent to choosing the permutation that places the most similar objects at the edges of the sub-nodes of N adjacently.

The EdgeDist criterion may be described as a "local" path length criterion because it computes the path length of a permutation π "local" to a node N. Note that the EdgeDist criterion makes sense for dendrogram seriation algorithms only, whereas the other criteria discussed in this section may be optimised using methods other than dendrogram seriation.

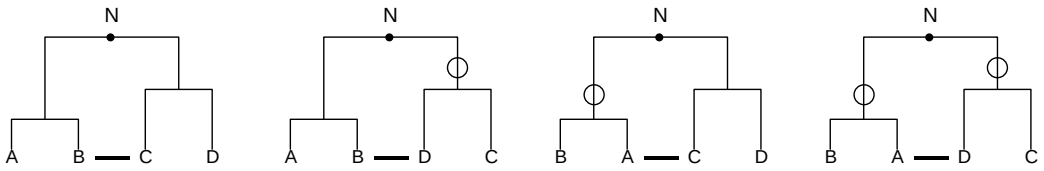


Figure 3.6: Applying the node operation R_1 to the node N in the first dendrogram produces the remaining three dendrograms, each with leaves A, B, C and D. The circles over the branches indicate which nodes are reflected in order to produce each dendrogram.

The following explains why DendSer with ED and R_1 converges in one iteration. During the first iteration, DendSer arranges a node N according to the minimum of the possible ED values for N. The ED values for N and the arrangement of N according to the minimum ED value are unaffected by other node reflections that DendSer may perform during the first iteration. This means that DendSer will make no changes during a second iteration and so DendSer with ED and R_1 requires only one iteration to converge.

Using DendSer with ED and R_{01} returns the same result as DendSer with ED and R_1 because for a node N in a dendrogram Δ, $R_1(N;\Delta) \subseteq R_{01}(N;\Delta)$ and the extra permutations returned by $R_{01}(N;\Delta)$ are equivalent, in the EdgeDist sense, to those returned by $R_1(N;\Delta)$. This is because the extra permutations returned by $R_{01}(N;\Delta)$ result from reflecting N in the initial arrangement of Δ and reflecting N in each of the dendrograms represented by $R_0(N_l;\Delta)$, $R_0(N_r;\Delta)$ and $R_0(N_l,N_r;\Delta)$ (i.e. the dendrograms represented by $R_1(N;\Delta)$). However, reflecting N does not affect the ED value of N.

Using DendSer with ED and either T_0, R_0, C_0, T_1 or T_{01} does not, in general, produce the same results as DendSer with ED and R_1. This is because, for a node N in a dendrogram Δ, $R_0(N;\Delta)$, $T_0(N;\Delta)$, $C_0(N;\Delta)$, $T_1(N;\Delta)$ and $T_{01}(N;\Delta)$ do not, in general, produce the same permutations as $R_1(N;\Delta)$. However, it is possible that using DendSer with ED and either T_0, R_0, C_0, T_1 or T_{01} may result in a permutation with a shorter path length than the permutation returned by using DendSer with ED and R_1.

3.5 A general framework for dendrogram seriation algorithms

DendSer uses the following general framework for dendrogram seriation:

1. Node selection: select each node N in a dendrogram Δ one-by-one in a

bottom-up manner, i.e. start with the first node formed by the hierarchical clustering process and end with the root node.

2. Node operation: rearrange N using a node operation T.

3. Seriation criterion: check if a new permutation returned by $T(N; \Delta)$ improves a criterion or cost function F. If so, then arrange N according to the new permutation, otherwise keep N in its original position.

4. Stopping criterion: stop when a full iteration fails to find any improvements to the cost function F, where one iteration is complete when all nodes in Δ have been examined.

Many of the dendrogram seriation algorithms listed in Figure 2.5 fit into the DendSer framework. For example, Section 3.4.3 described how using DendSer with the LS cost function and the T_0 node operation performs the "leaf sorting" algorithm described in Degerman (1982), Gale et al. (1984), Eisen et al. (1998) and Tien et al. (2008). Section 3.4.4 described how using DendSer with the ED criterion and the R_1 node operation performs the algorithm described in Gruvaeus and Wainer (1972). Both the leaf sorting and the Gruvaeus and Wainer (1972) algorithms stop after just one iteration.

DendSer can also perform the algorithm outlined in Wishart (1999). This algorithm selects each node in a bottom-up manner, uses the R_0 node operation, evaluates new permutations using a cost function similar to ARc and stops when a full iteration fails to find any improvement to the cost function.

Alon et al. (1999) and Wu et al. (2010) also described dendrogram seriation algorithms that fit into the DendSer framework. These algorithms rearrange nodes based on the distance to their "uncle" or "grand-uncle" node. For example, consider the dendrogram in Figure 3.7, where N_3 is the uncle of N_1, and N_6 is the grand-uncle of N_1:

- The algorithm described in Alon et al. (1999) rearranges the node N_5 if $d(C(N_1), C(N_3)) < d(C(N_2), C(N_3))$, where $C(N)$ denotes the centroid of a node N (the distance measure d is unspecified but is possibly Euclidean distance).

- The "Uncle algorithm" described in Wu et al. (2010) is similar to the algorithm in Alon et al. (1999) except for the distance measure d. The Uncle algorithm rearranges N_5 if $d(N_1, N_3) < d(N_2, N_3)$, where $d(N_a, N_b)$ is the mean dissimilarity between the leaves in the node N_a and the leaves in the node N_b, for whatever dissimilarity measure is used in the hierarchical clustering process.

- Wu et al. (2010) also described a "Grandpa algorithm", which rearranges N_5 if $d(N_1, N_6) < d(N_2, N_6)$, where the distance d is the same as for the Uncle algorithm.

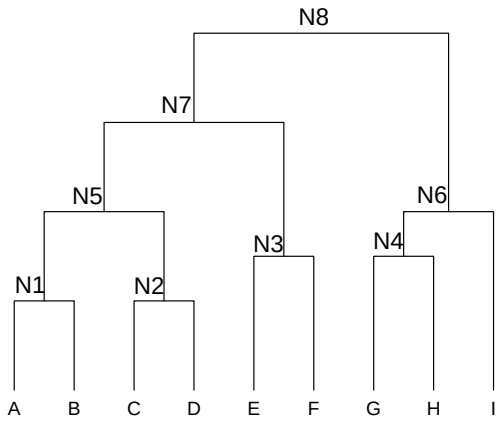


Figure 3.7: Dendrogram visualising a hierarchical clustering of nine random data objects. The node N_3 is the uncle of N_1, and N_6 is the grand-uncle of N_1.

Alon et al. (1999) and Wu et al. (2010) did not specify how their algorithms rearrange the nodes but worked examples suggest that the algorithm in Alon et al. (1999) and the Uncle algorithm use the R_0 node operation, while the Grandpa algorithm uses the T_0 node operation. These three algorithms also require just one iteration to converge.

The criteria developed in this thesis do not currently enable DendSer to perform the algorithms described in Alon et al. (1999) and Wu et al. (2010), however appropriate criteria may be developed in future.

The following three dendrogram seriation algorithms do not fit into the DendSer framework.

Morris et al. (2003) described a simulated annealing based dendrogram seriation algorithm. This algorithm randomly selects a node N in a dendrogram Δ and if $R_0(N; \Delta)$ produces a permutation that improves the ARc cost function, then N stays reflected. Otherwise, N stays reflected with probability inversely proportional to the increased cost of the permutation produced by $R_0(N; \Delta)$. The algorithm then repeats the process on another randomly selected node and continues until no improvement in the cost function is found for 20,000 node reflections.

Forina et al. (2007) described the following dendrogram seriation algorithm:

1. The algorithm works top-down, i.e. it begins with the root node, N_{n-1},

where n is the number of objects. A minimum and maximum value for a *depth* parameter is also specified.

2. The algorithm selects nodes within the specified *depth* of N_{n-1}. For example, if the nodes are labelled according to height, then for $depth = 2$ the selected nodes are $\{N_{n-1}, N_{n-2}\}$, for $depth = 3$ the selected nodes are $\{N_{n-1}, N_{n-2}, N_{n-3}\}$ and so on. Note that the selected nodes are not necessarily nested nodes.

3. The algorithm evaluates permutations corresponding to all possible combinations of applying the node operation T_0 to the selected nodes. The permutation with the shortest path length is kept and the dendrogram is updated according to this permutation.

4. The algorithm then increases the *depth* by one and repeats Steps 2 and 3.

5. When the maximum *depth* is reached, the algorithm repeats Steps 2-4 on the next lowest node in the hierarchy and continues until it reaches a specified lowest node.

6. After all required nodes have been examined, the algorithm either stops or repeats Steps 1-5 using R_0 instead of T_0.

Bar-Joseph et al. (2001) presented their Optimal Leaf Ordering (OLO) algorithm, which they described as being similar to dynamic programming. The difference between the OLO algorithm and all other algorithms described in this section is that the OLO algorithm is not a heuristic algorithm, i.e. the OLO algorithm is guaranteed to find the permutation from the dendrogram that minimises the path length criterion.

Table 3.1 provides an overview of the dendrogram seriation algorithms described in this section. The first column contains the references for each algorithm. The second and third columns contain the seriation criterion and the node operation (where relevant) for each algorithm respectively. The fourth column summarises the way in which each algorithm examines the nodes in a dendrogram and the fifth column summarises the stopping criterion for the algorithm. Finally, the sixth column indicates whether or not each algorithm fits into the DendSer framework.

Table 3.1: Overview of dendrogram seriation algorithms.

Dendrogram seriation algorithm	**Seriation criterion**	**Node operation**	**Node selection**	**Stopping criterion**	**Fits into DendSer framework**
Gruvaeus and Wainer (1972)	EdgeDist	R_1	Bottom-up one-by-one	1 iteration	✓
Degerman (1982)	LeafSort	T_0	Bottom-up one-by-one	1 iteration	✓
Gale et al. (1984)	LeafSort	T_0	Bottom-up one-by-one	1 iteration	✓
Eisen et al. (1998)	LeafSort	T_0	Bottom-up one-by-one	1 iteration	✓
Tien et al. (2008)	LeafSort	T_0	Bottom-up one-by-one	1 iteration	✓
Wishart (1999)	Anti-Robinson criterion	R_0	Bottom-up one-by-one	Repeat iterations until convergence	✓
Alon et al. (1999)	Distance to uncle node	$R_0{}^*$	Bottom-up one-by-one	1 iteration	✓
Wu et al. (2010) Uncle algorithm	Distance to uncle node	$R_0{}^*$	Bottom-up one-by-one	1 iteration	✓
Wu et al. (2010) Grandpa algorithm	Distance to grand-uncle node	$T_0{}^*$	Bottom-up one-by-one	1 iteration	✓
Morris et al. (2003)	Anti-Robinson criterion	R_0	Random	No improvement for 20,000 node reflections	✗
Forina et al. (2007)	Path length	T_0 & R_0	Top-down groups	When all specified nodes have been examined	✗
Bar-Joseph et al. (2001)	Path length	NA	NA	NA	✗

* Not specified by authors but worked examples suggest this choice of node operation.

3.6 Choice of node operation

This section investigates the best choice of node operation for use in DendSer when optimising each of the following criteria: path length, lazy path length, anti-Robinson form and banded anti-Robinson form. The node operations are assessed based on both the goodness of the seriation results returned by DendSer when used with each node operation and also the efficiency of DendSer when used with each node operation.

This section does not investigate the LS or ED cost functions because the LS cost function is designed to sort leaves by their weights and requires the T_0 node operation, and the ED cost function is designed for the Gruvaeus and Wainer (1972) algorithm and requires the R_1 node operation.

It is difficult to find intuitive arguments as to why a particular node operation is the most suitable when using DendSer to optimise the path length or lazy path length criteria. However, for anti-Robinson and banded anti-Robinson form, the following argues that the node operation T_0 is more suitable than the node operation R_0.

Consider the arbitrary dendrogram in Figure 3.8 and the outline of the

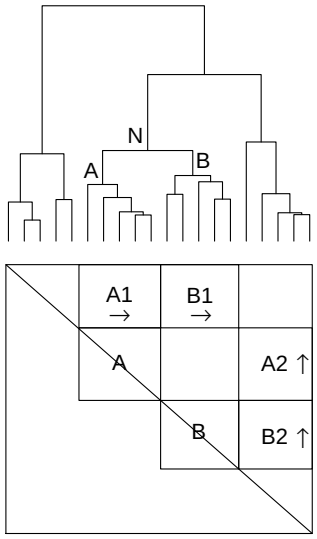


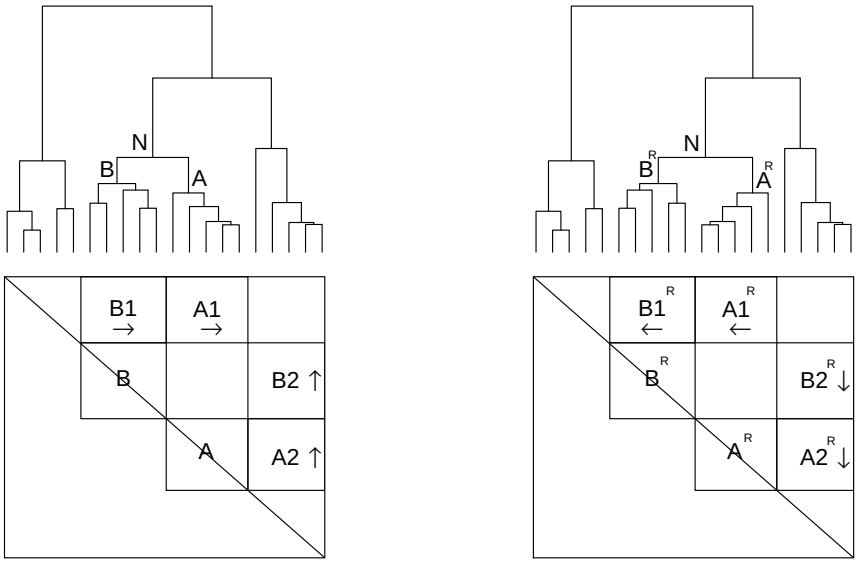
Figure 3.8: The nodes A and B in the dendrogram have been arranged so that areas of the dissimilarity matrix that contain dissimilarities involving the objects in A and B follow anti-Robinson form; these areas are A_1, A_2, B_1 and B_2. The arrow indicates that the values in the marked areas are increasing as one moves away from the main diagonal.

corresponding dissimilarity matrix below the dendrogram. The areas in the dissimilarity matrix marked A and B contain the dissimilarities between the

objects within the nodes A and B respectively. The areas A_1 and B_1 contain the dissimilarities between the objects to the left of node A and the objects in nodes A and B respectively. Similarly, the areas A_2 and B_2 contain the dissimilarities between the objects to the right of node B and the objects in nodes A and B respectively.

Assume that the nodes A and B are arranged so that the values in the areas A_1, B_1, A_2 and B_2 in the dissimilarity matrix follow anti-Robinson form. This is shown by the direction of the arrows in the areas A_1, A_2, B_1 and B_2, which indicate that the values are increasing as one moves away from the main diagonal. The values in B_1 may be lower than the values in A_1 or the values in A_2 may be lower than the values in B_2, which means that placing node B on the left of node A in the dendrogram would bring the dissimilarity matrix closer to following anti-Robinson form.

There are two ways of placing node B on the left of node A: translate the node N or reflect N. Figures 3.9.(a) and (b) illustrate the effect of translating



(a) Translate N (b) Reflect N

Figure 3.9: Figures (a) and (b) show the same dendrogram as in Figure 3.8. In (a), N is translated and so the values within A_1, A_2, B_1 and B_2 in the dissimilarity matrix still satisfy anti-Robinson form, as indicated by the direction of the arrows. In (b), N is reflected, which reverses the objects within A and B. This means that the values within A_1^R, A_2^R, B_1^R and B_2^R in the dissimilarity matrix are also reversed and so do not follow anti-Robinson form, as indicated by the direction of the arrows.

and reflecting N respectively on the dissimilarity matrix. Translating N preserves the order of the objects within the nodes A and B, and so, as indicated by the arrows in Figure 3.9.(a), the values in the areas A_1, B_1, A_2 and B_2 still

follow anti-Robinson form. However, reflecting N reverses the order of the objects within the nodes A and B, which means that the values in the areas A_1, B_1, A_2 and B_2 are also reversed, as indicated by the arrows in Figure 3.9.(b). Therefore, the values in these areas no longer follow anti-Robinson form.

In summary, translating N retains previous arrangements of nodes in a dendrogram, whereas reflecting N undoes previous arrangements. Therefore, it seems that the node operation T_0 is more suitable than the node operation R_0 when using DendSer to optimise anti-Robinson criteria. This suggests that T_0 is also more suitable than R_0 when using DendSer to optimise banded anti-Robinson criteria.

3.6.1 Measuring the efficiency of DendSer

DendSer generally requires more than one iteration to converge when minimising PL, LPL, ARc and BAR, regardless of the chosen node operation. Therefore, it is somewhat complicated to assess the efficiency of DendSer when used with different node operations.

For example, the previous section argued that T_0 is suitable when using DendSer to minimise the ARc cost function. This suggests that T_{01}, R_{01} and C_0 are also suitable node operations to use in DendSer when minimising the ARc cost function (because for a node N in a dendrogram Δ, $T_0(N;\Delta) \subseteq T_{01}(N;\Delta)$, $T_0(N;\Delta) \subseteq R_{01}(N;\Delta)$ and $T_0(N;\Delta) \subseteq C_0(N;\Delta)$). T_0 appears to give DendSer the least amount of work because it produces the fewest permutations per node for DendSer to evaluate. However, DendSer may converge faster with T_{01}, R_{01} or C_0 than with T_0.

The workload of DendSer is broken down as follows:

1. For each iteration required until convergence:
   (a) for each node N in a dendrogram Δ:
      i. calculate the cost function F for each of the permutations returned by $T(N;\Delta)$, where T is the node operation.
      ii. update Δ according to the permutation with the lowest cost.

This workload gives DendSer the following time complexity:

$$\#\text{ iterations} \times O(n) \times ((O(F) \times \#\text{ perms returned by } T) + O(n)). \quad (3.31)$$

For each node operation T, assessing the efficiency of DendSer when used with T requires counting the number of times DendSer computes F and the number of times DendSer updates the dendrogram.

Table 3.2 summarises the complexity of DendSer when used in conjunction with each of the cost functions discussed in Section 3.4.

Table 3.2: Time complexity of DendSer when used in conjunction with different cost functions.

Cost function F	Complexity of F	Complexity of DendSer + F
PL LPL	$O(n)$	# iterations $\times$ $O(n^2)$
ARc BAR	$O(n^2)$	# iterations $\times$ $O(n^3)$
LS ED	$O(n)$	$O(n^2)$

Note that DendSer usually converges in less than ten iterations when used in conjunction with either the PL, LPL, ARc or BAR cost function. Also note that the implementation of the BAR cost function used in this thesis is $O(n^2)$.

3.6.2 Simulation study

The following simulation study investigates which node operation is most suitable for use in DendSer when minimising PL, LPL, ARc and BAR:

1. For each of the Iris (Fisher 1936), Laser (see Section 3.4.2) and Sleep (Allison and Cicchetti 1976) datasets:
   (a) Create 100 samples, each with fifty randomly selected cases and standardised variables (all variables are used for the Laser and Sleep datasets, and all variables except the species variable are used for the Iris dataset).
2. For each sample:
   (a) Construct a hierarchical clustering of the cases using Euclidean distance and average linkage.
   (b) Seriate the dendrogram using DendSer with F = PL and each of R_0, T_0, R_1, T_1, R_{01}, T_{01} and C_0.
   (c) Record:
      i. The value of F for each of the seven permutations returned.
      ii. The number of times DendSer updates the dendrogram for each of the seven runs.

iii. The number of times DendSer computes F for each of the seven runs.

iv. The number of times DendSer with C_0 preferred the R_0 or T_0 option.

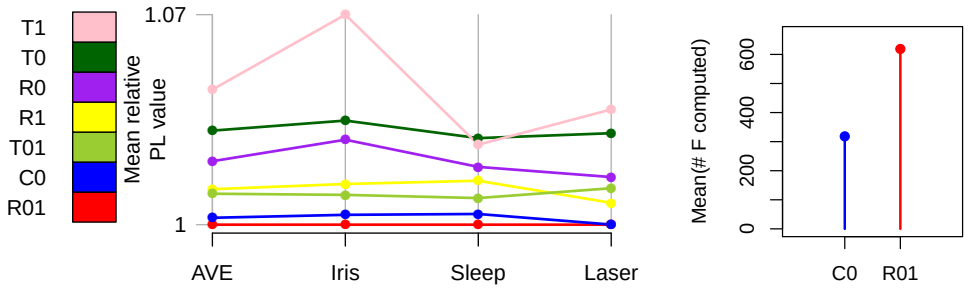
3. Repeat Steps 1 and 2 with F = LPL, ARc and BAR.

The following reports the results of the simulation study for each cost function. The number of times DendSer updated a dendrogram is not reported because the choice of node operation had no effect on this value. Note that for each cost function F, the mean number of times DendSer computed F is only reported for the node operations that resulted in DendSer producing permutations with low cost values.

Also, the following results correspond to average linkage, however the results are similar for single, complete and Ward's linkage.

Simulation results for the PL cost function

For each dataset, the axes in Figure 3.10.(a) show the mean of the PL values for each of the seven node operations relative to the PL values for R_{01}. The first axis in Figure 3.10.(a) shows the overall mean relative PL value for each of the node operations.



(a) (b)

Figure 3.10: For each dataset, the axes in Figure (a) show the mean of the PL values for each of the node operations relative to the PL values for R_{01}. The first axis shows the overall mean relative PL value for each of the node operations. The plot in (b) shows the mean number of times that DendSer computed the PL cost function when used with R_{01} and C_0. The legend orders the node operations according to the first axis in Figure (a).

The mean relative PL value for R_{01} is equal to one. If a node operation T has a mean relative PL value greater than one, then DendSer with PL

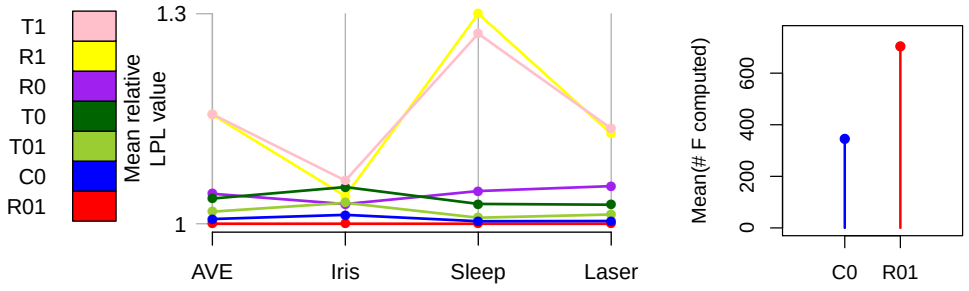
and T returned permutations having, on average, higher PL values than the corresponding permutations returned by DendSer with PL and R_{01}. Similarly, if a node operation T has a mean relative PL value less than one, then DendSer with PL and T returned permutations having, on average, lower PL values than the corresponding permutations returned by DendSer with PL and R_{01}.

Figure 3.10.(b) shows the overall mean number of times DendSer computed the PL cost function when used with C_0 and R_{01} (these figures are consistent across all three datasets). The legend in Figure 3.10 orders the node operations according to the first axis in the parallel coordinates plot in Figure 3.10.(a).

Figure 3.10.(a) shows that DendSer with R_{01} produced permutations with the lowest PL values. However, DendSer with C_0 performed almost as well as DendSer with R_{01} and Figure 3.10.(b) shows that DendSer with C_0 calculated the PL cost function far less then DendSer with R_{01}. Therefore, it may be more efficient to use DendSer with C_0 when seriating large numbers of objects using the PL cost function.

Simulation results for the LPL cost function

Figure 3.11 shows the simulation results for the LPL cost function, where Figures 3.11.(a) and (b) are constructed as in Figure 3.10.



(a) (b)

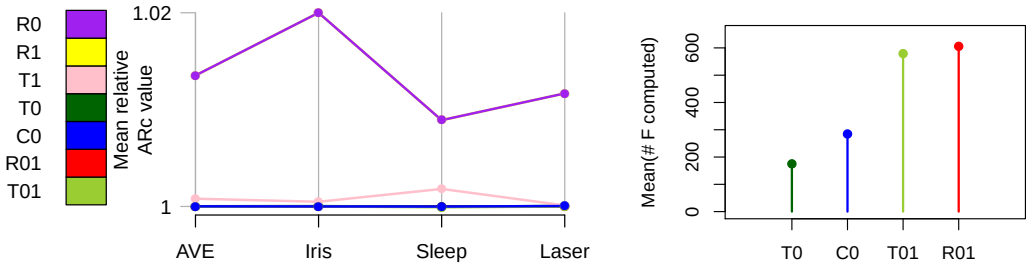
Figure 3.11: For each dataset, the axes in Figure (a) show the mean of the LPL values for each of the node operations relative to the LPL values for R_{01}. The first axis shows the overall mean relative LPL value for each of the node operations. The plot in (b) shows the mean number of times that DendSer computed the LPL cost function when used with R_{01} and C_0. The legend orders the node operations according to the first axis in Figure (a).

Figure 3.11.(a) shows that DendSer with R_{01} and C_0 produced permutations with the lowest LPL values for all datasets. Although DendSer with R_{01} performed slightly better than DendSer with C_0, Figure 3.11.(b) shows that DendSer with C_0 computed the LPL cost function far less than DendSer with

R_{01}. Therefore, it is more efficient to use DendSer with C_0 when seriating large numbers of objects using the LPL cost function.

Simulation results for the ARc cost function

Figure 3.12 shows the simulation results for the ARc cost function, where Figures 3.12.(a) and (b) are constructed as in Figure 3.10. Note that in Figure 3.12.(a), the yellow line for R_1 is plotted beneath the purple line for R_0, and the lines for T_{01}, R_{01} and T_0 are plotted beneath the blue line for C_0.



(a) (b)

Figure 3.12: For each dataset, the axes in Figure (a) show the mean of the ARc values for each of the node operations relative to the ARc values for R_{01}. The first axis shows the overall mean relative ARc value for each of the node operations. The plot in (b) shows the mean number of times that DendSer computed the ARc cost function when used with T_0, C_0, T_{01} and R_{01}. The legend orders the node operations according to the first axis in Figure (a).

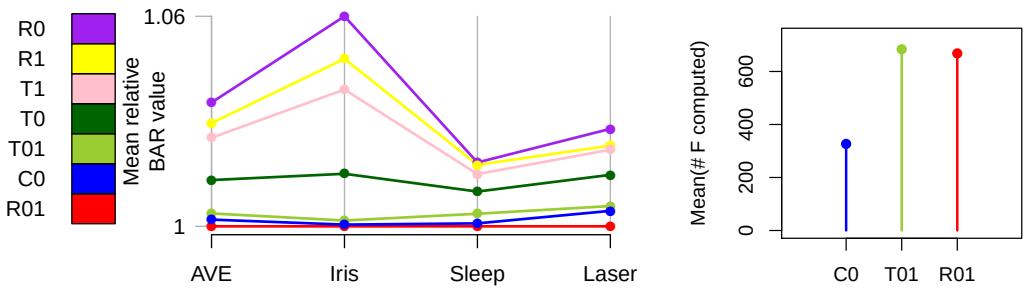
Figure 3.12.(a) shows that DendSer with either T_{01}, R_{01}, C_0 or T_0 performed equally well for all datasets. However, Figure 3.12.(b) shows that DendSer with T_0 computed the ARc cost function less often than DendSer with either R_{01}, T_{01} or C_0. Therefore, T_0 is the best choice of node operation for use in DendSer when minimising the ARc cost function.

Note that DendSer with T_0 produced permutations with lower ARc values than DendSer with R_0. This agrees with the argument in the beginning of Section 3.6 that T_0 is more suitable than R_0 when using DendSer to optimise anti-Robinson form.

Simulation results for the BAR cost function

Figure 3.13 shows the simulation results for the BAR cost function, where Figures 3.13.(a) and (b) are constructed as in Figure 3.10.

Figure 3.13.(a) shows that DendSer with either R_{01}, C_0 or T_{01} performed almost equally well for the Iris and Sleep datasets, while DendSer with R_{01}



(a) (b)

Figure 3.13: For each dataset, the axes in Figure (a) show the mean of the BAR values for each of the node operations relative to the BAR values for R_{01}. The first axis shows the overall mean relative BAR value for each of the node operations. The plot in (b) shows the mean number of times that DendSer computed the BAR cost function when used with C_0, T_{01} and R_{01}. The legend orders the node operations according to the first axis in Figure (a).

performed better for the Laser dataset. However, Figure 3.13.(b) shows that DendSer with C_0 calculated the BAR cost function far less than DendSer with either R_{01} and T_{01}. Therefore, it may be more efficient to use DendSer with C_0 when seriating large numbers of objects using the BAR cost function.

These results somewhat agree with the argument in the beginning of Section 3.6 that T_0 is more suitable than R_0 when using DendSer to optimise banded anti-Robinson criteria. This is because DendSer with BAR and C_0 preferred the T_0 option over the R_0 option in approximately 84% of cases.

3.6.3 Discussion

Based on the results of the simulation study, Table 3.3 recommends suitable node operations to use in DendSer when minimising each of the examined cost functions.

Table 3.3: Recommended node operations to use in DendSer when minimising different cost functions.

Cost function	Node operation
ARc	T_0
BAR	R_{01} & C_0
PL	R_{01} & C_0
LPL	R_{01} & C_0

Of the seven node operations examined in this study, DendSer with R_{01}

produced permutations with the lowest cost values for PL, LPL and BAR. DendSer with R_{01} also performed equally well as DendSer with T_0 for the ARc cost function (see Figure 3.12.(a)). This versatility of R_{01} suggests that it is a suitable node operation for use in DendSer when minimising other cost functions that were not examined in this study. DendSer with C_0 also performed consistently well for all four cost functions and so the node operation C_0 is also quite flexible and is a more efficient, although in some cases slightly less optimal, alternative to R_{01}.

Note the both of the node operations, R_{01} and C_0, are newly defined in this thesis and have not been implemented in other dendrogram seriation algorithms.

The simulation results for the ARc cost function shown in Figure 3.12.(a) suggest that some currently available dendrogram seriation algorithms are not using the most suitable node operation. Wishart (1999) and Morris et al. (2005) presented dendrogram seriation algorithms for optimising an anti-Robinson function with both of these algorithms using R_0 as the node operation. However, the simulation results in Figure 3.12.(a) suggest that both of these algorithms could be improved by using T_0 instead of R_0 because DendSer with T_0 produced permutations with better ARc values than DendSer with R_0.

3.7 Summary

This chapter presented a new dendrogram seriation algorithm called DendSer.

One of the features of DendSer is the choice of how to rearrange the nodes in a dendrogram, which is an important but generally ignored aspect of dendrogram seriation. Section 3.3 developed notation and terminology for describing dendrogram seriation algorithms and defined several node operations, which rearrange or operate on nodes in different ways.

DendSer is a flexible seriation algorithm, allowing the user to choose from a variety of seriation criteria including path length and anti-Robinson form. The choice of criteria also includes two new seriation criteria called lazy path length and banded anti-Robinson form, which are described in Sections 3.4.1 and 3.4.2.

Section 3.5 discussed how DendSer provides a general framework for dendrogram seriation with several algorithms fitting into this framework (Gruvaeus and Wainer 1972, Degerman 1982, Gale et al. 1984, Eisen et al. 1998, Tien et al. 2008, Alon et al. 1999, Wishart 1999 and Wu et al. 2010). This section then gave a brief account of three other dendrogram seriation algo-

rithms (Bar-Joseph et al. 2001, Morris et al. 2003 and Forina et al. 2007) that do not fit into the DendSer framework. Table 3.1 provides an overview of all algorithms discussed in Section 3.5.

Section 3.6 investigated the most suitable choice of node operation for use in DendSer when minimising each of the following cost functions: PL, LPL, ARc and BAR. For each cost function, the simulation study in Section 3.6.2 examined both the goodness of the permutations returned by DendSer when used with each of the node operations and the amount of work each node operation created for DendSer. The findings of Section 3.6.2 are summarised in Table 3.3.

Chapter 4

Applications of DendSer

4.1 Introduction

This chapter presents visualisation applications of the DendSer algorithm.

Section 4.2 describes an analysis of Pottery data (Tubb et al. 1980). For this data, seriating a heatmap, an array of glyphs and a parallel coordinates plot using DendSer helps to informally assess the goodness of a hierarchical clustering solution and answer various questions about the resulting clusters.

The example in Section 4.3 shows how DendSer with the LeafSort criterion may be used to order genes in a heatmap according to the algorithm described in Eisen et al. (1998) (see Section 3.4.3).

Section 4.4 uses the lazy path length criterion to place interesting panels in a prominent position in a scatterplot matrix, thereby making it easier to extract interesting information from data.

Section 4.5 explains why optimising an anti-Robinson criterion may produce uninformative results when seriating data that follow a specific pattern called a "circumplex" pattern, whereas optimising banded anti-Robinson form or path length is more suitable for seriating this type of data.

Section 4.6 explores a larger dataset that also follows a circumplex pattern. In this case, DendSer with the banded anti-Robinson criterion produces more informative seriation results than DendSer with the path length criterion.

Note that in all of the following examples, DendSer is used with the node operation R_{01} when minimising the PL, LPL and BAR cost functions, and the node operation T_0 when minimising the ARc and LS cost functions. See Section 3.6 for justification of these choices of node operations.

The chapter ends with a brief summary and discussion.

4.2 Pottery data

Tubb et al. (1980) analysed data on the chemical composition of Romano-British pottery found at different kiln sites in Britain. The dataset used in this example contains nine chemical measurements of 45 pieces of pottery. The five kiln sites at which the pottery pieces were found are Llanedeyrn and Caldicot (both in Wales), Islands Thorns and Ashley Rails (both in Hampshire) and Gloucester.

The first step in analysing this data is to see if the pots cluster into distinct groups based on their chemical composition. Figure 4.1 shows a dendrogram visualising a hierarchical clustering of the pots using Euclidean distance and average linkage. This dendrogram suggests that there are three distinct clusters of pots.

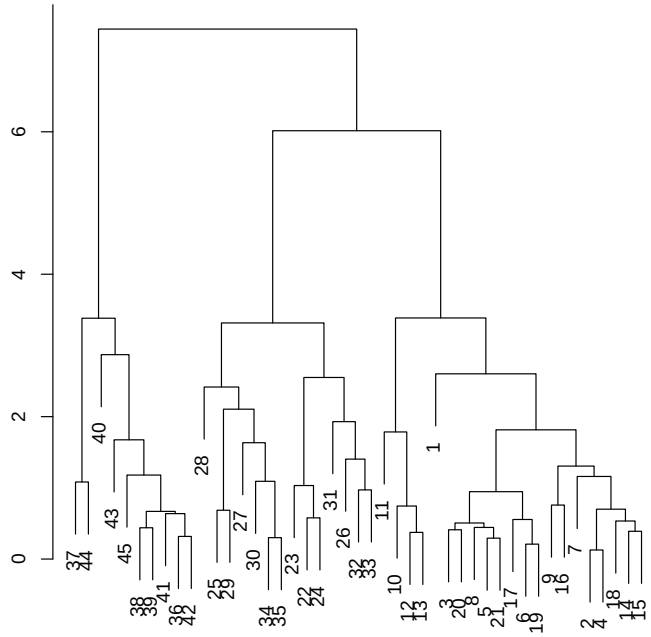
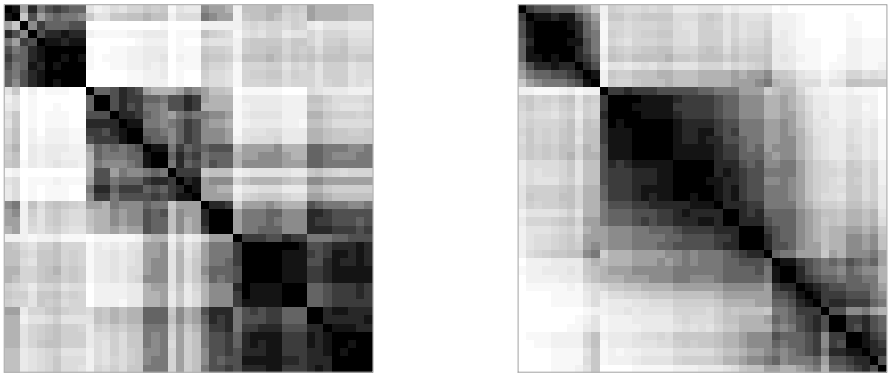


Figure 4.1: Dendrogram visualising a hierarchical clustering (average linkage) of the pots in the Pottery dataset.

The heatmap of the Euclidean distance matrix for the Pottery data, shown in Figure 4.2.(a), allows an informal assessment of the hierarchical clustering solution. The rows and columns of the heatmap in Figure 4.2.(a) are ordered according to the permutation of the 45 pots from the dendrogram in Figure 4.1. In Figure 4.2.(b), the row and column ordering is obtained from the same dendrogram except the dendrogram has been rearranged using DendSer with BAR. Using DendSer to optimise banded anti-Robinson form in the Euclidean



(a) Initial ordering from dendrogram (b) DendSer + BAR

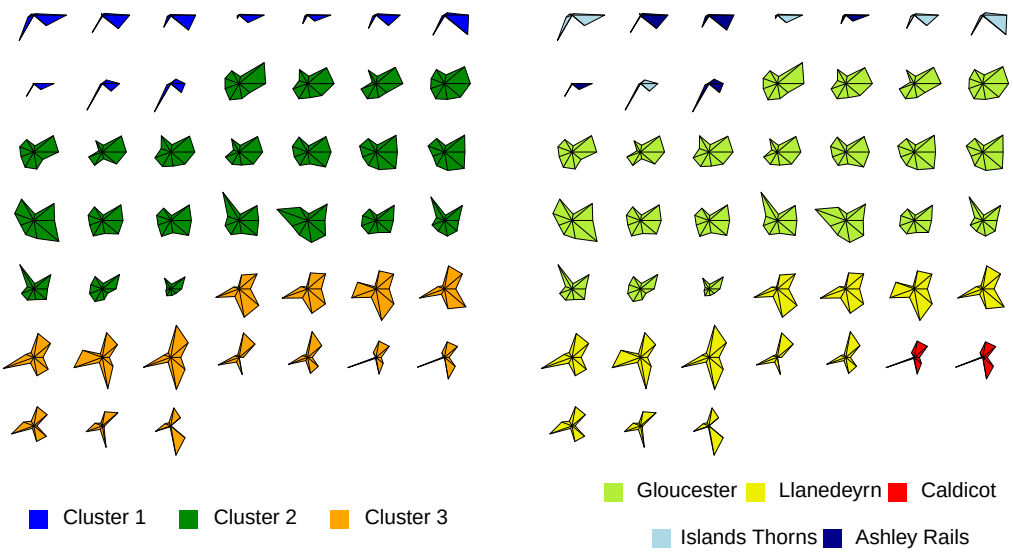
Figure 4.2: Heatmaps of the Euclidean distance matrix for the pots in the Pottery dataset. The heatmap in (a) is ordered according to the permutation from the dendrogram in Figure 4.1. The heatmap in (b) is ordered according to the permutation of the pots returned by using DendSer with BAR. The colour scale black to white represents low to high Euclidean distances.

distance matrix results in a more contiguous display of colour in the heatmap, making it easier to determine clustering patterns in the data. Based on both the dendrogram in Figure 4.1 and the heatmap in Figure 4.2.(b), the pots are divided into three clusters.

An array of glyphs is another useful visualisation for analysing clustering results. Figure 4.3.(a) shows an array of star glyphs (see, for example, Ward 2002), one star for each pot, with the stars coloured according to the three cluster solution. The stars are ordered according to the permutation of the pots returned by DendSer with BAR, as in Figure 4.2.(b). The array shows that pots in the same cluster have similarly shaped stars and pots from different clusters have differently shaped stars.

The array of stars may also be used to examine how the three clusters relate to the kiln sites at which the pots were found. In the array in Figure 4.3.(b), the stars are coloured according to their corresponding kiln sites and ordered as in Figure 4.3.(a). It is clear that cluster 1 contains the pots found at Islands Thorns and Ashley Rails, cluster 2 contains the pots found at Gloucester and cluster 3 contains the pots found at Llanedeyrn and Caldicot. Therefore, the clusters correspond exactly to Hampshire, Gloucester and Wales, which are the three regions in which the pots were found.

This example now explores the chemical composition of the three clusters using the parallel coordinates plots (PCPs) in Figure 4.4. The lines in Figure 4.4.(a) are coloured according to the three cluster solution (or equivalently, the



(a) 3-cluster solution (b) Kiln sites

Figure 4.3: The stars in the array in (a) are coloured according to the three cluster solution from the hierarchical clustering of the pots in the Pottery dataset. The stars in the array in (b) are coloured according to the Kiln sites at which the pots were found. The stars in both arrays are ordered according to the permutation of the pots returned by using DendSer with the BAR cost function.

regions) and the variables are ordered arbitrarily. The lines in this PCP are quite "zig-zaggy" and the panels show poor separation of the three clusters.

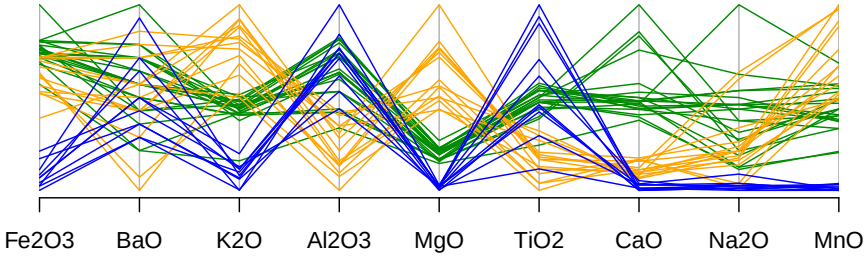
For exploring the cluster differences, a desirable permutation of the variables is one that results in the PCP panels showing good separation of the three clusters. The following defines a merit measure, which rewards such panels. Let $\bar{x}_{i,k}$ be the mean value of variable i for the pots in cluster k, where variable i is scaled to lie in the unit interval $[0, 1]$ (note that this is the usual choice of scaling for a PCP). Let $d_{i,j}(k_1, k_2)$ be the Euclidean distance between the cluster centroids on the variables i and j for the clusters k_1 and k_2, i.e.

$$d_{i,j}(k_1, k_2) = \sqrt{(\bar{x}_{i,k_1} - \bar{x}_{i,k_2})^2 + (\bar{x}_{j,k_1} - \bar{x}_{j,k_2})^2}. \tag{4.1}$$

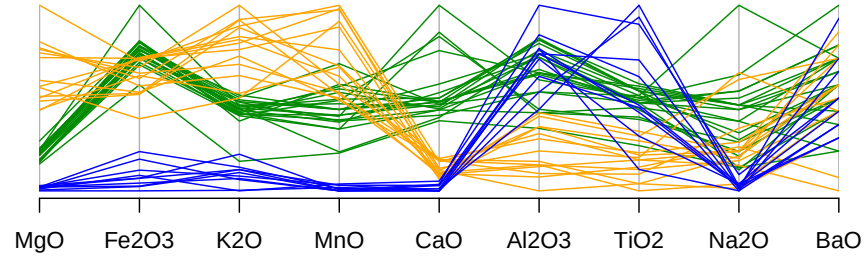
Then, for two variables i and j, the merit measure, $m_{i,j}$, is defined to be the following:

$$m_{i,j} = d_{i,j}(1, 2) + d_{i,j}(1, 3) + d_{i,j}(2, 3). \tag{4.2}$$

The larger the $m_{i,j}$ value, the better the separation of the three clusters in the PCP panel formed by the variables i and j. The merit values $m_{i,j}$ form a merit matrix $M = [m_{i,j}]$, for $1 \leqslant i, j, \leqslant 9$. The merit matrix M is converted



(a) Arbitrary ordering



(b) DendSer + LPL

Figure 4.4: The variables in the PCP in (a) are arbitrarily ordered. The variables in the PCP in (b) are ordered according to the permutation returned by using DendSer with the LPL cost function. The lines are coloured according to the three cluster solution from the hierarchical clustering of the Pottery data.

into a loss matrix $L = [l_{i,j}]$ using the transformation $L = \max(M) - M$. Therefore, the smaller the $l_{i,j}$ value, the better the separation of the three clusters in the PCP panel formed by the variables i and j.

The variables are now hierarchically clustered using the loss matrix L and average linkage. DendSer with the LPL cost function returns a permutation of the variables with a small sum of the loss values between adjacent variables, with the loss values generally increasing along the permutation. This permutation is used to construct the PCP in Figure 4.4.(b).

The PCP in Figure 4.4.(b) now contains panels that show better separation of the clusters, which makes it easier to extract information about the differences between the chemical composition of the three clusters of pots. Minimising the LPL cost function also positions panels showing the most separation of the three clusters at the beginning of the PCP and panels showing the least separation of the three clusters at the end. Given that people generally read from left to right, it follows that people are also likely to examine a PCP from left to right (or top to bottom, depending on the orientation of the PCP). Therefore, placing the most "interesting" panels at the beginning of the PCP allows the analyst to immediately see features of interest in the data.

Figure 4.4.(b) clearly shows that the pots from Hampshire (blue lines) con-

tain the lowest levels of magnesium (MgO), iron (Fe_2O_3), potassium (K_2O), manganese (MnO) and calcium (CaO). The pots from Wales (orange lines) contain the highest levels of magnesium (MgO), potassium (K_2O), and manganese (MnO), and the lowest levels of aluminium (Al_2O_3). The pots from Gloucester (green lines) contain the highest levels of iron (Fe_2O_3) and calcium (CaO), and medium levels of magnesium (MgO), potassium (K_2O) and manganese (MnO). The arbitrarily ordered PCP in Figure 4.4.(a) requires much closer inspection in order to extract the above information.

4.3 Cancer data

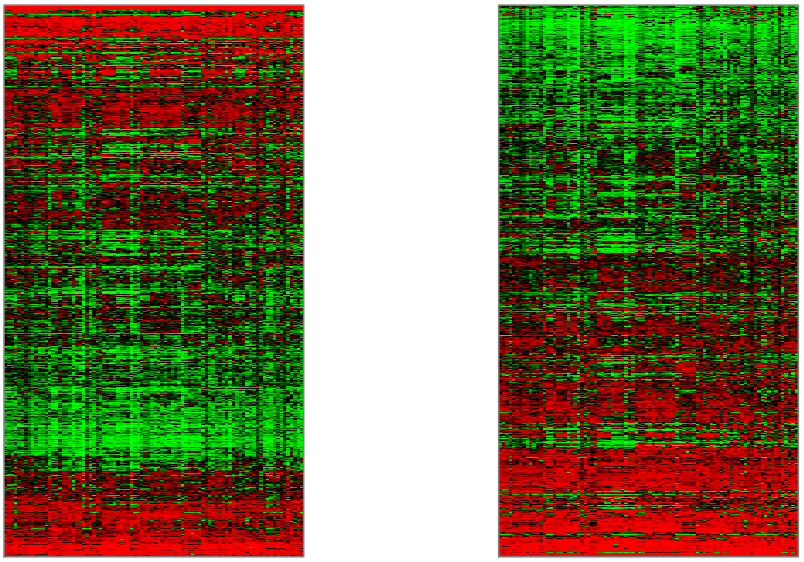
In the following example, DendSer with the LeafSort criterion demonstrates the method described in Eisen et al. (1998) (see Section 3.4.3).

Khan et al. (2001) described gene expression data containing 2308 genes and 88 tumours, where the 88 tumours fall into five categories of cancer. Figure 4.5.(a) shows the heatmap of the gene expressions, where the genes (rows) are ordered according to the permutation obtained from a hierarchical clustering of the genes using average linkage and Euclidean distance. The tumours (columns) are ordered according to their cancer type. The colour scale green to red represents low to high expression.

The heatmap in Figure 4.5.(a) shows large bands of red and green indicating groups of genes with similar expression levels. However, the initial ordering from the dendrogram places two groups of genes with high expression (red areas) at opposite ends of the heatmap. In Figure 4.5.(b), the gene ordering is obtained from the same hierarchical clustering as in Figure 4.5.(a). The difference is that, in this case, the dendrogram is rearranged using DendSer with the LS cost function, where the leaf (i.e. gene) weight is the mean expression level for that gene. The resulting permutation places the groups of high expression genes adjacently in the heatmap in Figure 4.5.(b), which now shows a general trend of low to high expression in the genes.

Although DendSer with LS appears to work well for this example, a more sensible general method for ordering genes may be to use DendSer with either the PL, BAR or ARc cost function. This is because minimising PL, BAR, or ARc places similar genes close together in the heatmap. This is not a guaranteed result of using DendSer with LS because two genes having the same mean expression level does not imply that the genes are similar.

However, gene expression datasets typically contain thousands of genes meaning that DendSer with PL, BAR or ARc may be too time consuming.



(a) Initial ordering from dendrogram (b) DendSer + LS

Figure 4.5: Heatmaps of the gene expression dataset described in Khan et al. (2001). The rows (i.e. genes) of the heatmap in (a) are ordered according to the permutation obtained from a hierarchical clustering of the genes using Euclidean distance and average linkage. The rows (i.e. genes) of the heatmap in (b) are ordered according to DendSer with the LS cost function.

For such large datasets, it may be more efficient to use DendSer with LS (i.e. the method from Eisen et al. 1998) because it runs in a faster time than DendSer with either PL, BAR or ARc (see Table 3.2).

4.4 Sleep data

Based on Bertin's (1983) concept of "diagonalisation", Hurley (2004) proposed seriating variables in scatterplot matrices so that interesting panels were placed close to the main diagonal. The logic behind this is that interesting panels are placed in a prominent position, making it easier for the analyst to observe features of interest in the data.

Seriation with either the PL, BAR or ARc cost function is one way of positioning interesting panels close to the main diagonal. However, this example shows that seriating variables using DendSer with the lazy path length criterion makes interesting panels even more prominent by not only positioning interesting panels close to the main diagonal but also positioning the *most*

interesting panels in the north-west of the scatterplot matrix.

Outliers are one of the first features to look for when analysing data and scatterplot matrices are a convenient visualisation tool for revealing bivariate outliers in data. Figure 4.6 shows a scatterplot matrix of the Sleep data (Allison and Cicchetti 1976), which contains ten measurements on 62 mammals. The bars in the lower triangle of the scatterplot matrix represent the Outlying scagnostic index (Wilkinson et al. 2005) for each of the corresponding scatterplots in the upper triangle. The Outlying scagnostic index is a merit measure with values lying in the unit interval $[0, 1]$. The dashed lines in the panels in the lower triangle of the scatterplot matrix represent the value 0.5.

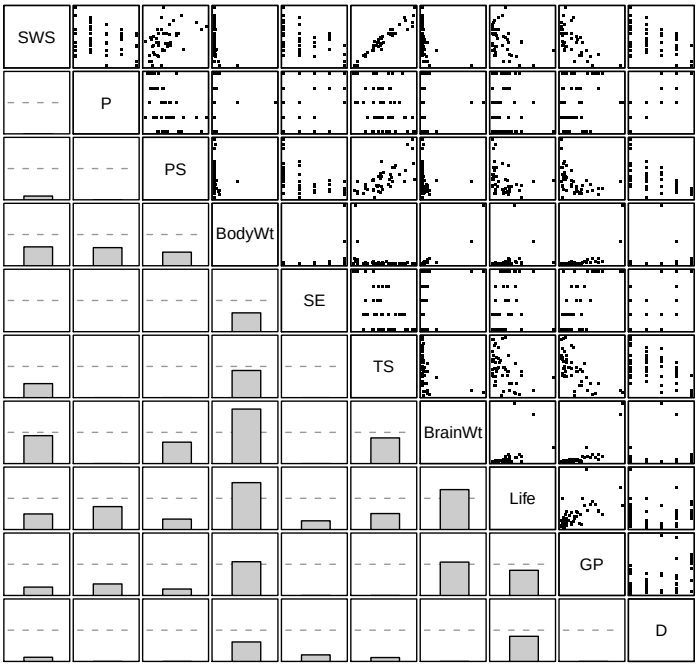


Figure 4.6: Scatterplot matrix of the Sleep dataset, constructed using an arbitrary ordering of the variables. The bars in the lower triangle represent the Outlying scagnostic value for each of the corresponding scatterplots and the dashed lines in the panels in the lower triangle represent the value 0.5.

The Outlying scagnostic values form a merit matrix $M = [m_{i,j}]$, for $1 \leqslant i, j \leqslant 10$, where $m_{i,j}$ is the Outlying scagnostic value for the scatterplot formed by variables i and j. The larger the value of $m_{i,j}$, the stronger the presence of outliers in the scatterplot formed by variables i and j. The merit matrix M is converted into a loss matrix L using the transformation $L = 1 - M$.

After computing a hierarchical clustering of L using average linkage, the corresponding dendrogram is rearranged using DendSer with the LPL cost function. The scatterplot in Figure 4.7 is constructed using the resulting per-

mutation of variables.

Minimising the LPL cost function positions panels with the most extreme outliers (panels with an Outlying scagnostic value of 0.5 or more) in the north-west of the scatterplot matrix, making it easier to see that the variables BrainWt and BodyWt contain two extreme outliers (Asian and African elephant) and the variable Life contains one, less extreme, outlier (Human). These outliers are not as clearly observed from the arbitrarily ordered scatterplot matrix in Figure 4.6.

Figure 4.7: Scatterplot matrix of the Sleep data, with variables ordered according to DendSer with LPL. The bars in the lower triangle represent the Outlying scagnostic value for each of the corresponding scatterplots and the dashed lines in the panels in the lower triangle represent the value 0.5.

Note that the Outlying scagnostic index is an example of a visualisation based measure because, for two variables i and j, it measures the "interestingness" of the scatterplot produced by variables i and j. The above application could also be done for other scagnostic indexes (Wilkinson et al. 2005).

4.5 Morse code data

Rothkopf (1957) described an experiment, where inexperienced subjects listened to pairs of morse codes and then decided whether a pair of codes were identical. The data used in this example contain the results for the ten single

digits, where the ij^{th} entry in the data is the percentage of subjects who said codes i and j were identical after hearing code i first and then code j.

The data is an asymmetric similarity matrix, denoted by S, because the response to hearing code i first and then code j may not be the same as the reponse to hearing code j first and then code i. Following Everitt and Dunn (2001), S is symmetrised by averaging the corresponding pairs of off-diagonal elements. After this, S is converted into a dissimilarity matrix using the transformation $D = 100 - S$.

Applying two dimensional classical multidimensional scaling to D results in the plot shown in Figure 4.8, where the dashed and dotted lines illustrate the morse codes for the digits. This plot visualises a two dimensional representation of the dissimilarities between the codes: similar codes are close together and dissimilar codes are far apart.

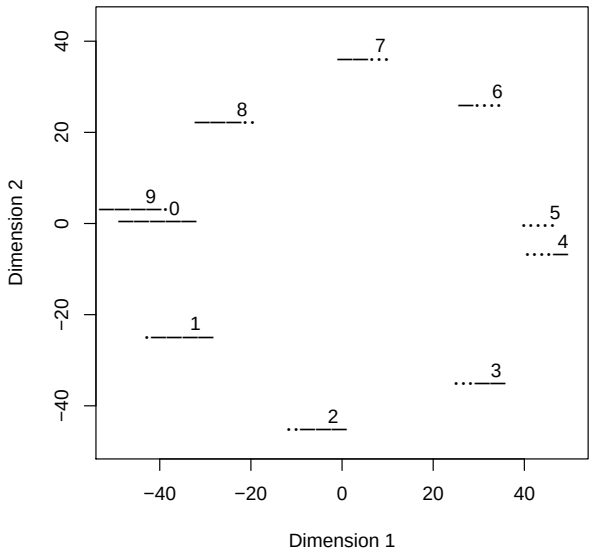
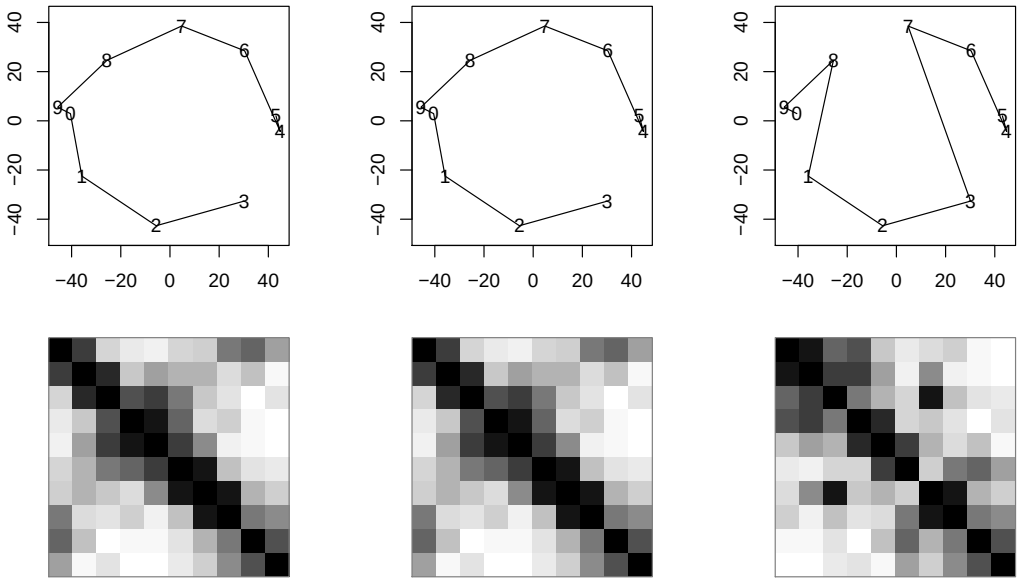


Figure 4.8: Scatterplot of the two dimensional classical multidimensional scaling solution for the dissimilarity matrix of the digits in the Morse code dataset. The dashed and dotted lines illustrate the morse codes for the digits.

The ten codes fall onto a circle and if the codes are ordered along this circle, then the corresponding dissimilarity matrix follows a "circumplex" pattern (see, for example, Wilkinson 2005, §16.5). A dissimilarity matrix follows a circumplex pattern if when moving away from the main diagonal in the matrix, the dissimilarities begin low, then increase to a point and then decrease becoming low again. The heatmap in Figure 4.9.(a) shows the circumplex pattern in the dissimilarity matrix for the Morse code data, where black to white represents low to high dissimilarities.

The codes are clustered using average linkage applied to D and the dendrogram is rearranged using DendSer with each of the PL, BAR and ARc cost functions. The paths through the points in the scatterplots in Figures 4.9.(a), (b) and (c) correspond to the permutations returned by DendSer with each of PL, BAR and ARc respectively. The rows/columns in the heatmaps of the dissimilarity matrix beneath the scatterplots are ordered according to the corresponding permutations.



(a) DendSer + PL (b) DendSer + BAR (c) DendSer + ARc

Figure 4.9: The path through the points in the scatterplot in (a) corresponds to the permutation of digits from using DendSer with PL. The same permutation of digits is used to order the rows/columns in the corresponding heatmap of the dissimilarity matrix of the digits. In (b) the digits are ordered using DendSer with BAR and in (c) the digits are ordered using DendSer with ARc.

DendSer with ARc fails to recover the circular ordering of the morse codes for the following reason. If a dissimilarity matrix follows anti-Robinson form, then the values in this matrix increase when moving away from the main diagonal. However, in a circumplex dissimilarity matrix, the values increase and then decrease when moving away from the main diagonal. It is the region of low values in the north-east and south-west corners of a circumplex dissimilarity matrix (see, for example, the heatmap in Figure 4.9.(a)) that messes up the anti-Robinson pattern.

DendSer with both PL and BAR recover the circular ordering of the morse codes because neither of these cost functions are affected by the region of low values in the north-east and south-west corners of a circumplex dissimilarity

matrix. The PL cost function is the sum of the values just off the main diagonal of a dissimilarity matrix and the BAR cost function is a weighted sum of the values in the first two diagonals off the main diagonal. (Note that this example uses the default value of w for BAR, which is $n/5 = 2$.)

4.6 Fibroblast data

Fibroblasts are a cell type in the body that construct various tissues such as skin. Iyer et al. (1999) analysed fibroblasts in order to determine the importance of fibroblasts in wound repair. In their experiment, human fibroblasts were grown in culture and then deprived of serum for 48 hours. The serum was added back and the expression level of 8613 genes was measured at twelve times ranging from 0 minutes to 24 hours after the re-introduction of the serum. Of the 8613 genes measured, the expression of 517 genes changed substantially in response to the serum. The reader is referred to Iyer et al. (1999) for full details and results of the experiment and the selection process of the 517 genes.

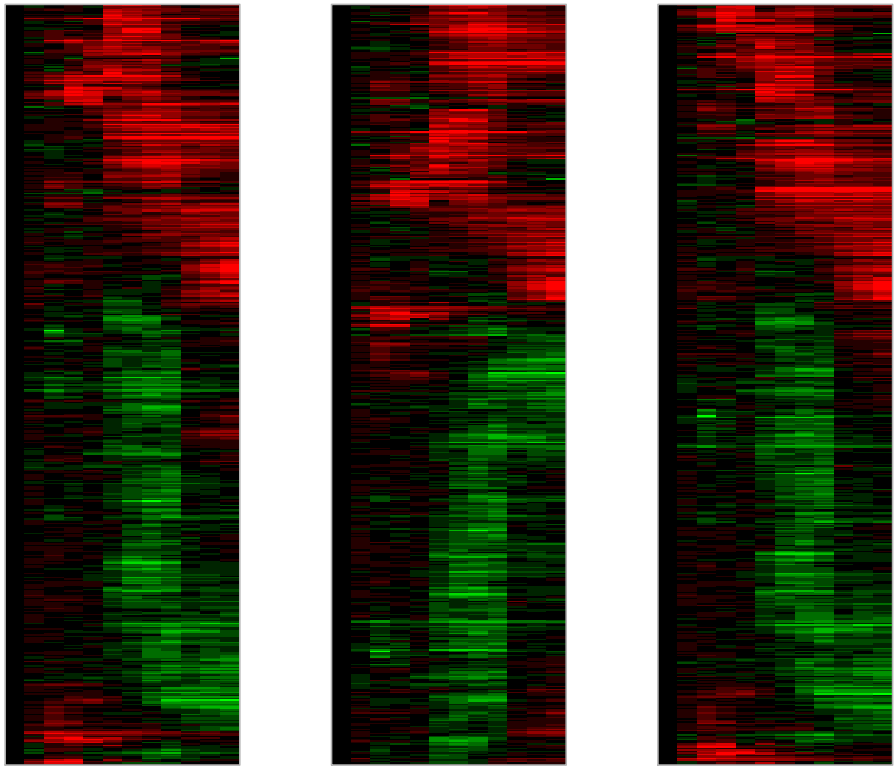
The dataset used in this example contains the expression, relative to time 0, of the 517 genes at twelve different time points: 0, 15 minutes, 30 minutes, 1 hour, 2 hours, 3 hours, 4 hours, 8 hours, 12 hours, 16 hours, 20 hours and 24 hours. Following Eisen et al. (1998), the log transform of the data using base two is taken and so the ij^{th} entry in the dataset is now:

$$\log_2 \left(\frac{\text{expression of gene } i \text{ at time } j}{\text{expression of gene } i \text{ at time } 0} \right). \tag{4.3}$$

With the Fibroblast data, interest lies in finding groups of genes that behave similarly and so a hierarchical clustering of the genes (using Ward's linkage) is performed, where the dissimilarity between genes i and j is measured using $1 - \text{Pearson correlation}(i, j)$, as suggested in Eisen et al. (1998).

Figure 4.10 shows heatmaps of the gene expressions, where the genes (rows) are ordered according to the permutations returned from DendSer with each of the PL, ARc and BAR cost functions respectively. The large patches of red and green indicate groups of genes that share similar expression patterns. Notice that reading the BAR ordered heatmap in Figure 4.10.(c) from top to bottom shows a smoother transition between the expression of the genes than that shown in the ARc ordered heatmap in Figure 4.10.(b) and a slightly smoother transition between the expression of the genes than that shown in the PL ordered heatmap in Figure 4.10.(a).

Figure 4.11 shows heatmaps of the correlation matrices of the genes, where



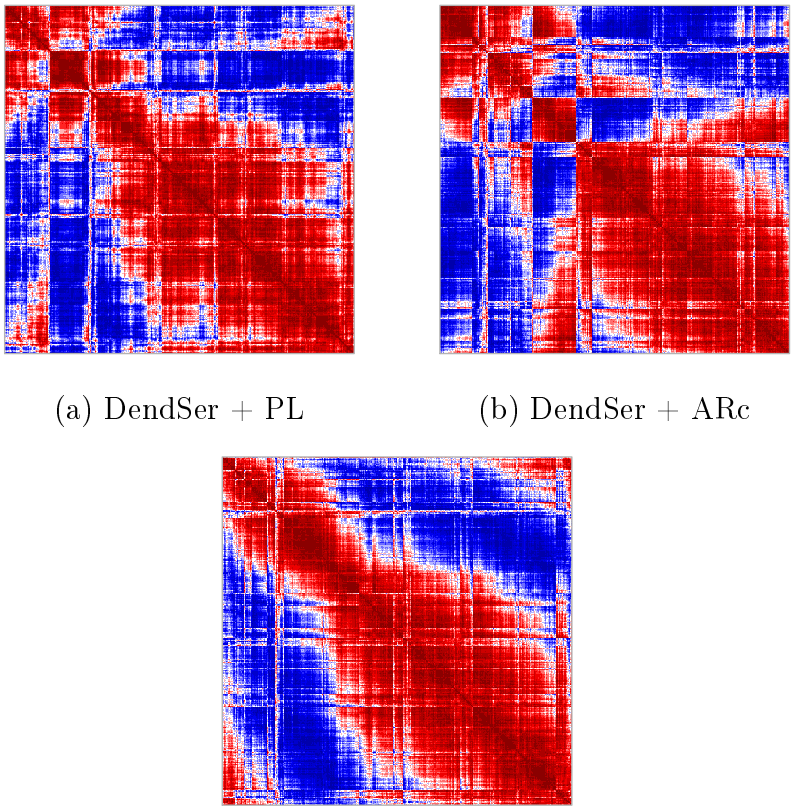
(a) DendSer + PL (b) DendSer + ARc (c) DendSer + BAR

$\log_2(ratio)$

Figure 4.10: Heatmaps of the gene expressions in the Fibroblast dataset. In (a)-(c), the genes (rows) are ordered according to the permutation returned by DendSer with each of PL, ARc and BAR, respectively.

the genes (rows and columns) are ordered according to the permutations returned from DendSer with each of PL, ARc and BAR respectively. Ordering the genes using DendSer with BAR reveals a circumplex pattern in the correlation matrix. This circumplex pattern explains why DendSer with ARc produces a less informative ordering of the genes, for reasons discussed in Section 4.5. However, unlike in Section 4.5, DendSer with PL also struggles to find the circumplex pattern.

The PL cost function only considers dissimilarities between adjacent objects and so minimising PL generally reveals local structure in data but may not reveal more global trends. However, the BAR cost function considers dissimilarities between objects that are up to $n/5$ spaces apart (where n is the number of objects) and so minimising BAR is more suitable for uncovering global patterns in data. This may explain why the PL ordered heatmap in



(a) DendSer + PL

(b) DendSer + ARc

(c) DendSer + BAR

Figure 4.11: Heatmaps of the correlation matrix for the 517 genes in the Fibroblast dataset. In (a)-(c), the genes (rows and columns) are ordered according to the permutation returned from DendSer with PL, ARc and BAR, respectively.

Figure 4.11.(a) does not reveal the circumplex pattern as well as the BAR ordered heatmap in Figure 4.11.(c).

The circumplex pattern is also revealed by the two dimensional classical multidimensional scaling solution of the correlation matrix shown in Figure 4.12, where the 517 genes clearly form a circle.

Returning to the clustering solution, Figure 4.13 shows a heatmap of the gene expressions, where the genes (rows) are ordered according to DendSer with BAR. The colour bar beside the rows of the heatmap shows the eight cluster solution from the hierarchical clustering (informal analysis and subjective assessment suggested an eight cluster solution was appropriate). The graphs on the right show the mean expression for the genes in the corresponding clusters and the dashed line in the graphs represents the value zero.

Reading Figure 4.13 from top to bottom follows a general trend of moving

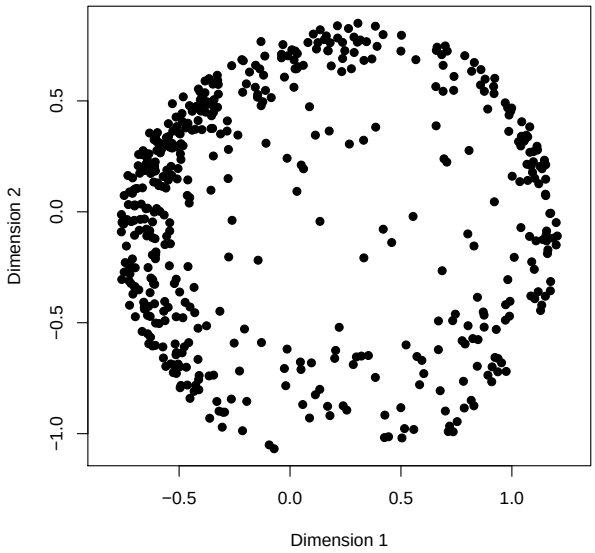


Figure 4.12: Scatterplot of the two dimensional classical multidimensional scaling solution of the correlation matrix for the genes in the Fibroblast dataset.

from high expression (A, B, C, D and E) to low expression (F, G and H). A smooth transition between the clusters is also shown: the maximum expression shifts from approximately 1 hour in cluster A to 24 hours in cluster E and then back to 1 hour at the end of cluster H, the minimum expression shifts from approximately 4 hours in cluster F to 24 hours in cluster H. Clusters A, B, F and G also suggest that the expression of the genes behaves periodically over time. The reader is referred to Spellman et al. (1999), who discussed various reasons as to why genes behave in a periodic manner.

To summarise the analysis, seriating the genes using DendSer with BAR revealed a circumplex pattern in the correlation matrix for the genes, which was not as clearly revealed by using DendSer with PL and not at all revealed by using DendSer with ARc. This circumplex pattern was also not revealed by the analysis in Iyer et al. (1999) because they seriated the genes using the method described in Eisen et al. (1998) (i.e. DendSer with the LS cost function and the T_0 node operation).

Due to the circumplex pattern, seriating using DendSer with BAR showed a smoother transition between the expression of the genes in Figure 4.13 than that shown in Figures 4.10.(a) and (b). This smooth transition made it easier to observe global relationships between the clusters produced by the hierarchical clustering of the genes.

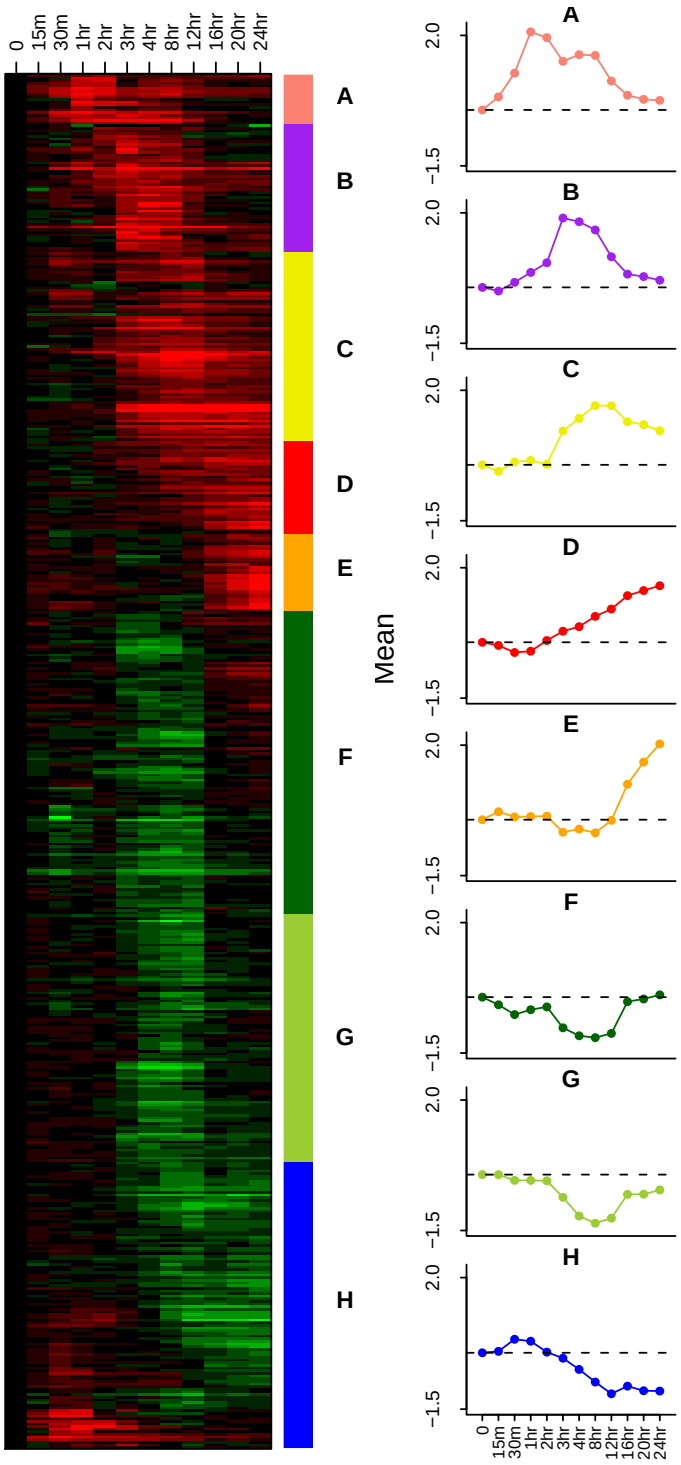


Figure 4.13: Heatmap of the gene expressions in the Fibroblast dataset showing the eight cluster solution from the hierarchical clustering. The genes (rows) in the heatmap are ordered according to the permutation returned by using DendSer with the BAR cost function. The graphs on the right show the mean expression for the genes in the corresponding clusters. The dashed line on the graphs represents the value 0.

4.7 Summary

This chapter highlighted the flexibility of DendSer using several examples encompassing a variety of datasets, visualisations, dissimilarity measures and cost functions, particularly the new cost functions BAR and LPL. This chapter also points out that the user does not need to be interested in clustering their data in order to use DendSer, as shown in the Sleep and Morse code examples in Sections 4.4 and 4.5.

For the Pottery example in Section 4.2, DendSer with BAR worked well in seriating the heatmap of the Euclidean distance matrix in Figure 4.2. DendSer with BAR also produced good seriation results in the Morse code and Fibroblast examples in Sections 4.5 and 4.6, while DendSer with ARc produced less informative seriation results because of the circumplex pattern. These examples suggest that the banded anti-Robinson criterion is suitable for a wider range of data patterns than the anti-Robinson criterion. This flexible nature of the banded anti-Robinson criterion is examined further at the end of Chapter 5.

Chapter 5

A comparison of seriation algorithms

5.1 Introduction

This chapter compares the performance of DendSer with seriation algorithms that are available in the software program `R` (R Development Core Team 2010).

This comparison study follows a similar strategy to Chen (2002), Wilkinson (2005, §16.5), Hahsler et al. (2008) and Tien et al. (2008), who seriated some dataset(s) using a selection of seriation algorithms and then compared the performance of the algorithms using heatmaps and seriation criteria such as path length and anti-Robinson form.

This study, however, compares a broader set of algorithms: Hahsler et al. (2008) did not include algorithms based on dimension reduction techniques, Tien et al. (2008) did not include TSP heuristics, and Chen (2002) and Wilkinson (2005, §16.5) did not include algorithms such as the OLO algorithm (Bar-Joseph et al. 2001) and ARSA (Brusco et al. 2007).

Section 5.2 lists the different seriation algorithms and datasets used in the comparison study, which include the datasets used by Wilkinson (2005, §16.5), Chen (2002) and Hahsler et al. (2008). Section 5.3 compares the performance of the selected algorithms using heatmaps and the path length, anti-Robinson and banded anti-Robinson seriation criteria.

Section 5.4 summarises the performance of the algorithms and also discusses the efficiency of the algorithms. Finally, Section 5.5 discusses the results of the comparison study and gives guidelines on choosing the most suitable algorithm for different seriation interests and visualisations.

5.2 Comparison Tools

5.2.1 Seriation algorithms

The algorithms selected for this comparison study are DendSer and those available in the software program `R` (R Development Core Team 2010).

Figure 5.1 illustrates the selected algorithms and is also an updated version of the overview of seriation algorithms shown in Figure 2.5 because it now includes DendSer. Note that DendSer is placed in the "Other" category for "Dendrogram seriation" due to the variety of criteria that DendSer can optimise, however DendSer also fits into the "Path length" and "Anti-Robinson form" categories.

The following lists and categorises the algorithms selected for the comparison study into three groups:

1. Algorithms minimising the path length criterion:
   (a) TSP: orders objects by creating a tour using the farthest insertion heuristic and then improving the tour using the 2-Opt heuristic (Croes 1958), see Section 2.5.1.
   (b) OLO: the Optimal Leaf Ordering algorithm from Bar-Joseph et al. (2001) with average linkage (see Section 3.5).
   (c) DendSer with the PL cost function, the node operation R_{01} and average linkage.
2. Algorithms minimising the anti-Robinson or banded anti-Robinson criteria:
   (a) ARSA: the simulated annealing algorithm from Brusco et al. (2007) that tries to maximise the ARc merit function in Equation 2.3 (see Section 2.5.4).
   (b) DendSer with the ARc cost function, the node operation T_0 and average linkage.
   (c) DendSer with the BAR cost function, the node operation R_{01} and average linkage.
3. Other:
   (a) MDS1: orders objects using the one dimensional solution from Kruskal's (1964a, 1964b) non-metric multidimensional scaling method.

(b) PCA1: orders objects according to the scores on the first principal component.

(c) SVD2: computes a singular value decomposition of the data and orders the objects according to the angles formed by the first two eigenvectors, see Friendly and Kwan (2003) and Section 2.5.3.

(d) R2E: the Rank-two Ellipse algorithm described in Chen (2002), see Section 2.5.4. [1]

At least one algorithm is selected from every category in Figure 5.1 except the "Partial enumeration" category (see Section 2.5.2). This is because the dynamic programming (Hubert et al. 2001) and branch-and-bound (Brusco and Stahl 2005) algorithms are not feasible for seriating even moderate numbers of objects. (Note that DendSer with ARc fits into the "Anti-Robinson form" category for "Dendrogram seriation".)

There are other algorithms implemented in the `R seriation` package (Hahsler et al. 2010) that are also not included in this comparison study. The Bond Energy Algorithm (see McCormick et al. 1972 and Section 2.5.4) is omitted because this algorithm generally performs poorly and Arabie and Hubert (1990) questioned its use on non-binary data. The Gruvaeus and Wainer (1972) method is also not included because it generally performs less well than the OLO algorithm.

5.2.2 Datasets

This study compares the performance of the selected seriation algorithms using the Iris dataset (Fisher 1936) and the five synthetic datasets described in Wilkinson (2005, §16.5). Using these datasets makes the results of the comparison study in this chapter comparable to the results in Chen (2002), Wilkinson (2005, §16.5) and Hahsler et al. (2008). This study also uses the Laser dataset (see Section 3.4.2) because the seriation results for this dataset help to understand how the seriation algorithms work and how they differ from each other.

The synthetic datasets have 80 rows and 80 columns, and are generated using the formulae described in Wilkinson (2005, §16.5), which are included in Appendix B. Figures 5.2.(a)-(e) show three heatmaps for the Band, Simplex, Circumplex, Equi-correlation and Block datasets respectively. The first row of

[1] In the R2E algorithm, when the objects project onto an ellipse, there are two cutting points for forming the permutation. The `seriation` package implementation of R2E chooses the top most cutting point. However, the version used in this comparison study is adjusted so that R2E chooses the cutting point where the two successive objects are the least similar.

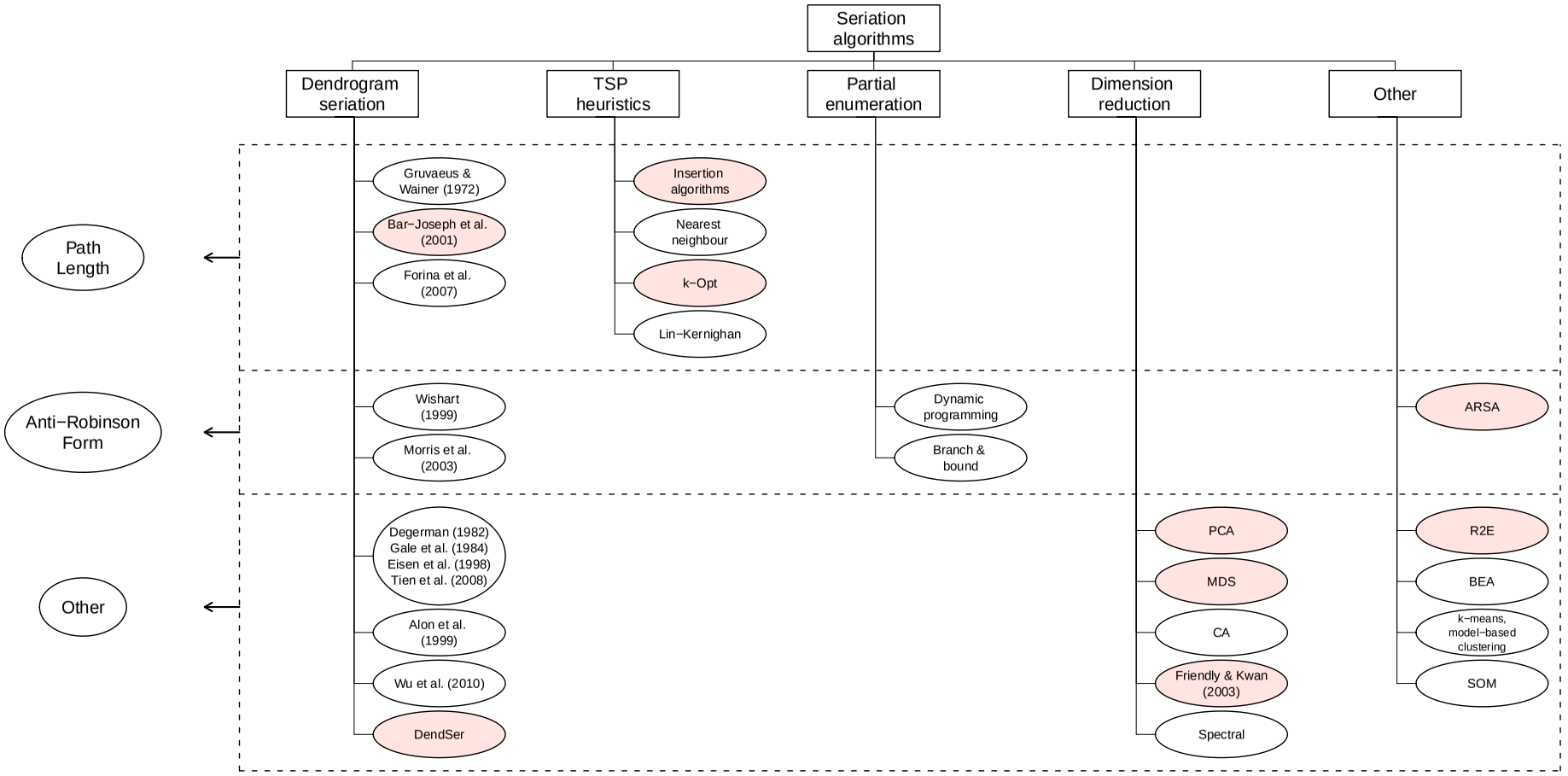
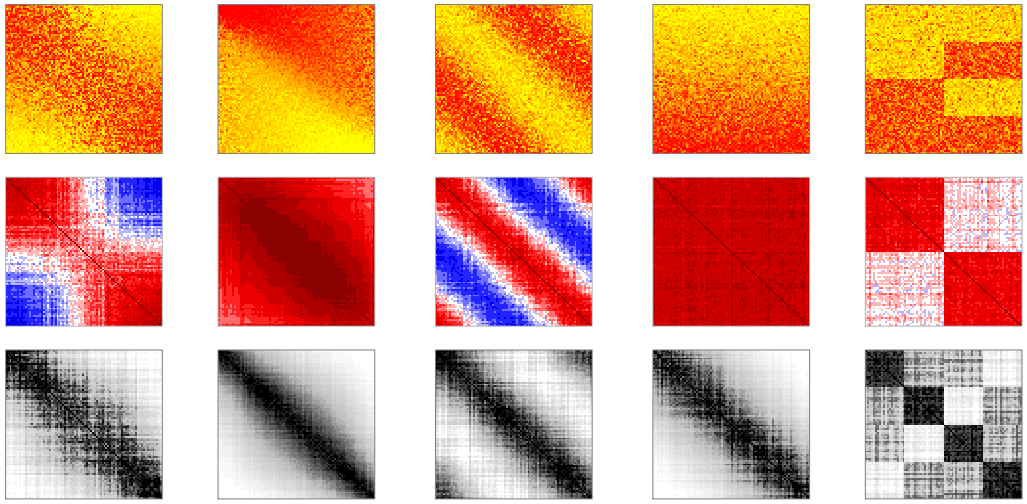


Figure 5.1: Seriation algorithms selected for the comparison study.



(a) Band (b) Simplex (c) Circumplex (d) Equi (e) Block

Figure 5.2: Heatmaps of the five synthetic datasets used in this comparison study. Columns (a)-(e) show heatmaps for the Band, Simplex, Circumplex, Equi-correlation and Block datasets respectively. The first row of heatmaps visualise the raw data, where yellow to red indicates low to high values. The second row of heatmaps visualise the Pearson correlation between the columns, where red to blue indicates high positive to low negative correlation. The third row of heatmaps visualise the Euclidean distance between the rows, where black to white indicates low to high Euclidean distance.

heatmaps in Figure 5.2 visualise the raw data, where yellow to red indicates low to high values. The second row of heatmaps visualise the Pearson correlation between the columns, where red to blue indicates high positive to low negative correlation. The third row of heatmaps visualise the Euclidean distance between the rows, where black to white indicates low to high Euclidean distance.

The synthetic datasets are generated to have the following patterns:

1. In the Band dataset, the correlations between near columns are positive and the correlations between distant columns are negative.

2. In the Simplex dataset, the correlation between all pairs of columns is positive with the correlation between near columns higher than the correlation between distant columns.

3. In the Circumplex dataset, the correlations between near columns are positive and the correlations between distant columns are negative. The first few columns and the last few columns are also positively correlated.

4. In the Equi-correlation dataset, the correlation between all pairs of columns is quite large and positive.

5. In the Block dataset, the columns divide into two blocks and the rows divide into four blocks. The correlations between columns from the same block are near one and the correlations between columns from different blocks are near zero.

Following Wilkinson (2005, §16.5), the synthetic datasets are scrambled by randomly permuting the rows and columns, and the dissimilarity between the rows and between the columns is computed using the Euclidean distance. The Euclidean distance matrices for the columns in the synthetic datasets follow the same patterns as the correlation matrices for the columns (shown in the second row of Figure 5.2). This is because the columns are standardised, which means that the Euclidean distance ($d_{i,j}$) and the correlation ($r_{i,j}$) between the columns i and j are related in the following way:

$$d_{i,j} = \sqrt{(2n-2)(1-r_{i,j})}. \tag{5.1}$$

For each of the Iris and Laser datasets, the variables are standardised and then the dissimilarity between the rows is computed using the Euclidean distance.

5.3 Results of the comparison study

For each of the seven datasets, the following sections compare the performance of the seriation algorithms in two ways: how well they recover patterns in the heatmaps of the dissimilarity matrices and how well they optimise the path length, anti-Robinson and banded anti-Robinson criteria.

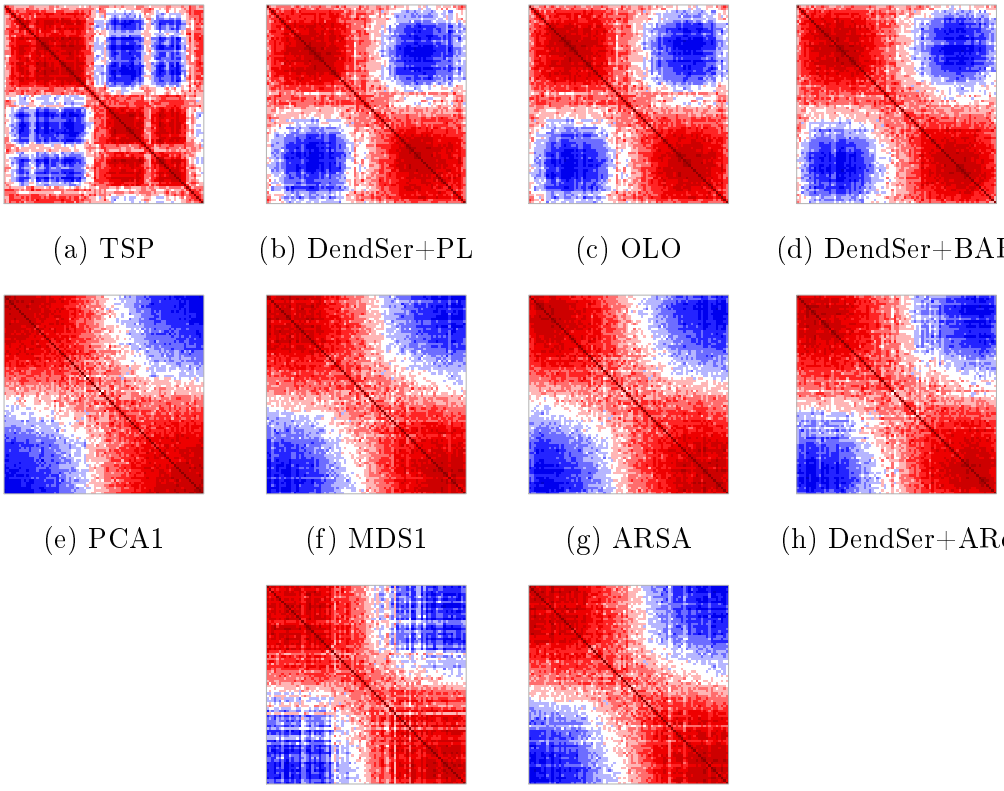
5.3.1 Band dataset

The algorithms perform similarly for both the rows and the columns in the Band dataset and so this section only discusses the results for the columns. Figure 5.3 shows ten heatmaps of the correlation matrix for the columns in the Band dataset, where the rows/columns of the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithm.

Twenty Band datasets are generated and the columns in each dataset are permuted using each of the seriation algorithms. For each seriation algorithm, Figures 5.4.(a), (b) and (c) contain dotcharts showing the mean PL, ARc and BAR values for the twenty permutations, relative to the best mean PL, ARc and BAR values respectively. Within each dotchart, the algorithms are ordered from top to bottom in order of best to worst.

The heatmaps in Figure 5.3 show that all algorithms except TSP recover the original pattern in the dissimilarity matrix for the columns in the Band dataset, although some algorithms are more successful than others.

PCA1, MDS1, ARSA, DendSer+ARc (all in the second row of Figure 5.3) and SVD2 (third row of Figure 5.3) produce the "smoothest" heatmaps and Figure 5.4.(b) shows that these algorithms also produce permutations that have the best ARc values. These five algorithms produce the smoothest heatmaps because they are concerned with the global structure in the dissimilarity matrix, which in this situation closely follows anti-Robinson form.



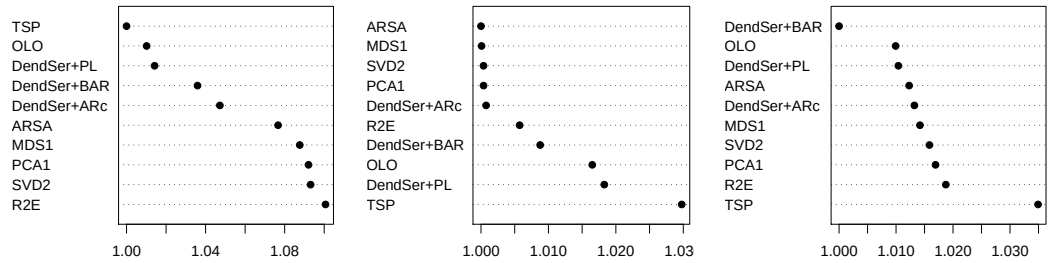
(a) TSP (b) DendSer+PL (c) OLO (d) DendSer+BAR

(e) PCA1 (f) MDS1 (g) ARSA (h) DendSer+ARc

(i) R2E (j) SVD2

Figure 5.3: Heatmaps of the correlation matrix for the columns in the Band dataset, where the rows/columns in the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithms.

DendSer+BAR, DendSer+PL and OLO (all in the first row of Figure 5.3) produce heatmaps that are less smooth than the heatmaps produced by PCA1, MDS1 ARSA, DendSer+ARc and SVD2. This is because these three algorithms are not concerned with the global structure of the dissimilarity matrix: DendSer+BAR is concerned with the structure within a band around the main diagonal, and OLO and DendSer+PL are concerned with the structure just off the main diagonal. However, Figure 5.4.(c) shows that these three algorithms



(a) PL values (b) ARc values (c) BAR values

Figure 5.4: Dotcharts of the mean PL, ARc and BAR values, relative to the best mean PL, ARc and BAR values respectively, for permutations of the columns in twenty Band datasets returned by each seriation algorithm.

produce permutations having the best BAR values.

The R2E algorithm also results in a heatmap (Figure 5.3.(i)) that is less smooth that the heatmaps corresponding to PCA1, MDS1, ARSA, SVD2 and DendSer+ARc.

TSP finds the shortest path through the columns (Figure 5.4.(a)), however TSP fails to recover the original pattern in the dissimilarity matrix. This coincides with the dotcharts in Figures 5.4.(b) and (c), which show that the permutations returned by TSP have the worst ARc and BAR values. TSP produces the least smooth heatmap because it minimises the PL cost function and so is only concerned with structure just off the main diagonal in the dissimilarity matrix. OLO and DendSer+PL also minimise the PL cost function, however their underlying hierarchical clustering structure helps them produce smoother heatmaps than TSP.

5.3.2 Simplex dataset

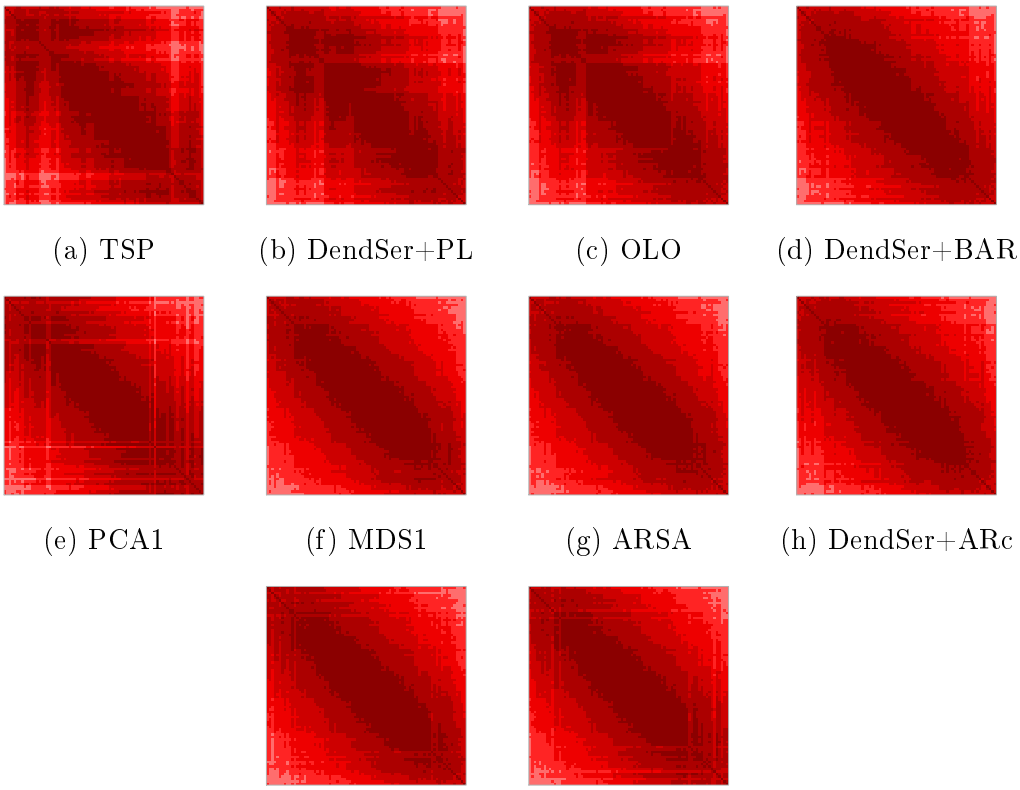
All algorithms except TSP recover the original pattern in the dissimilarity matrix for the rows in the Simplex dataset. TSP does not recover the pattern as well as the other algorithms for the same reason it does not successfully recover the pattern in the columns for the Band dataset. The remainder of this section concerns the seriation results for the columns in the Simplex dataset.

Figure 5.5 shows ten heatmaps of the correlation matrix for the columns in the Simplex dataset, where the rows/columns of the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithms.

Twenty Simplex datasets are generated and the columns in each dataset are permuted using each of the seriation algorithms. For each seriation algorithm, Figures 5.6.(a), (b) and (c) contain dotcharts showing the mean PL, ARc and BAR values for the twenty permutations, relative to the best mean PL, ARc

and BAR values respectively. Within each dotchart, the algorithms are ordered from top to bottom in order of best to worst.

The heatmaps in Figure 5.5 show that DendSer+BAR, MDS1, ARSA, DendSer+ARc and R2E are the most successful in recovering the original pattern in the columns of the Simplex dataset. These five algorithms result in the smoothest heatmaps and also produce permutations with the best ARc and BAR values (Figures 5.6.(b) and (c)).



(a) TSP (b) DendSer+PL (c) OLO (d) DendSer+BAR

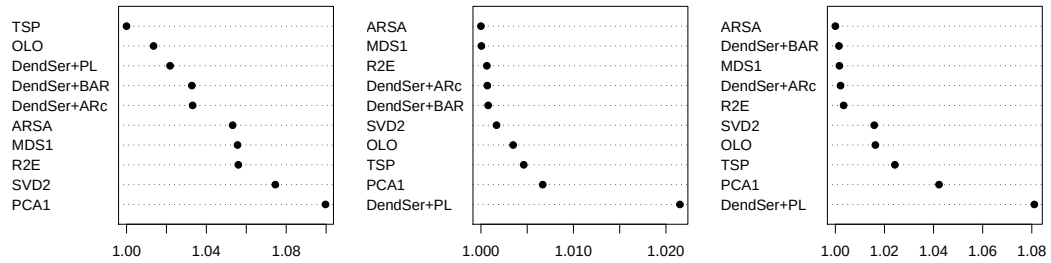
(e) PCA1 (f) MDS1 (g) ARSA (h) DendSer+ARc

(i) R2E (j) SVD2

Figure 5.5: Heatmaps of the correlation matrix for the columns in the Simplex dataset, where the rows/columns in the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithms.

TSP, DendSer+PL and OLO (all in the first row of Figure 5.5) result in less smooth heatmaps because they minimise the PL cost function and so are only concerned with structure just off the main diagonal. However, these three algorithms, unsurprisingly, produce permutations with the best PL values (Figure 5.6.(a)).

The heatmap for PCA1 (Figure 5.5.(e)) shows a "grid" effect in the four corners of the heatmap. This effect is explained by the following. Principal components analysis is equivalent to classical multidimensional scaling (MDS) with Euclidean distance (see, for example, Cox and Cox 1984, pg. 34). There-

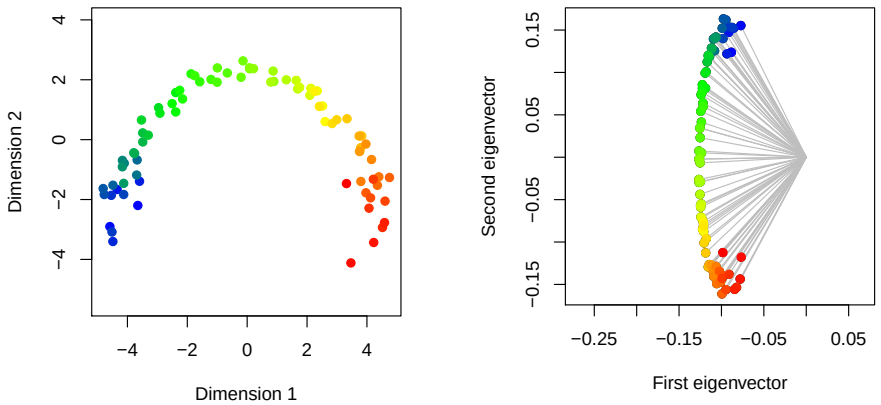


(a) PL values (b) ARc values (c) BAR values

Figure 5.6: Dotcharts of the mean PL, ARc and BAR values, relative to the best mean PL, ARc and BAR values respectively, for permutations of the columns in twenty Simplex datasets returned by each seriation algorithm.

fore, ordering the columns according to the scores on the first principal component is equivalent to ordering the columns according to the one dimensional classical MDS solution of the Euclidean distance matrix for the columns in the Simplex dataset.

Figure 5.7.(a) shows the two dimensional classical MDS solution for the columns in the Simplex dataset, where the points are coloured according to the original column indexes. The points form a horse-shoe shape (see Kendall 1971 for a discussion of the horse-shoe shape in MDS solutions) and so ordering the columns based on the one dimensional solution mixes the blue and red points at the ends of the horse-shoe with some of the green and orange points respectively. This results in the grid effect in the heatmap in Figure 5.5.(e).



(a) MDS solution (b) SVD solution

Figure 5.7: Scatterplots of the two dimensional classical multidimensional scaling solution of the Euclidean distance matrix for the columns in the Simplex dataset, and the first and second eigenvectors of the columns.

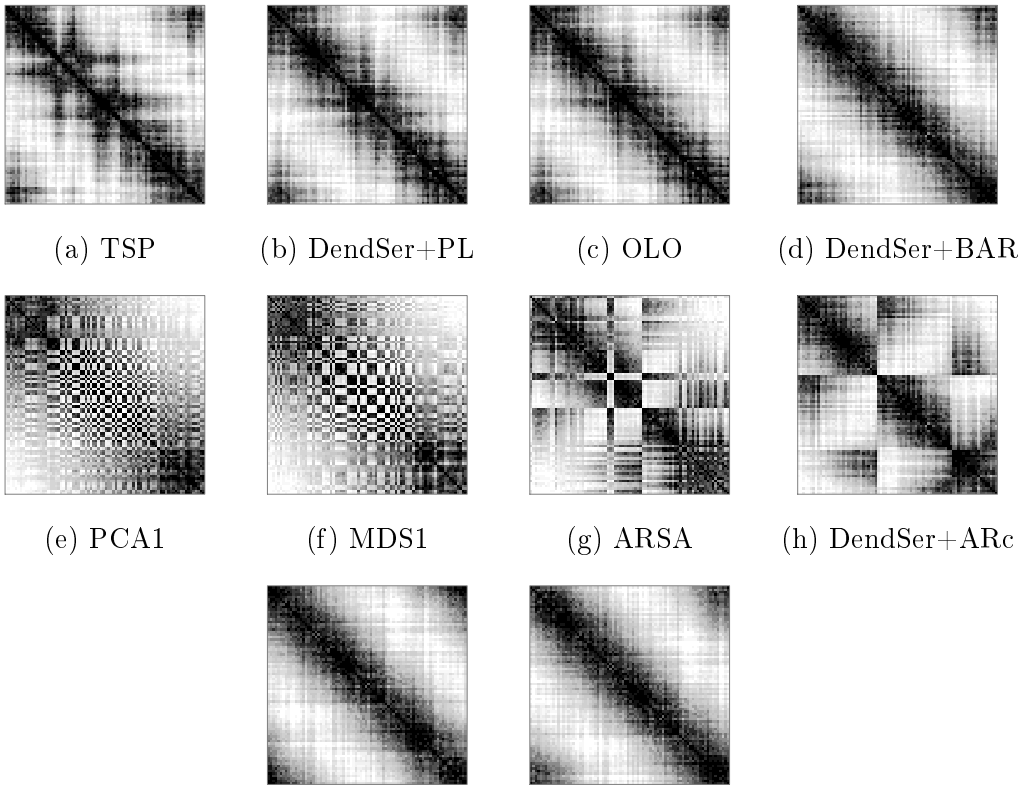
SVD2 also results in a faint grid effect in the north-west and south-east corners of the heatmap in Figure 5.5.(j). This is explained by examining Figure 5.7.(b), which shows a scatterplot of the first and second eigenvectors of the

columns in the Simplex dataset with the points coloured according to the original column indexes. SVD2 orders the objects according to the angles formed by the eigenvectors in Figure 5.7.(b), however, SVD2 does not take the *length* of the vectors into account. This means that SVD2 mixes some of the blue and green points in the top of Figure 5.7.(b) together, and also some of the red and orange points in the bottom of Figure 5.7.(b) together.

5.3.3 Circumplex dataset

The heatmaps in Figure 5.2.(c) show that the dissimilarity matrices for the rows and columns in the Circumplex dataset follow a similar pattern. Therefore, the algorithms perform similarly for both the rows and the columns in the Circumplex dataset and so this section only discusses the results for the rows.

Figure 5.8 shows ten heatmaps of the Euclidean distance matrix for the rows in the Circumplex dataset, where the rows/columns of the heatmaps are ordered according to the permutations returned by the seriation algorithms.



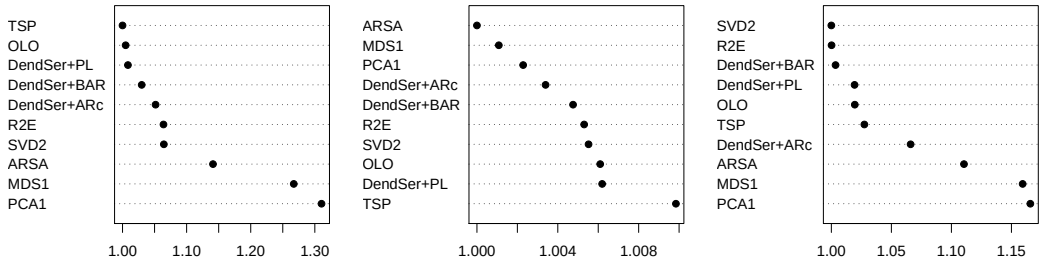
(a) TSP (b) DendSer+PL (c) OLO (d) DendSer+BAR

(e) PCA1 (f) MDS1 (g) ARSA (h) DendSer+ARc

(i) R2E (j) SVD2

Figure 5.8: Heatmaps of the Euclidean distance matrix for the rows in the Circumplex dataset, where the rows/columns in the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithms.

Twenty Circumplex datasets are generated and the rows in each dataset are permuted using each of the seriation algorithms. For each seriation algorithm, Figures 5.9.(a), (b) and (c) contain dotcharts showing the mean PL, ARc and BAR values for the twenty permutations, relative to the best mean PL, ARc and BAR values respectively. Within each dotchart, the algorithms are ordered from top to bottom in order of best to worst.



(a) PL values (b) ARc values (c) BAR values

Figure 5.9: Dotcharts of the mean PL, ARc and BAR values, relative to the best mean PL, ARc and BAR values respectively, for permutations of the rows in twenty Circumplex datasets returned by each seriation algorithm.

Although PCA1, MDS1, ARSA and DendSer+ARc produce permutations with the best ARc values (Figure 5.9.(b)), the heatmaps in the second row of Figure 5.8 show that these algorithms fail to recover the original pattern in the rows of the Circumplex dataset. ARSA and DendSer+ARc fail because they try to optimise anti-Robinson form, which may not produce useful seriation results when data follow a circumplex pattern (see Section 4.5). PCA1 and MDS1 fail because they are based on one dimensional representations of data. However, the rows in the Circumplex dataset require two dimensions to be adequately represented, as shown by the two dimensional classical MDS solution of the Euclidean distance matrix for the rows in Figure 5.10.

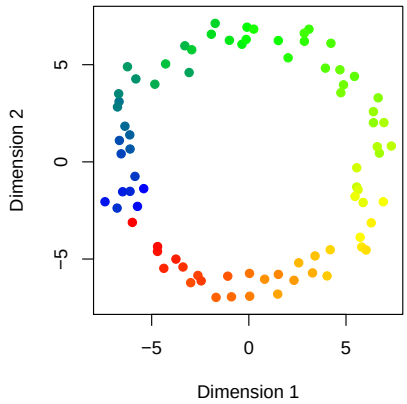
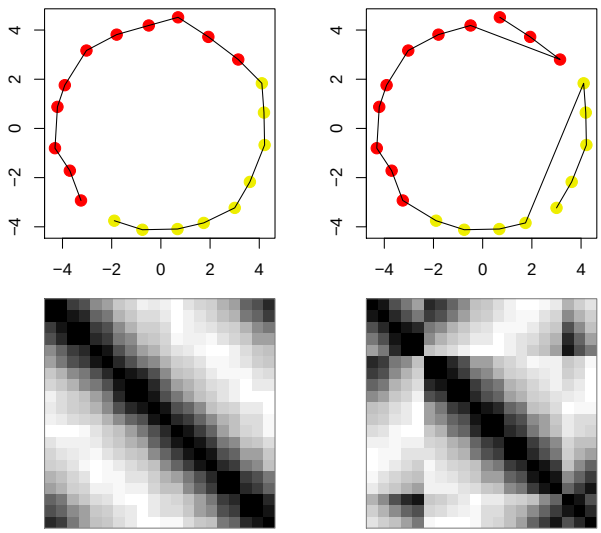


Figure 5.10: Scatterplot of the two dimensional classical multidimensional scaling solution of the Euclidean distance matrix for the rows in the Circumplex dataset. The points are coloured according to the original rows indexes.

The first and third rows of heatmaps in Figure 5.8 show that the other six algorithms recover the original pattern in the rows of the Circumplex dataset with R2E, SVD2 and DendSer+BAR producing the smoothest heatmaps. This coincides with Figure 5.9.(c), which shows that DendSer+BAR, R2E and SVD2 produce permutations with the best BAR values. Figure 5.9.(a) shows that TSP, OLO and DendSer+PL produce permutations with the best PL values.

The remainder of this section compares the performance of DendSer+BAR and DendSer+ARc on a smaller Circumplex dataset, which contains twenty rows and twenty columns. The scatterplots in Figure 5.11 show the two dimensional classical MDS solution of the Euclidean distance matrix for the rows in the smaller Circumplex dataset. The path through the points in the scatterplots in Figures 5.11.(a) and (b) correspond to the permutations returned by DendSer+BAR and DendSer+ARc respectively. The same permutations are used to order the rows/columns of the corresponding heatmaps of the Euclidean distance matrix for the rows in the Circumplex dataset.

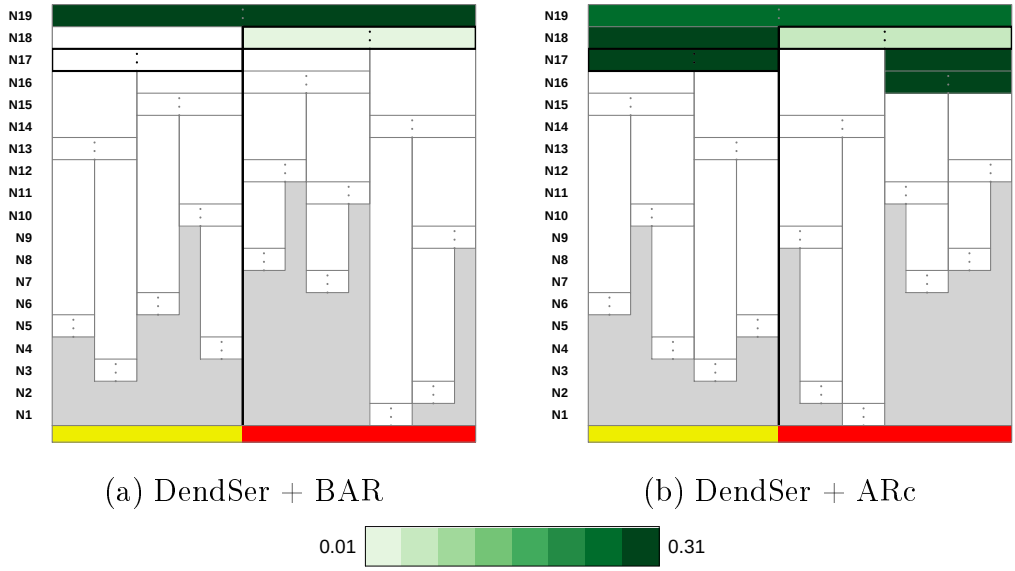


(a) DendSer + BAR (b) DendSer + ARc

Figure 5.11: The scatterplots show the two dimensional classical MDS solution of the Euclidean distance matrix for the rows in a Circumplex dataset, which contains twenty rows and twenty columns. The path through the points in Figures (a) and (b) correspond to the permutations returned by DendSer+BAR and DendSer+ARc respectively. The same permutations are used to order the rows/columns in the corresponding heatmaps of the Euclidean distance matrix. The points in the scatterplots are coloured according to whether they belong to the second last or third last node formed by the hierarchical clustering process.

As with the larger Circumplex dataset discussed above, DendSer+BAR re-

covers the circular ordering of the rows in the smaller Circumplex dataset, while DendSer+ARc fails to do so. However, for this smaller Circumplex dataset, the icicle plots in Figure 5.12 allow a closer inspection of the permutations returned by DendSer+BAR and DendSer+ARc.



(a) DendSer + BAR (b) DendSer + ARc

Proportion of anti-Robinson violations

Figure 5.12: Icicle plots visualising a hierarchical clustering of a Circumplex dataset The objects (columns) in (a) and (b) are ordered according to DendSer+BAR and DendSer+ARc respectively. Each node is coloured according to the proportion of anti-Robinson violations in the sub-Euclidean distance matrix for the (ordered) objects in the node. The colour scale, light to dark green, indicates low to high proportions of anti-Robinson violations and white indicates zero anti-Robinson violations. The colour bar at the bottom of the icicle plots indicates which objects belong to the nodes N_{17} and N_{18}.

The icicle plots are constructed as described in Section 2.4, however in this case each node is coloured according to the proportion of anti-Robinson violations in the sub-Euclidean distance matrix for the (ordered) objects in the node. The colour scale, light to dark green, indicates low to high proportions of anti-Robinson violations and white indicates zero anti-Robinson violations.

The groups of yellow and red points in the scatterplots in Figure 5.11 correspond to the objects that are contained in nodes N_{17} and N_{18} respectively in the icicle plots in Figure 5.12. Both of these nodes are highlighted using black lines in Figures 5.12.(a) and (b). The colour bar at the bottom of the icicle plots indicates which objects belong to the nodes N_{17} and N_{18}.

The icicle plots in Figure 5.12 show that DendSer+ARc results in less anti-Robinson violations in the Euclidean distance matrix than DendSer+BAR.

This is clear by the lighter green colour of the final node N_{19} in Figure 5.12.(b) than in Figure 5.12.(a).

The icicle plot in Figure 5.12.(a) also shows that DendSer+BAR is more concerned with good *local* anti-Robinson form in the Euclidean distance matrix. This is shown by the white colour of the nodes $N_1, \ldots, N_{17}$, which indicates that the ordering of the objects in these nodes results in their corresponding sub-Euclidean distance matrices following anti-Robinson form. The node N_{18} is also coloured a very light green, which indicates that its corresponding sub-Euclidean distance matrix contains a very small proportion of anti-Robinson violations.

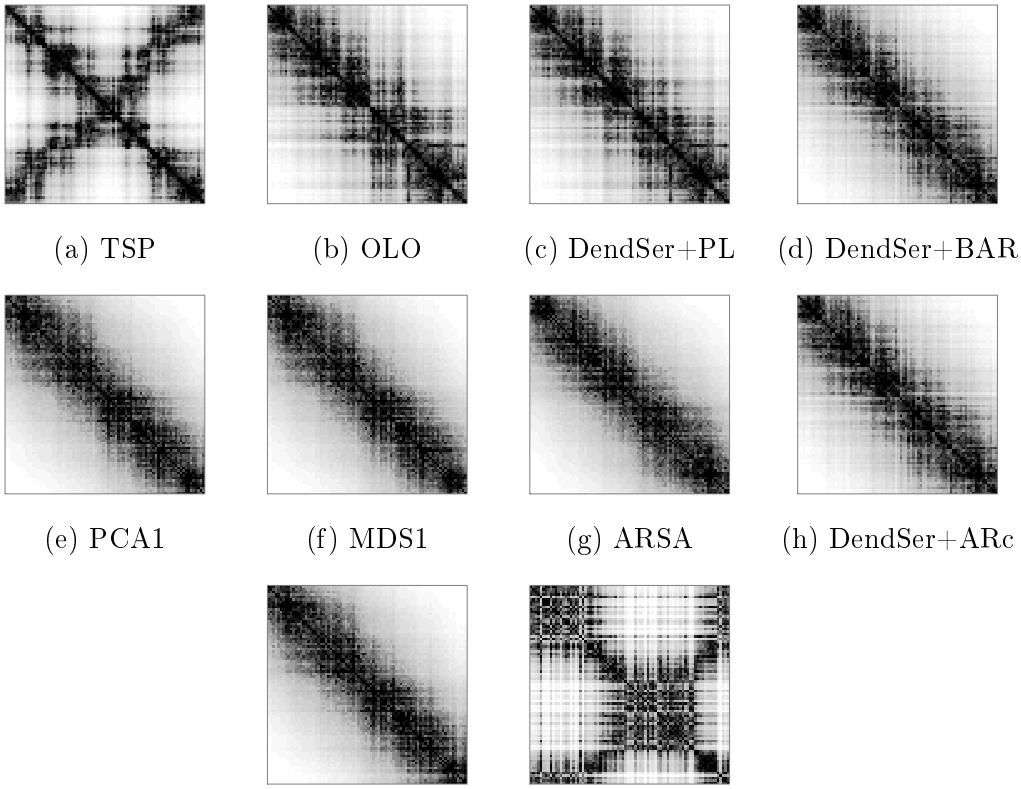
DendSer+ARc, however, sacrifices good *local* anti-Robinson form in order to produce better *global* anti-Robinson form in the Euclidean distance matrix. This is shown by the dark green colour of the nodes N_{16} and N_{17}, which indicates that their corresponding sub-Euclidean distance matrices contain a large proportion of anti-Robinson violations. The colour of the node N_{18} in Figure 5.12.(b) is also slightly darker than in Figure 5.12.(a), which means that the DendSer+ARc ordering of the objects leads to the sub-Euclidean distance matrix for N_{18} containing a higher proportion of anti-Robinson violations than the DendSer+BAR ordering of the objects.

5.3.4 Equi-correlation dataset

This section does not discuss the seriation results for the columns in the Equi-correlation dataset because there is no structure for the algorithms to recover (see the middle heatmap in Figure 5.2.(d)).

Figure 5.13 shows ten heatmaps of the Euclidean distance matrix for the rows in the Equi-correlation dataset, where the rows/columns of the heatmap are ordered according to the permutations returned by the corresponding seriation algorithms.

Twenty Equi-correlation datasets are generated and the rows in each dataset are permuted using each of the seriation algorithms. For each seriation algorithm, Figures 5.14.(a), (b) and (c) contain dotcharts showing the mean PL, ARc and BAR values for the twenty permutations, relative to the best mean PL, ARc and BAR values respectively. Within each dotchart, the algorithms are ordered from top to bottom in order of best to worst. Note that TSP and SVD2 produce permutations having very poor ARc values (1.092 and 1.098 respectively) and BAR values (1.083 and 1.156 respectively), and so the upper limits for the x-axes in Figures 5.14.(b) and (c) are truncated to avoid loss of resolution.

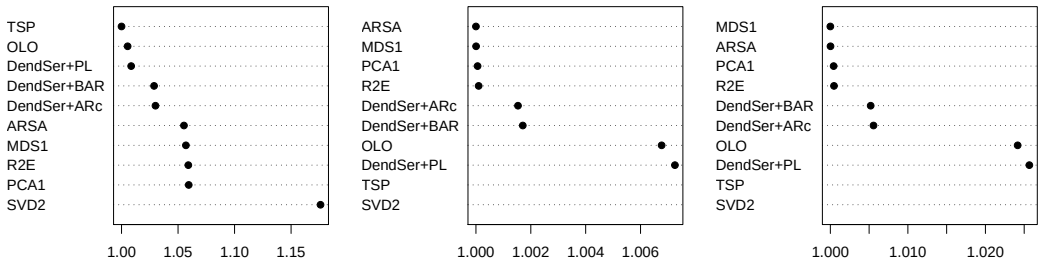


(a) TSP (b) OLO (c) DendSer+PL (d) DendSer+BAR

(e) PCA1 (f) MDS1 (g) ARSA (h) DendSer+ARc

(i) R2E (j) SVD2

Figure 5.13: Heatmaps of the Euclidean distance matrix for the rows in the Equi-correlation dataset, where the rows/columns in the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithms.



(a) PL values (b) ARc values (c) BAR values

Figure 5.14: Dotcharts of the mean PL, ARc and BAR values, relative to the best mean PL, ARc and BAR values respectively, for permutations of the rows in twenty Equi-correlation datasets returned by each seriation algorithms.

Figure 5.13 shows that all algorithms successfully recover the original pattern in the rows for the Equi-correlation dataset except for TSP and SVD2. TSP fails to recover the pattern as well as the other algorithms for the same reason that it does not recover the pattern in the columns for the Band dataset. The reason that SVD2 fails to recover the original pattern is explained by examining Figure 5.15, which shows a scatterplot of the first and second eigen-

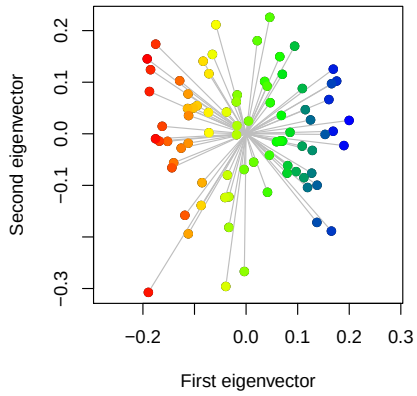


Figure 5.15: Scatterplot of the first and second eigenvectors of the rows in the Equi-correlation dataset. The points are coloured according to the original rows indexes.

vectors of the rows in the Equi-correlation dataset with the points coloured by the original row indexes.

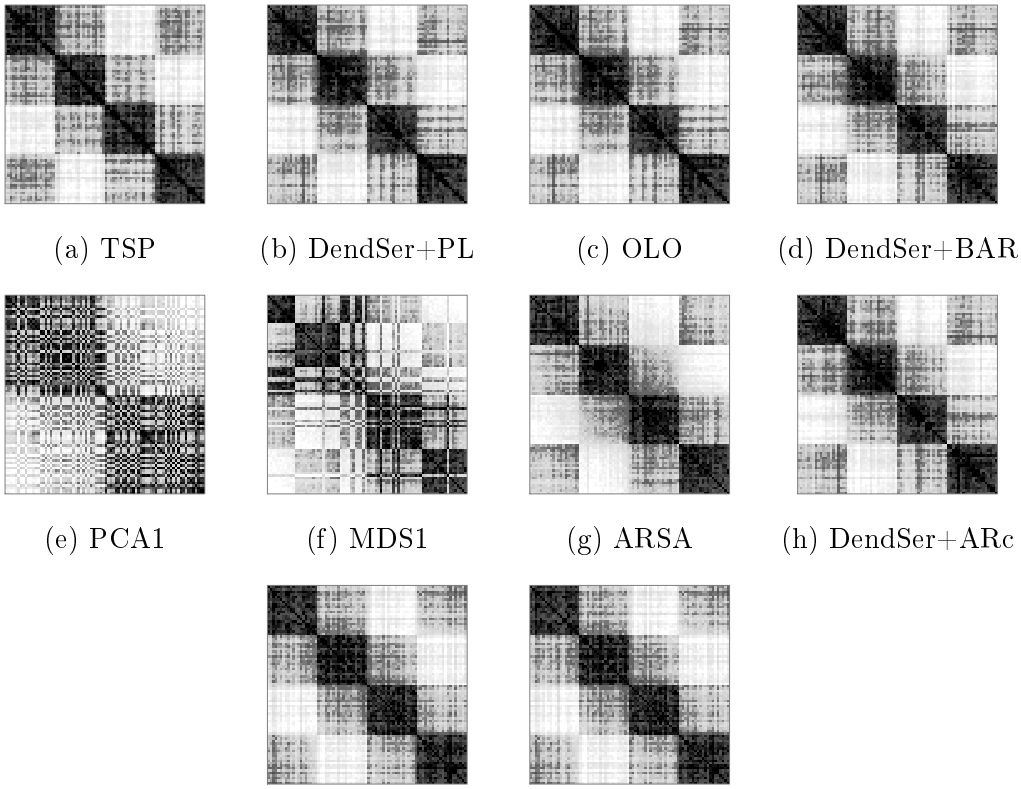
As discussed in the results for the Simplex dataset, SVD2 orders the rows according to the angles formed by the eigenvectors in Figure 5.15 but does not take the length of the vectors into account. Therefore, SVD2 mixes the blue and green points together and also the red and yellow points.

For the eight algorithms that recover the original pattern, Figures 5.13 and 5.14 show that:

- PCA1, MDS1, ARSA and R2E produce permutations giving the smoothest heatmaps and the best ARc and BAR values.
- DendSer+ARc and DendSer+BAR produce permutations giving the next smoothest heatmaps and the next best ARc and BAR values.
- OLO and DendSer+PL produce permutations giving the least smooth heatmaps, however these algorithms, along with TSP, produce permutations having the best PL values.

5.3.5 Block dataset

All of the algorithms recover the original two block pattern in the columns for the Block dataset (see the middle heatmap in Figure 5.2.(e)) and so this section only discusses the results for the rows in the Block dataset. Figure 5.16 shows ten heatmaps of the Euclidean distance matrix for the rows in the Block dataset, where the rows/columns of the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithms.



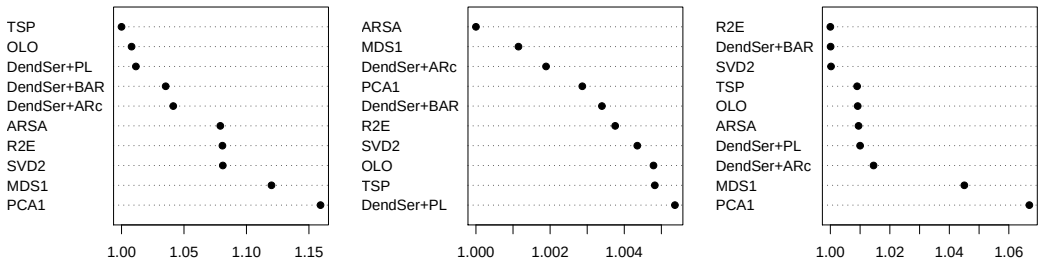
(a) TSP (b) DendSer+PL (c) OLO (d) DendSer+BAR

(e) PCA1 (f) MDS1 (g) ARSA (h) DendSer+ARc

(i) R2E (j) SVD2

Figure 5.16: Heatmaps of the Euclidean distance matrix for the rows in the Block dataset, where the rows/columns in the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithms.

Twenty Block datasets are generated and the rows in each dataset are permuted using each of the seriation algorithms. For each seriation algorithm, Figures 5.17.(a), (b) and (c) contain dotcharts showing the mean PL, ARc and BAR values for the twenty permutations, relative to the best mean PL, ARc and BAR values respectively. Within each dotchart, the algorithms are ordered from top to bottom in order of best to worst.

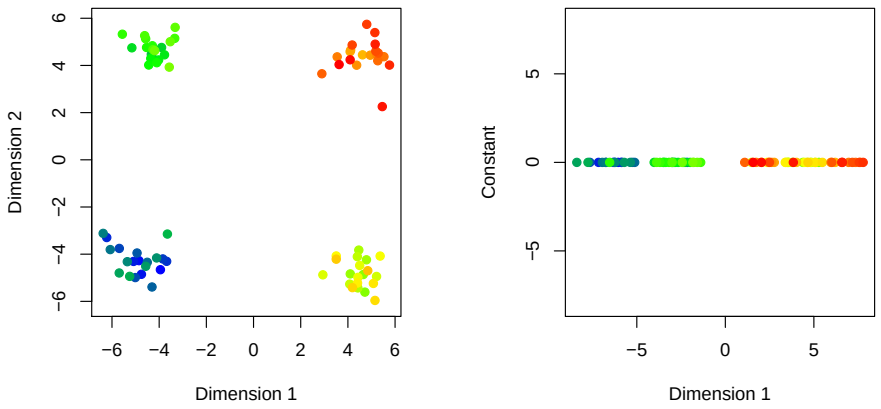


(a) PL values (b) ARc values (c) BAR values

Figure 5.17: Dotcharts of the mean PL, ARc and BAR values, relative to the best mean PL, ARc and BAR values respectively, for permutations of the rows in twenty Block datasets returned by each seriation algorithms.

The heatmaps in Figure 5.16 show that all of the algorithms except PCA1 and MDS1 recover the four blocks in the rows of the Block dataset. The reason PCA1 does not recover the pattern is explained by examining Figure 5.18.(a), which shows the two dimensional classical MDS solution of the Euclidean distance matrix for the rows in the Block dataset with the points coloured by the original row indexes. The four blocks in the rows require two dimensions to be adequately represented. However, PCA1 orders the rows according to the one dimensional MDS solution, which means that PCA1 mixes the green and blue blocks together, and also the red and yellow blocks.

Similarly, MDS1 fails to recover the four blocks because it orders the rows according to the one dimensional non-metric (Kruskal 1964a, 1964b) MDS solution of the Euclidean distance matrix for the rows. Figure 5.18.(b) shows that the non-metric MDS solution also mixes the green and blue, and red and yellow blocks together.

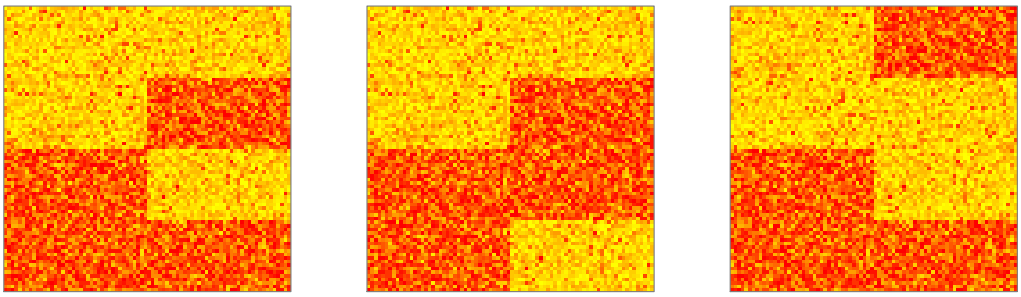


(a) 2-D classical MDS solution (b) 1-D non-metric MDS solution

Figure 5.18: Scatterplots of the two dimensional classical MDS solution and the one dimensional non-metric MDS solution (Kruskal 1964a, 1964b) of the Euclidean distance matrix for the rows in the Block dataset. The points are coloured according to the original row indexes.

Note that, although the other seriation algorithms recover the four blocks in the rows, they do not recover the original ordering of the blocks. The heatmap of the Block dataset in Figure 5.19.(a) shows the original order of the four blocks in the rows of the Block dataset. Some of the seriation algorithms swap the third and fourth blocks, as shown in the heatmap in Figure 5.19.(b), and some of the algorithms swap the first and second blocks, as shown in the heatmap in Figure 5.19.(c).

The ordering of the four blocks recovered by the seriation algorithms is an improvement over the original ordering. This is because the algorithms place two similar blocks in the second and third positions along the main diagonal



(a) (b) (c)

Figure 5.19: Heatmaps of the Block dataset, where yellow to red indicates low to high values. Figure (a) shows the original ordering of the four blocks in the rows, Figure (b) swaps the third and fourth blocks and Figure (c) swaps the first and second blocks.

of the dissimilarity matrix for the rows (see, for example, Figure 5.16.(d)), whereas the original ordering places two dissimilar blocks in the second and third positions of the dissimilarity matrix for the rows (see the bottom heatmap in Figure 5.2.(e)).

The dotcharts in Figure 5.17 point out the following features of the algorithms:

- TSP, OLO and DendSer+PL produce permutations of the rows with the best PL values.
- ARSA, MDS1 and DendSer+ARc produce permutations with the best ARc values.
- DendSer+BAR, R2E and SVD2 produce permutations with the best BAR values.

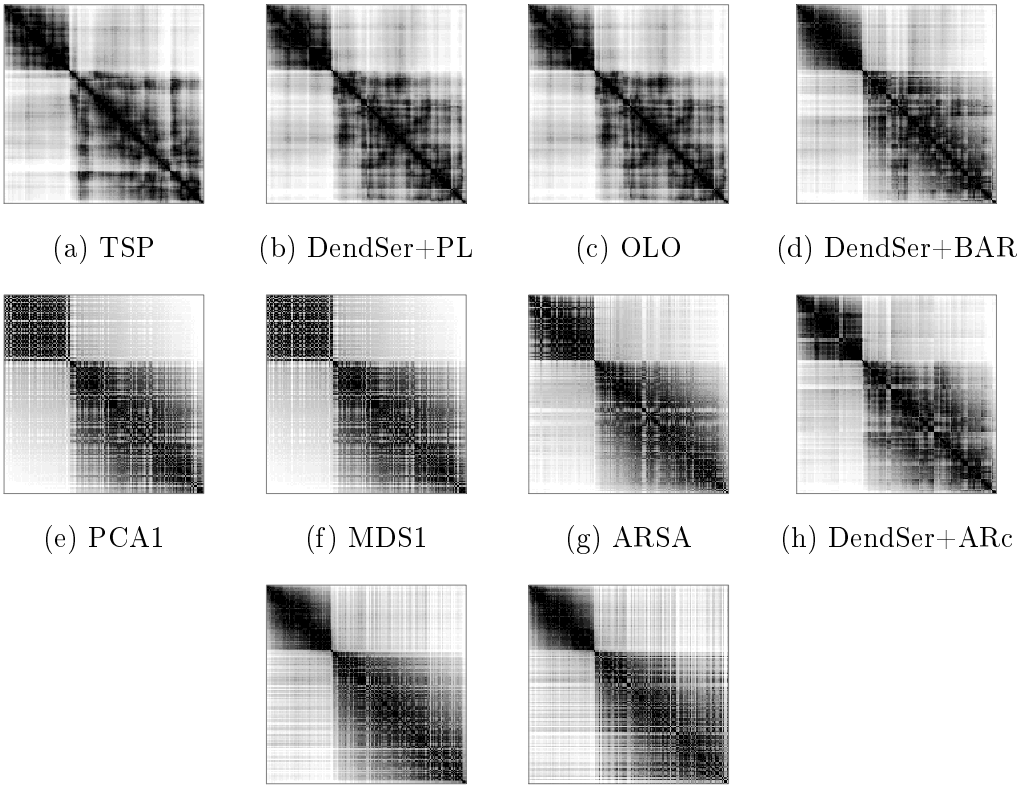
5.3.6 Iris dataset

Figure 5.20 shows ten heatmaps of the Euclidean distance matrix for the rows in the Iris dataset, where the rows/columns of the heatmaps are ordered according to the permutations returned by the corresponding algorithms. Figure 5.21 contains dotcharts of the PL, ARc and BAR values, relative to the best PL, ARc and BAR values respectively, for each of the ten permutations.

Figure 5.20 shows that all of the algorithms recover the two block pattern in the rows of the Iris dataset and the dotcharts in Figure 5.21 show that:

- TSP, OLO and DendSer+PL find the shortest paths through the rows.

- ARSA and DendSer+ARc produce permutations with the best ARc values.
- DendSer+BAR and R2E produce permutations with the best BAR values.

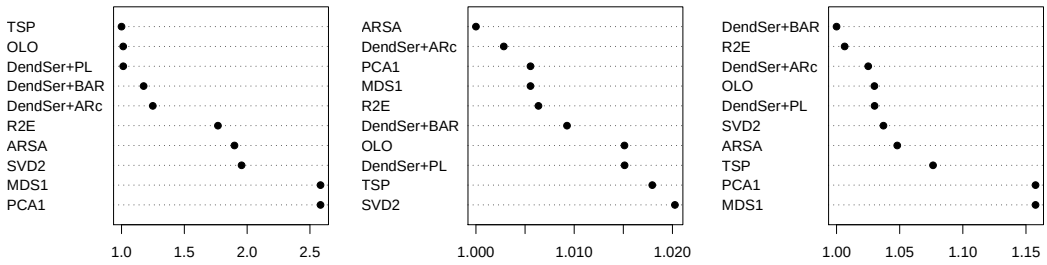


(a) TSP (b) DendSer+PL (c) OLO (d) DendSer+BAR

(e) PCA1 (f) MDS1 (g) ARSA (h) DendSer+ARc

(i) R2E (j) SVD2

Figure 5.20: Heatmaps of the Euclidean distance matrix for the rows in the Iris dataset, where the rows/columns in the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithms.



(a) PL values (b) ARc values (c) BAR values

Figure 5.21: Dotcharts of the PL, ARc and BAR values, relative to the best PL, ARc and BAR values respectively, for each of the permutations of the rows in the Iris dataset returned by the seriation algorithms.

Hahsler et al. (2008) and Chen (2002) also compared the performance of

seriation algorithms using heatmaps of the Euclidean distance matrix of the rows in the Iris dataset, and using the path length and anti-Robinson criteria. The results in this section agree with Hahsler et al. (2008), who concluded that ARSA is best for optimising anti-Robinson form, and TSP and OLO are best for minimising path length. Chen (2002) also concluded that TSP is best for minimising path length.

5.3.7 Laser dataset

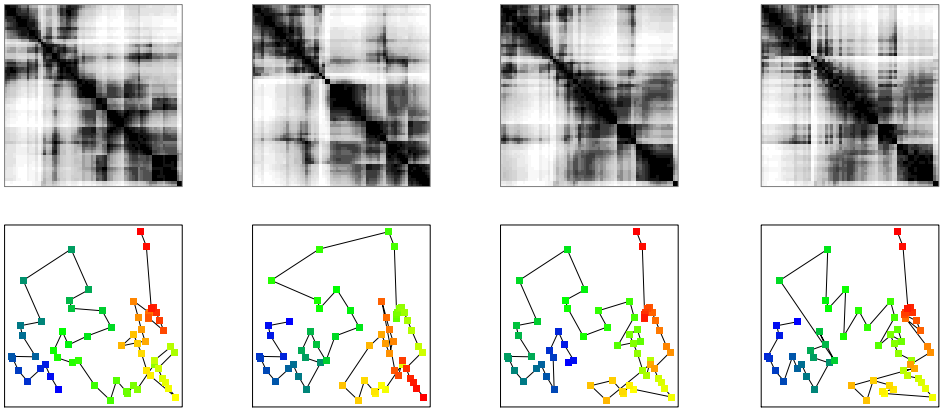
Figure 5.22 shows ten heatmaps of the Euclidean distance matrix for the rows in the Laser dataset, where the rows/columns of the heatmaps are ordered according to the permutations returned by the corresponding algorithms. Beneath each of the heatmaps is a scatterplot of the two dimensional classical MDS solution of the Euclidean distance matrix of the rows, where the path through the points corresponds to the permutations returned by the algorithms. The points in each scatterplot are coloured from blue to red according to the order in which each path traverses the points. Figure 5.23 contains dotcharts of the PL, ARc and BAR values, relative to the best PL, ARc and BAR values respectively, for each of the ten permutations.

Although PCA1, MDS1, ARSA and DendSer+ARc produce permutations with the best ARc values (Figure 5.23.(b)), their corresponding heatmaps in the second row of Figure 5.22 show lots of white squares close to the main diagonal. This "chess-board" effect makes it difficult to perceive the structure in the Euclidean distance matrix.

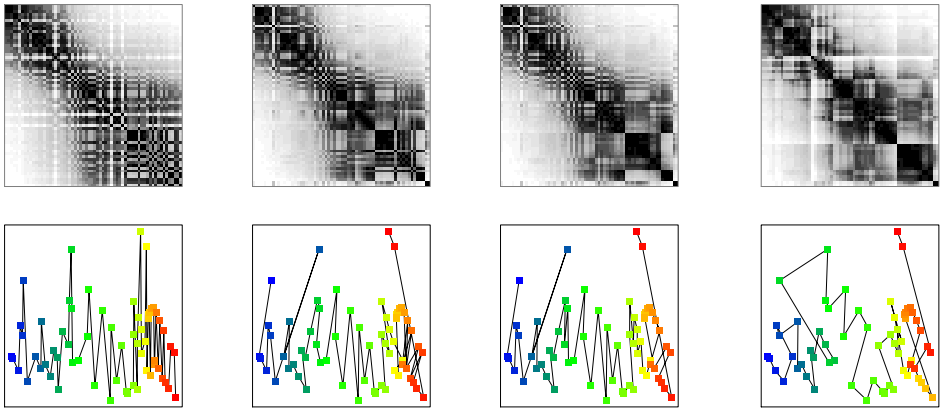
The heatmaps show this chess-board effect for the following reasons. PCA1 orders the rows according to the one dimensional classical MDS solution, i.e. the projection of the points onto the x-axis in the scatterplot in Figure 5.22.(e). MDS1 and ARSA also appear to order the rows as if they project approximately onto the x-axis in the scatterplots in Figures 5.22.(f) and (g). DendSer+ARc very roughly orders the rows as if they project onto a straight line running diagonally from the south-west to the north-east of the scatterplot in Figure 5.22.(h).

R2E and SVD2 also produce heatmaps with a chess-board effect because these two algorithms order the rows as if they project onto an ellipse, as shown in Figures 5.22.(i) and (j).

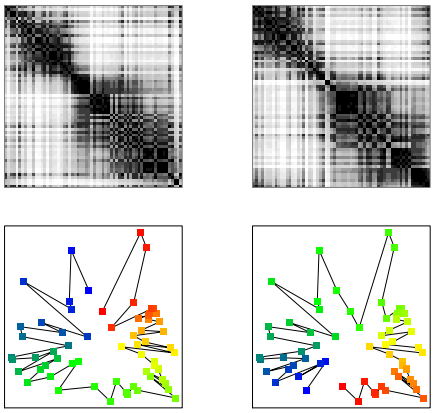
The scatterplots of the MDS solutions in the first row of Figure 5.22 show that TSP, DendSer+PL, OLO and DendSer+BAR produce permutations that "weave" through the points, which results in the more interpretable heatmaps in Figure 5.22.



(a) TSP (b) DendSer+PL (c) OLO (d) DendSer+BAR



(e) PCA1 (f) MDS1 (g) ARSA (h) DendSer+ARc

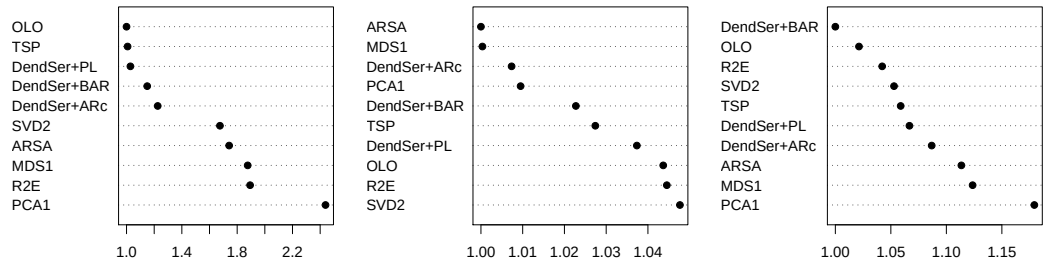


(i) R2E (j) SVD2

Figure 5.22: Seriated heatmaps of the Euclidean distance matrix for the rows in the Laser dataset, where the rows/columns in the heatmaps are ordered according to the permutations returned by the corresponding seriation algorithms. Beneath each heatmap is a scatterplot of the two dimensional classical MDS solution of the Euclidean distance matrix of the rows, where the path through the points corresponds to the permutations returned by the seriation algorithms. The points in each scatterplot are coloured from blue to red according to the order in which each path traverses the points.

In particular, DendSer+BAR and OLO produce the smoothest heatmaps and also produce permutations having the best BAR values. Their corresponding heatmaps also reveal five clusters of rows in the Laser dataset (indicated by dark blocks around the main diagonals of the heatmaps), which are not as clearly revealed by the other heatmaps. As one reads down the main diagonal of the DendSer+BAR ordered heatmap in Figure 5.22.(d), observe that

- the first cluster is quite compact and roughly corresponds to the blue points in the MDS solution.
- the second cluster is larger and roughly corresponds to the green and yellow points in the MDS solution.
- the third and fourth clusters are also quite compact and roughly correspond to the light orange and red points in the MDS solution respectively.
- the fifth cluster is very small and contains the two red points in the north-east of the scatterplot of the MDS solution.



(a) PL values (b) ARc values (c) BAR values

Figure 5.23: Dotcharts of the PL, ARc and BAR values, relative to the best PL, ARc and BAR values respectively, for each of the permutations of the rows in the Laser dataset returned by the seriation algorithms.

5.4 Summary

The following sections summarise the performance of the seriation algorithms based first on their heatmaps and second on how well each seriation algorithm minimises the PL, ARc and BAR cost functions. Section 5.4.3 also briefly discusses the efficiency of the algorithms.

5.4.1 Heatmaps

Sections 5.3.1-5.3.5 showed that most of the seriation algorithms recovered the original patterns in the rows and/or columns of the five synthetic datasets.

However, depending on the dataset, some algorithms were more successful than others, while other algorithms completely failed to recover the original pattern.

For the real datasets, Section 5.3.6 showed that all of the seriation algorithms recovered the same two block structure in the rows of the Iris dataset. However, for the Laser dataset, Section 5.3.7 showed that some algorithms resulted in much more interpretable heatmaps than other algorithms.

Table 5.1 "grades" the algorithms on how well they performed for each dataset except for the Iris dataset because all algorithms produced very similar heatmaps for this dataset.

For each of the five synthetic datasets, the algorithms are graded on how well they recovered the original patterns in the rows and/or columns: algorithms graded with "A" were very successful in recovering the original pattern, algorithms graded with "B" generally recovered the original pattern but were not as successful (i.e. their heatmaps were not as smooth) as the "A-grade" algorithms, and algorithms graded with "F" failed to recover the original pattern in the rows and/or columns.

For the Laser dataset, algorithms graded with "A" resulted in the most interpretable heatmaps, algorithms graded with "B" resulted in heatmaps that were useful but not quite as interpretable as the "A-grade" algorithms, and algorithms graded with "F" produced heatmaps that were very difficult to interpret.

Table 5.1: Summary of the performance of seriation algorithms in recovering the original patterns in the five synthetic datasets and the usefulness of their heatmaps for the Laser dataset.

Algorithm	Band	Simplex	Circumplex	Equi-cor	Block	Laser
TSP	F	F	B	F	A	B
DendSer+PL	B	B	B	B	A	B
OLO	B	B	B	B	A	A
DendSer+BAR	B	A	A	A	A	A
PCA1	A	B	F	A	F	F
MDS1	A	A	F	A	F	F
ARSA	A	A	F	A	A	F
DendSer+ARc	A	A	F	A	A	F
SVD2	A	B	A	F	A	F
R2E	B	A	A	A	A	F

DendSer+PL, OLO, DendSer+BAR and R2E are the only algorithms that recovered all of the original patterns in the synthetic datasets. However,

DendSer+BAR and OLO also produced the most useful heatmaps for the Laser dataset. Therefore, based on the heatmaps, DendSer+BAR and OLO emerge as the strongest performing seriation algorithms of those examined in this comparison study.

Wilkinson (2005, §16.5) used the five synthetic datasets to compare the performance of a small set of seriation algorithms including the Gruvaeus and Wainer (1972) method and MDS1. The performance of MDS1 in Table 5.1 coincides with Wilkinson's results.

Wilkinson also concluded that seriation methods based on cluster analysis provide no real advantage unless the user is particularly interested in clustering their data. However, this conclusion is based on the performance of the Gruvaeus and Wainer (1972) algorithm. The comparison study in this chapter showed that better dendrogram seriation algorithms such as OLO and DendSer (particularly DendSer with BAR) generally perform equally well, and in some cases better, than seriation algorithms that are not based on cluster analysis.

5.4.2 Seriation criteria

Twenty samples of each of the synthetic datasets were generated and the rows and columns in each sample were permuted using each of the seriation algorithms. For each set of rows and columns in the synthetic datasets (except for the columns in the Equi-correlation dataset), the mean PL value of the twenty permutations returned by each algorithm is recorded. For each of the Iris and Laser datasets, there is only one PL value for each of the seriation algorithms.

All of this information forms a table with eleven columns (nine for the synthetic datasets and one each for the Iris and Laser datasets) and ten rows (one for each seriation algorithm). The ij^{th} entry in the table is the mean PL value for algorithm i corresponding to data j. The columns of this table are standardised and seriated using DendSer with BAR (average linkage), where the dissimilarity between the columns is computed using 1− Pearson correlation.

Figure 5.24 shows a parallel coordinates plot (PCP), where the axes correspond to either the rows or columns in the seven datasets used in this comparison study and the lines show the standardised mean PL values for each of the seriation algorithms. The first axis in the PCP shows the overall mean PL value for each algorithm and the legend orders the algorithms according to the first axis. DendSer with BAR places data (i.e. axes in the PCP) for which the algorithms performed similarly close together in the PCP.

Figures 5.25 and 5.26 are constructed in the same way as Figure 5.24 and show the mean ARc and BAR values for each algorithm.

The parallel coordinates plot in Figure 5.24 shows that:

- TSP is the best algorithm for minimising the PL cost function.
- OLO is the best clustering based algorithm for minimising the PL cost function with DendSer with PL performing almost as well. Note that the OLO algorithm always finds the permutation from a dendrogram that minimises the PL cost function, whereas DendSer with PL is not guaranteed to find the best permutation. However, based on the results in Section 5.3, DendSer with PL produced permutations having PL values that were on average only 0.4% higher than the PL values of the permutations returned by OLO.
- The worst algorithms for minimising the PL cost function are the algorithms based on dimension reduction techniques: PCA1, MDS1 and SVD2.

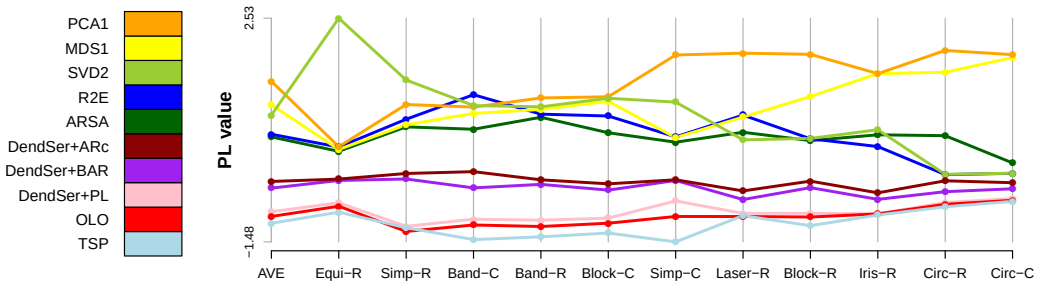


Figure 5.24: Parallel coordinates plot of the standardised mean PL values of the permutations returned by the seriation algorithms for the rows and columns in each of the seven datasets used in this comparison study. The parallel axes correspond to the data and the lines show the mean PL values for each of the algorithms. The first parallel axis shows the overall mean PL value for each of the algorithms.

The parallel coordinates plot in Figure 5.25 shows that:

- ARSA is the best algorithm for minimising the ARc cost function with MDS1 also performing quite well.
- DendSer with ARc is the best clustering based algorithm for minimising ARc.
- TSP, DendSer with PL and OLO produce the worst ARc values on average.

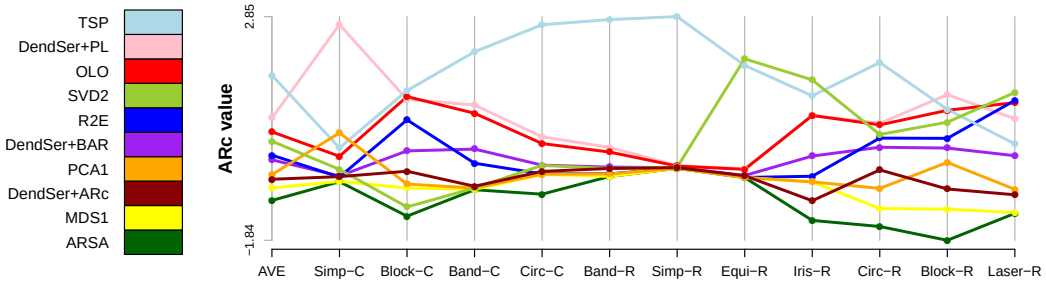


Figure 5.25: Parallel coordinates plot of the standardised mean ARc values of the permutations returned by the seriation algorithms for the rows and columns in each of the seven datasets used in this comparison study. The parallel axes correspond to the data and the lines show the mean ARc values for each of the algorithms. The first parallel axis shows the overall mean ARc value for each of the algorithms.

The parallel coordinates plot in Figure 5.26 shows that:

- DendSer with BAR is the best algorithm for minimising the BAR cost function.
- PCA1 and TSP produce the worst BAR values on average.

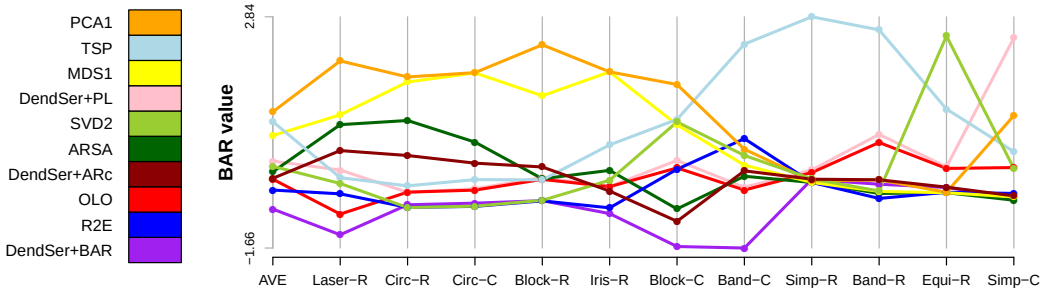


Figure 5.26: Parallel coordinates plot of the standardised mean BAR values of the permutations returned by the seriation algorithms for the rows and columns in each of the seven datasets used in this comparison study. The parallel axes correspond to the data and the lines show the mean BAR values for each of the algorithms. The first parallel axis shows the overall mean BAR value for each of the algorithms.

5.4.3 Efficiency of seriation algorithms

Another factor to consider when choosing a seriation algorithm is the complexity of the algorithm. In each of the following complexities, n is the number of objects to be seriated in an $n \times p$ matrix with $p < n$.

The TSP algorithm used in this comparison study (i.e. farthest insertion with 2-Opt) is $O(n^2) \times \#$ iterations. However, there are better algorithms for solving the TSP. See, for example, the Concorde algorithm (Applegate et al. 2000). The version of TSP algorithm used in this study was chosen because of its availability in the `R` package.

The study showed that both ARSA and MDS1 performed well in minimising the ARc cost function. Both algorithms are $O(n^2) \times \#$ iterations. In practice, ARSA is the slowest algorithm to run of those compared in this study[2]. The number of iterations for ARSA is very large (Brusco et al. 2007). MDS1 performs Kruskal's (1964a, 1964b) non-metric MDS method, implemented as `isoMDS` in `R`, which the `R` documentation reports converges in approximately 10 iterations.

For the PL cost function, the OLO algorithm performed well and is $O(n^3)$ (Bar-Joseph et al. 2001, 2003). DendSer with PL performed almost as well (see Figure 5.24) and runs in time $O(n^2) \times \#$ iterations, where the number of iterations is quite small (see Table 3.2).

DendSer with either BAR or ARc runs in time $O(n^3) \times \#$ iterations, where the number of iterations is small (see Table 3.2). The R2E algorithm also runs in time $O(n^3) \times \#$ iterations (according to Tien et al. 2008).

PCA1 and SVD2 are the fastest algorithms running in time $O(np^2)$ (see, for example, Golub and Van Loan 1996). However, these two algorithms do not always produce the most informative seriation results (see, for example, the performance of PCA1 and SVD2 for the Laser dataset in Figures 5.22.(e) and (j), the performance of PCA1 for the Circumplex dataset in Figure 5.8.(e) and the Block dataset in Figure 5.16.(e), and the performance of SVD2 for the Equi-correlation dataset in Figure 5.13.(j)).

5.5 Discussion

From an optimisation point of view, TSP, ARSA and DendSer are the best seriation algorithms for minimising the PL, ARc and BAR cost functions respectively. For other cost functions, DendSer is currently the only option.

[2]The `R` implementation of ARSA is $O(n^4)$.

Although the primary objective is to seriate data, the user may also have additional interests. For example, the user may wish to obtain a clustering of their data as well as a seriation or they may be interested in obtaining a low dimensional representation of their data. The tree in Figure 5.27 summarises the best choice of algorithm for different seriation interests:

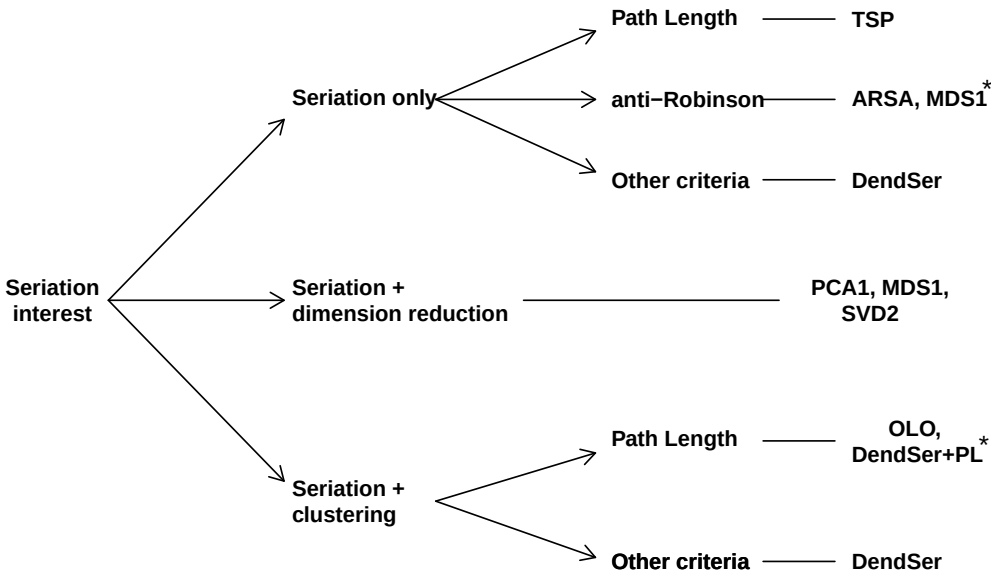


Figure 5.27: Tree illustrating the best choice of seriation algorithm for particular seriation interests. Algorithms marked with "*" indicate that they are a more efficient, although slightly less optimal, choice of algorithm.

5.5.1 Choice of seriation criterion

In two branches of Figure 5.27, the best choice of algorithm depends on the seriation criterion that the user wishes to optimise. So, before the user decides with seriation *algorithm* is best for their task, they must first decide which seriation *criterion* is best for their task.

Visualisation

One factor influencing the choice of seriation criterion is the visualisation application. For seriating parallel coordinates plots, Hurley (2004) described why path length is a suitable criterion and Section 4.2 showed that lazy path length is also useful. For seriating scatterplot matrices, Hurley (2004) described why anti-Robinson form is a suitable criterion and Section 4.4 discussed how banded anti-Robinson and lazy path length are also useful criteria.

When seriating heatmaps, a common goal is to order the objects so that the corresponding dissimilarity matrix follows anti-Robinson form (see, for example, Gale et al. 1984 and Chen 2002). However, the examples and results in Sections 4.5, 4.6 and 5.3 suggest that banded anti-Robinson form is a more suitable choice of criterion when seriating heatmaps:

- For the Morse code and Fibroblast examples (Sections 4.5 and 4.6), minimising ARc using DendSer produced poor seriation results because the corresponding dissimilarity matrices followed a circumplex pattern. However, DendSer with BAR produced more informative seriation results for these two datasets.

- Optimising ARc using ARSA or DendSer also performed poorly for the Laser dataset in Section 5.3.7 by producing a chess-board effect in the heatmap of the Euclidean distance matrix (see Figures 5.22.(g) and (h)), which made it difficult to discern the structure in the data. However, minimising BAR using DendSer resulted in a smoother, more interpretable heatmap of the Euclidean distance matrix (Figure 5.22.(d)).

- The results in Sections 5.3.1-5.3.5 show that minimising BAR using DendSer recovered the original pattern in all of the synthetic datasets, whereas optimising ARc using ARSA or DendSer failed to recover the original pattern in the Circumplex dataset.

Flexibility

The Morse code, Fibroblast and Laser examples show that seriating using anti-Robinson form is not always appropriate. Also, determining whether or not anti-Robinson form is suitable, prior to seriation, is generally difficult (although circumplex patterns may be revealed using dimension reduction techniques). Therefore, seriating using anti-Robinson form may be "risky".

The strong performance of DendSer with BAR for the datasets listed above show that banded anti-Robinson form is suitable for a wider range of data patterns than anti-Robinson form. Therefore, banded anti-Robinson form is a "safer" and more flexible alternative to anti-Robinson form.

Global versus local

Path length is another possible alternative to anti-Robinson form. However, just as optimising anti-Robinson form enforces too *global* a structure on a dissimilarity matrix, optimising path length looks for too *local* a structure.

Path length is only concerned with structure just off the main diagonal in the dissimilarity matrix and so generally reveals local structure in data but may not reveal more global patterns. See, for example, the performance of the TSP algorithm for the Band and Equi-correlation datasets in Sections 5.3.1 and 5.3.4.

Banded anti-Robinson form provides a compromise between the two extremes of path length and anti-Robinson form. For example, see the Laser and Fibroblast datasets in Sections 3.4.2 and 4.6, where optimising banded anti-Robinson form reveals more global trends that optimising path length but does not impose as rigid a structure as when optimising anti-Robinson form.

These arguments all point to banded anti-Robinson form being an all-round useful seriation criterion and so if the user chooses this criterion, then DendSer is currently the only suitable seriation algorithm.

Chapter 6

Concluding remarks

This chapter summarises and discusses the outcomes of the research described in this thesis.

DendSer is a new flexible dendrogram seriation algorithm that allows the user to choose from many seriation criteria and also to input their own criteria. This is in contrast to other seriation algorithms, which generally focus on one criterion only. DendSer also provides a general framework for performing several other dendrogram seriation algorithms such as those described in Gruvaeus and Wainer (1972), Degerman (1982) and Wishart (1999).

The choice of seriation criteria for DendSer includes two new criteria called banded anti-Robinson form and lazy path length, both of which have applications to a variety of visualisation settings, as shown in the examples in Chapter 4. The motivation for the lazy path length criterion came from Hurley (2004), who discussed the concept of placing interesting features in prominent positions in statistical graphics such as scatterplot matrices and parallel coordinates plots.

The original motivation for developing banded anti-Robinson form was the poor performance of anti-Robinson form when seriating data that follow a circumplex pattern. However, further investigation into banded anti-Robinson form revealed that it is a very flexible seriation criterion and often results in more meaningful visualisations than the more standard path length and anti-Robinson criteria.

Other seriation criteria could also be developed. For example, a hybrid of the lazy path length and banded anti-Robinson criteria could be developed, which would aim to position the smallest values in a dissimilarity matrix in the north east region of the matrix. This would extend the lazy path length criterion, which aims to position the smallest values close to the beginning of the main diagonal only. Such a criterion could have visualisation applications

in seriating heatmaps and scatterplot matrices.

DendSer also gives the user a choice of how to rearrange the nodes in a dendrogram, which is an important, yet generally ignored, feature of dendrogram seriation. This thesis developed notation and terminology for describing how to rearrange nodes in a dendrogram and defined several different node operations, which rearrange or operate on nodes in different ways. Some of these node operations are used in existing dendrogram seriation algorithms, while others are newly defined in this thesis.

This thesis also gives guidelines for the appropriate use of node operations in DendSer. The choice of node operation is important because different node operations are suitable for different seriation criteria and so using an inappropriate node operation may have a negative effect on the seriation results returned by DendSer. The results of comparing the different node operations showed that DendSer, when used in conjunction with one of two new node operations, performed better than when used in conjunction with one of the node operations currently used in other dendrogram seriation algorithms.

The flexibility of DendSer is highlighted in Chapter 4, which described several examples encompassing a variety of datasets, visualisations, dissimilarity measures and seriation criteria. Other visualisation applications could also be explored. For example, in correspondence, Al Inselberg discussed his idea of ordering triples in parallel coordinates plots, which relates to the problem of reading information between triples of variables in a parallel coordinates plot. This problem suggests that when seriating the variables in a parallel coordinates plot, the dissimilarity between variables that are up to two spaces apart should be considered and not just the dissimilarity between adjacent variables. Therefore, using DendSer to optimise the banded anti-Robinson criterion with a band-width of two appears to be an appropriate seriation method for Inselberg's "triple ordering" problem.

This thesis explored many different aspects of DendSer but some issues remain to be investigated. For example, all uses of DendSer to date suggest that DendSer will always converge. However, this property requires further study. The choice of linkage is another issue that requires further inspection. Different linkages affect the results of hierarchical clustering and so they also affect the results of dendrogram seriation algorithms. The extent of this effect has yet to be investigated. The choice of dissimilarity measure also requires further research.

Although this thesis approached the task of developing flexible seriation tools by using hierarchical clustering based methods, there are other possible approaches. For example, simulated annealing algorithms such as ARSA

(Brusco et al. 2007) could be modified to also give the user a choice of seriation criteria. However, one disadvantage of this particular approach is that simulated annealing based seriation algorithms generally perform more slowly than dendrogram seriation algorithms. Dendrogram seriation also seems more appropriate for data visualisation applications because it marries together two data analytic techniques: clustering and seriation.

As a final remark, the significance of the research presented in this thesis is that it provides practical seriation tools for enhancing data visualisation, which will be made freely available to the worldwide statistical community via a software package in the statistical program `R` (R Development Core Team 2010).

Appendix A

Proofs of properties of node operations

The following is a list of properties of the node operations defined in Sections 3.3.1 and 3.3.2. In each of the following, N is a node in a dendrogram Δ.

Property 1. The order in which R_0 operates on two nodes N_a and N_b does not affect the returned permutation, i.e.

$$R_0(N_b; R_0(N_a; \Delta)) = R_0(N_a; R_0(N_b; \Delta)). \tag{A.1}$$

Equation A.1 may be written using the following simpler notation:

$$R_0(N_a, N_b; \Delta) = R_0(N_b, N_a; \Delta). \tag{A.2}$$

Equation A.2 extends to any number of nodes.

Proof of Property 1. The result in Equation A.2 is obvious if N_a and N_b are disjoint nodes in Δ. Showing that this result holds for the case when N_a and N_b are nested nodes relies on the fact that reflecting a node N reverses the order of the leaves in N and all descendant sub-nodes of N. Consider the following.

Let N_a be a node nested in N_b and without loss of generality, let N_b be the root node of the dendrogram Δ (see Figure A.1 for an example). Let $L(N_a)$ be the ordered leaves in N_a and let X and Y be the ordered leaves on the left and right of N_a respectively.

$R_0(N_a, N_b; \Delta)$ first reflects N_a and then N_b, changing the original permutation of leaves from

$$X, L(N_a), Y \longrightarrow X, \overline{L(N_a)}, Y \longrightarrow \overline{Y}, L(N_a), \overline{X},$$

where $\overline{X}$ indicates the reverse of X and similarly for $\overline{L(N_a)}$ and $\overline{Y}$.

$R_0(N_b, N_a; \Delta)$ first reflects N_b and then N_a, changing the original permutation of leaves from

$$X, L(N_a), Y \longrightarrow \overline{Y}, \overline{L(N_a)}, \overline{X} \longrightarrow \overline{Y}, L(N_a), \overline{X}.$$

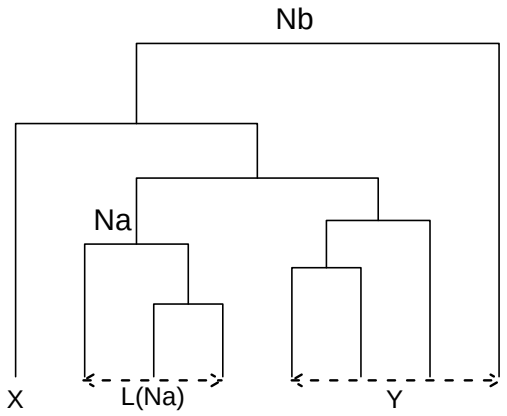


Figure A.1: An arbitrary dendrogram with the node N_a nested within the root node N_b. X and Y represent the ordered leaves to the left and right of N_a, and $L(N_a)$ represents the ordered leaves in N_a.

$R_0(N_a, N_b; \Delta) = R_0(N_b, N_a; \Delta)$ because, regardless of the order in which N_a and N_b are reflected, the leaves in N_b are reversed once and the leaves in N_a are reversed twice. Take three nested nodes, N_a nested in N_b nested in N_c. Then $R_0(N_a, N_b, N_c; \Delta)$ reverses the leaves in N_c once, the leaves in N_b twice and the leaves in N_a three times, regardless of the order in which N_a, N_b and N_c are reflected. Therefore, $R_0(N_a, N_b, N_c; \Delta) = R_0$(any permutation of $N_a, N_b, N_c; \Delta$). This argument extends to any number of nested nodes. □

Property 2. The order in which T_0 operates on two nodes N_a and N_b does not affect the returned permutation, i.e.

$$T_0(N_b; T_0(N_a; \Delta)) = T_0(N_a; T_0(N_b; \Delta)). \tag{A.3}$$

Equations A.3 may be written using the following simpler notation:

$$T_0(N_a, N_b; \Delta) = T_0(N_b, N_a; \Delta). \tag{A.4}$$

Equation A.4 extends to any number of nodes.

Proof of Property 2. The result in Equation A.4 is obvious if N_a and N_b are disjoint nodes in Δ. Showing that this result holds for the case when N_a and

N_b are nested nodes relies on the fact that translating a node N does not affect the order of the leaves in the descendant sub-nodes of N.

Let N_a be a nested node in a node N_b (see, for example, Figure A.1) and so translating N_b does not affect the order of the leaves in N_a. Therefore, $T_0(N_a, N_b; \Delta) = T_0(N_b, N_a; \Delta)$ because, regardless of the order in which N_a and N_b are translated, the left and right sub-nodes of N_a and N_b are swapped once and only once. This argument extends to any number of nested nodes.

□

Property 3. The following relationships hold between R_0 and T_0:

$$T_0(N; \Delta) = R_0(N, N_l, N_r; \Delta), \tag{A.5}$$

where N_l and N_r are the left and right sub-nodes of N.

$$R_0(N; \Delta) = T_0(N_a, \ldots, N_k, N; \Delta), \tag{A.6}$$

where $N_a, \ldots, N_k$ are all of the descendant sub-nodes of N.

Proof of Property 3. Let N_l and N_r be the left and right sub-nodes of N. Reflecting N swaps the positions of N_l and N_r but also reverses the order of the leaves in N_l and N_r and so reflecting N_l and N_r after reflecting N puts the leaves in N_l and N_r back in their original order. Therefore, reflecting N, N_r and N_l corresponds to the translation of N and so Equation A.5 is true.

Let N_3 be a node in a dendrogram Δ_1 and let N_1 and N_2 be the left and right sub-nodes of N_3 respectively. Assume that N_1 and N_2 are the only sub-nodes of N_3, i.e. N_1 and N_2 have at most two leaves each. See, for example, the dendrogram Δ_1 in Figure A.2.(a). Let $L(N_i)$ be the ordered leaves in the node N_i and let $\overline{L(N_i)}$ denote the reverse of the leaves in N_i.

$T_0(N_1, N_2, N_3; \Delta_1)$ translates N_1, then N_2 and then N_3 and so changes the original permutation of the leaves from:

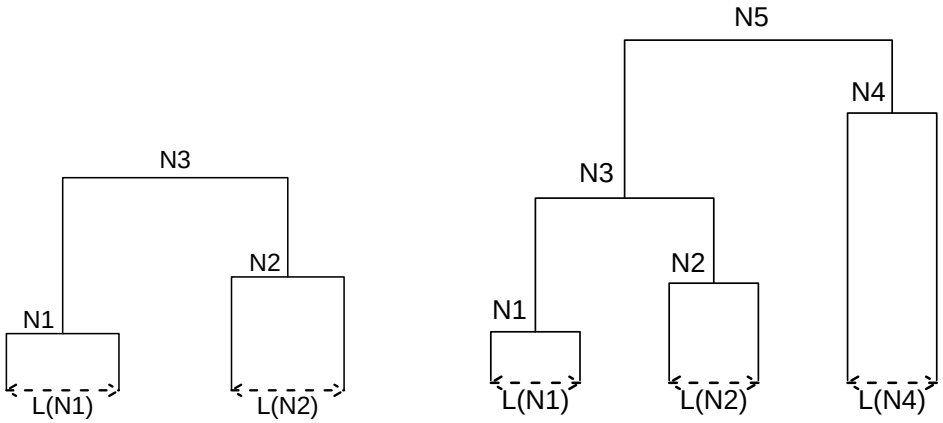
$$L(N_1)L(N_2) \longrightarrow \overline{L(N_1)}L(N_2) \longrightarrow \overline{L(N_1)}\ \overline{L(N_2)} \longrightarrow \overline{L(N_2)}\ \overline{L(N_1)}.$$

However, $\overline{L(N_2)}\ \overline{L(N_1)} = \overline{L(N_3)}$ and so $T_0(N_1, N_2, N_3; \Delta_1) = R_0(N_3; \Delta_1)$.

Now extend Δ_1 in Figure A.2.(a) so that N_3 and also a node N_4 are the left and right sub-nodes of a node N_5. For ease of demonstration, assume N_4 has at most two leaves. See, for example, the dendrogram Δ_2 in Figure A.2.(b).

The above argument shows that $T_0(N_1, N_2, N_3; \Delta_2) = \overline{L(N_3)}L(N_4)$ and so $T_0(N_1, N_2, N_3, N_4, N_5; \Delta_2)$ changes the permutation of leaves from

$$\overline{L(N_3)}L(N_4) \longrightarrow \overline{L(N_3)}\ \overline{L(N_4)} \longrightarrow \overline{L(N_4)}\ \overline{L(N_3)}.$$



(a) Dendrogram Δ_1 (b) Dendrogram Δ_2

Figure A.2: The dendrogram Δ_1 in Figure (a) visualises a hierarchical clustering of four randomly generated data objects. The dendrogram Δ_2 in Figure (b) extends Δ_1 so that N_3 is now a sub-node of a node N_5. $L(N_i)$ denotes the ordered leaves in node N_i.

However, $\overline{L(N_4)}\ \overline{L(N_3)} = \overline{L(N_5)}$ and so $T_0(N_1, \ldots, N_5; \Delta_2) = R_0(N_5; \Delta_2)$.

This argument extends recursively to any number of nested nodes and so Equation A.6 is true. □

Property 4. The sets of permutations returned by some node operations are subsets of the sets of permutations returned by other node operations:

(a) $T_0(N; \Delta) \subseteq R_{01}(N; \Delta)$, follows from Equation A.5.

(b) $R_0(N; \Delta) \subseteq R_{01}(N; \Delta)$, by definition of R_{01} in Equation 3.5.

(c) $R_1(N; \Delta) \subseteq R_{01}(N; \Delta)$, by definition of R_{01} in Equation 3.5.

(d) $T_0(N; \Delta) \subseteq T_{01}(N; \Delta)$, by definition of T_{01} in Equation 3.6.

(e) $T_1(N; \Delta) \subseteq T_{01}(N; \Delta)$, by definition of T_{01} in Equation 3.6.

(f) $C_0(N; \Delta) \subseteq R_{01}(N; \Delta)$, by Properties 4.(a) and 4.(b).

(g) $R_0(N; \Delta) \subseteq C_0(N; \Delta)$, by definition of C_0 in Equation 3.7.

(h) $T_0(N; \Delta) \subseteq C_0(N; \Delta)$, by definition of C_0 in Equation 3.7.

Property 5. The node operations R_0 and R_1 are commutative, i.e.

$$R_1(N; R_0(N; \Delta)) = R_0(N; R_1(N; \Delta)). \tag{A.7}$$

Proof of Property 5. If N_l and N_r are the left and right sub-nodes of a node N in a dendrogram Δ, then by definition

$$R_1(N; \Delta) = R_0(N_l; \Delta) \cup R_0(N_r; \Delta) \cup R_0(N_l, N_r; \Delta). \tag{A.8}$$

Therefore, $R_1(N; R_0(N; \Delta))$ can be written as:

$$\begin{aligned} R_1(N; R_0(N; \Delta)) &= R_0(N_l; R_0(N; \Delta)) \cup R_0(N_r; R_0(N; \Delta)) \\ &\quad \cup R_0(N_l, N_r; R_0(N; \Delta)). \end{aligned} \tag{A.9}$$

Equation A.9 can be written using the following simpler notation:

$$R_1(N; R_0(N; \Delta)) = R_0(N, N_l; \Delta) \cup R_0(N, N_r; \Delta) \cup R_0(N, N_l, N_r; \Delta). \tag{A.10}$$

Note that $R_0(N, N_l; \Delta)$ first reflects N and then reflects N_l. Similarly, $R_0(N, N_r; \Delta)$ reflects N first and then N_r and $R_0(N, N_l, N_r; \Delta)$ reflects N first, then N_l and then N_r. However, Property 1 says that the order in which R_0 operates on a set of nodes does not affect the returned result. Therefore, reflecting N *before* applying R_1 to N (i.e. $R_1(N; R_0(N; \Delta))$) produces the same set of permutations as reflecting N *after* applying R_1 to N (i.e $R_0(N; R_1(N; \Delta))$) and so Equation A.7 is true. (Note that $R_0(N; R_1(N; \Delta))$ reflects N in each of the dendrograms represented by $R_0(N_l; \Delta)$, $R_0(N_r; \Delta)$ and $R_0(N_l, N_r; \Delta)$.) □

Property 6. The node operations T_0 and T_1 are commutative, i.e.

$$T_1(N; T_0(N; \Delta)) = T_0(N; T_1(N; \Delta)). \tag{A.11}$$

Proof of Property 6. If N_l and N_r are the left and right sub-nodes of a node N in a dendrogram Δ, then by definition

$$T_1(N; \Delta) = T_0(N_l; \Delta) \cup T_0(N_r; \Delta) \cup T_0(N_l, N_r; \Delta). \tag{A.12}$$

Therefore, $T_1(N; T_0(N; \Delta))$ can be written as:

$$\begin{aligned} T_1(N; T_0(N; \Delta)) &= T_0(N_l; T_0(N; \Delta)) \cup T_0(N_r; T_0(N; \Delta)) \\ &\quad \cup T_0(N_l, N_r; T_0(N; \Delta)). \end{aligned} \tag{A.13}$$

Equation A.13 can be written using the following simpler notation:

$$T_1(N; T_0(N; \Delta)) = T_0(N, N_l; \Delta) \cup T_0(N, N_r; \Delta) \cup T_0(N, N_l, N_r; \Delta). \tag{A.14}$$

Note that $T_0(N, N_l; \Delta)$ first translates N and then translates N_l. Similarly, $T_0(N, N_r; \Delta)$ translates N first and then N_r and $T_0(N, N_l, N_r; \Delta)$ translates N first, then N_l and then N_r. However, Property 2 says that the order in

which T_0 operates on a set of nodes does not affect the returned result. Therefore, translating N *before* applying T_1 to N (i.e. $T_1(N; T_0(N; \Delta))$) produces the same set of permutations as translating N *after* applying T_1 to N (i.e $T_0(N; T_1(N; \Delta))$) and so Equation A.11 is true. (Note that $T_0(N; T_1(N; \Delta))$ translates N in each of the dendrograms represented by $T_0(N_l; \Delta)$, $T_0(N_r; \Delta)$ and $T_0(N_l, N_r; \Delta)$.) □

Appendix B

Synthetic datasets

The following formulae for the Simplex, Band, Circumplex, Equi-correlation and Block datasets are the same, or slightly modified versions of, the formulae used in Wilkinson (2005, §16.5). Wilkinson (2005, §16.5) did not report the values he used for the various parameters in the following formulae. Therefore, the parameter values used in this thesis are chosen so that the heatmaps of the resulting data resemble the heatmaps in Wilkinson (2005, §16.5) as closely as possible. After generating the values, the columns in each dataset are standardised.

B.1 Simplex dataset

The Simplex dataset is generated using the following formula:

$$x_{ij} = \frac{e^{t_{ij}}}{1 + e^{t_{ij}}} + u_{ij},\ i = 1, \dots n,\ j = 1, \dots, p, \tag{B.1}$$

where

$$\begin{aligned} t_{ij} &= \frac{s_j - r_i}{b}, \\ s_j &= \frac{j}{p}, \\ r_i &= \frac{i}{n}. \end{aligned}$$

The random error u_{ij} is a weighted standard normal random variable Z:

$$u_{ij} \sim N(0, k). \tag{B.2}$$

The parameter k determines the amount of random error in the data and the positive non-zero parameter b determines the slope of the logistic function

generated by the exponentials, i.e. the sharpness of the boundary between the yellow and red regions in the top heatmap in Figure 5.2.(b). This thesis uses $b = 0.1$ and $k = 0.1$. In the Simplex dataset, the correlation between all pairs of columns is positive with the correlation between near columns higher than the correlation between distant columns.

Note that Wilkinson (2005, §16.5) used $u_{i,j} \sim N(0, ke^{-t_{i,j}^2})$ to create the random error in the formula for the Simplex data. However, the formula for $u_{i,j}$ in Equation B.2 results in data that appear more similar to the heatmaps shown in Wilkinson (2005, §16.5).

B.2 Band dataset

The Band dataset is generated using the following formula:

$$x_{ij} = e^{-t_{ij}^2} + u_{ij},\ i = 1, \ldots n,\ j = 1, \ldots, p, \tag{B.3}$$

where

$$\begin{aligned} t_{ij} &= \frac{s_j - r_i}{b}, \\ s_j &= \frac{j}{p}, \\ r_i &= \frac{i}{n}. \end{aligned}$$

The random error u_{ij} is a weighted standard normal random variable Z:

$$u_{ij} \sim N(0, ke^{-t_{ij}^2}). \tag{B.4}$$

This thesis uses $b = 1.05$ and $k = 0.15$. With the Band dataset, correlations between near columns are positive and correlations between distant columns are negative.

B.3 Circumplex dataset

The Circumplex dataset is generated using the following formula:

$$x_{ij} = e^{-t_{ij}^2} + u_{ij},\ i = 1, \ldots n,\ j = 1, \ldots, p, \tag{B.5}$$

where

$$t_{ij} = \cos(\pi(s_j - r_i)),$$

$$
\begin{aligned}
s_j &= \frac{j}{p}, \\
r_i &= \frac{i}{n}.
\end{aligned}
$$

The random error u_{ij} is a weighted standard normal random variable Z:

$$u_{ij} \sim N(0, ke^{-t_{ij}^2}). \tag{B.6}$$

This thesis uses $k = 0.3$. In the Circumplex dataset, correlations between near columns are positive and correlations between distant columns are negative. However, for the Circumplex dataset distance is measured on the circumference of a circle and so the first few columns are positively correlated with the last few columns.

B.4 Equi-correlation dataset

The Equi-correlation dataset is generated using the following formula:

$$x_{ij} = t_i + u_{ij},\ i = 1, \ldots n,\ j = 1, \ldots, p, \tag{B.7}$$

where

$$
\begin{aligned}
t_i &= \frac{bi}{n}, \\
u_{ij} &\sim N(0, 1).
\end{aligned}
$$

The parameter b determines the strength of the correlation between the columns in the Equi-correlation dataset. This thesis uses $b = 5$, which results in the correlations between all pairs of columns being quite large and positive.

B.5 Block dataset

The Block dataset is generated so that the columns divide into two blocks and the rows divide into four blocks. In order to create this structure, the data matrix is divided into eight sections, where the values in these sections are generated as follows:

$$
\begin{pmatrix}
N(0,1) & N(0,1) \\
N(0,1) & N(2,1) \\
N(2,1) & N(0,1) \\
N(2,1) & N(2,1)
\end{pmatrix}
\tag{B.8}
$$

In the Block dataset, correlations between columns from the same block are near one and correlations between columns from different blocks are near zero. Note that Wilkinson (2005, §16.5) is unclear as to how he generates the Block dataset and so the above method is possibly different than the method he uses.

Bibliography

Allison, T. & Cicchetti, D. (1976), "Sleep in Mammals: Ecological and Constitutional Correlates", *Science*, 194, 732-734.

Alon, U., Barkai, N., Notterman, D.A., Gish, K., Ybarra, S., Mack, D. & Levine, A.J. (1999), "Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays", *Proc. Natl. Acad. Sci. USA*, 96, 6745-6750.

Applegate D., Bixby R.E., Chvátal V., Cook W. (2000), "TSP Cuts Which Do Not Conform to the Template Paradigm". In *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions*, edited by Junger, M. & Naddef, D., Springer-Verlag, London, UK.

Arabie, P. & Hubert, L.J. (1990), "The bond energy algorithm revisited", IEEE Transactions on Systems, Man, and Cybernetics, 20(1), 268-274.

Atkins, J.E., Boman E.G. & Hendrickson B. (1998) "A spectral algorithm for seriation and the consecutive ones problem", *SIAM J. Computing*, 28(1), 297-310.

Bar-Joseph, Z., Gifford, D.K. & Jaakkola, T.S. (2001), "Fast optimal leaf ordering for hierarchical clustering", *Bioinformatics*, 17(1), 22-29.

Bar-Joseph, Z., Demaine, E.D., Gifford, D.K., Srebro, N., Hamel, A.M. & Jaakkola, T.S. (2003), "K-ary clustering with optimal leaf ordering for gene expression data", *Bioinformatics*, 19(9), 1070-1078.

Bertin, J. (1983), *Semiology of Graphics: Diagrams, Networks, Maps*, translated by W.J. Berg, University of Wisconsin Press.

Brewer, C.A. (1994), "Color Use Guidelines for Mapping and Visualization", Chapter 7 in *Visualization in Modern Cartography*, edited by MacEachren, A.M. & Taylor, D.R.F., Elesvier Science, Tarrytown, NY.

Brusco, M.J., Kohn, H. & Stahl, S. (2007), "Heuristic implementation of dynamic programming for matrix permutation problems in combinatorial data analysis", *Psychometrika*, 73(3), 503-522.

Brusco, M., and Stahl, S. (2005), *Branch-and-Bound applications in combinatorial data analysis*, New York, Springer.

Caraux, G. & Pinloche, S. (2005), "Permutmatrix: A graphical environment to arrange gene expression profiles in optimal linear order", *Bioinformatics*, 21(7), 1280-1281.

Carroll, J.D. & Arabie, P. (1980), "Multidimensional scaling", *Annual Review of Psychology*, 31, 607-649.

Chambers, J.M., Cleveland, W.S., Kleiner, B. & Tukey, P.A. (1983), *Graphical methods for Data analysis*, Belmont, CA: Wadsworth.

Chen, C-H., (2002), "Generalized association plots: information visualization via iteratively generated correlation matrices", *Statistica Sinica*, 12, 7-29.

Cleveland, W.S., (1985), *The Elements of Graphing Data*, Monterey, CA: Wadsworth.

Cleveland, W.S., (1993), *Visualizing data*, Hobart Press, Summit, New Jersey.

Cook, D. & Wickham, H. (2010), "tourr: Implement tour methods in pure R code", R package version 0.1.

Cox T.F & Cox M.A.A., (1994), *Multidimensional scaling*, Chapman and Hall.

Croes, G.A. (1958), "A method for solving traveling-salesman problems", *Operations Research*, 6(6), 791-812.

Degerman, R. (1982), "Ordered Binary Trees constructed through an application of Kendall's Tau", *Psychometrika*, 47, 523-527.

Der, G. & Everitt, B.S., (2001), *A Handbook of Statistical Analyses using SAS (2nd Ed.)*, Chapman & Hall, Boca Raton, FL.

Eisen, M.B., Spellman P.T., Brown, P.O., & Botstein, D. (1998), "Cluster analysis and display of genome-wide expression patterns". *Proceedings of the National Academy of Science of the United States*, 95(25), 14863-14868.

Everitt, B.S. & Dunn, G. (2001), *Applied Multivariate Data Analysis (2nd Ed.)*, Arnold, London.

Fisher, R.A. (1936), "The Use of Multiple Measurements in Taxonomic Problems", *Annals of Eugenics*, 7, 179-188.

Forina, M., Lanteri, S., Casale, M. & Concepción Cerrato Oliveros, M. (2007), "A new algorithm for seriation and its use in similarity dendrograms", *Chemometrics and Intelligent Laboratory Systems*, 87(2), 262-274.

Fraley, C. and Raftery, A.E. (1999), "MCLUST: Software for Model-Based Cluster Analysis", *Journal of Classification*, 16, 297-306.

Fraley, C. and Raftery, A.E. (2002), "Model-Based Clustering, Discriminant Analysis, and Density Estimation", *Journal of the American Statistical Association*, 97, 611-631.

Friendly, M. (2002), "Corrgrams: Exploratory Displays for Correlation Matrices", *The American Statistician*, 56, 316-324.

Friendly, M. (1994), "Mosaic displays for multi-way contingency tables", *Journal of the American Statistical Association*, 89, 190-200.

Friendly, F. & Kwan, E. (2003), "Effect ordering for data displays", *Computational Statistics & Data Analysis*, 43, 509-539.

Gale, N., Halperin, W.C. & Costanzo, C.M. (1984), "Unclassed matrix shading and optimal ordering in hierarchical cluster analysis", *Journal of Classification*, 1, 75-92.

Garey, M.R. & Johnson, D.S. (1979), *Computers and intractability: A guide to the theory of NP-completeness*, San Francisco: Freeman & Co.

Garfinkel, R.S. (1985), *Motivation and Modeling*, Wiley, New York..

Golub, G.H. & Van Loan, C.F. (1996), *Matrix Computations, (3rd Ed.)*, The John Hopkins University Press, Baltimore and London.

Gruvaeus, G. & Wainer, H. (1972), "Two additions to hierarchical cluster analysis", *British Journal of Mathematical and Statistical Psychology*, 25, 200-206.

Gutin, G. & Punnen A.P. (2002), *The Travelling Salesman Problem and its Variations*, Kluwer Academic Publishers.

Hahsler, M. & Hornik, K. (2009), "TSP: Traveling Salesperson Problem (TSP).", R package version 1.0-0.

Hahsler, M. & Hornik, K. (2007), "TSP - Infrastructure for the traveling salesperson problem", *Journal of Statistical Software*, 23(2), 1-21.

Hahsler, M., Hornik, K. & Buchta, C. (2010), "seriation: Infrastructure for seriation". R package version 1.0-2.

Hahsler, M., Hornik, K. & Buchta, C. (2008), "Getting things in order: an introduction to the R package seriation", *Journal of Statistical Software*, 25(3), 1-34.

Hastie, T., Tibshirani, R., & Friedman, J., (2009), *The Elements of Statistical Learning (2nd Ed.)*, Springer Series in Statistics.

Hubert, L.J. (1974), "Some applications of graph theory and related non-metric techniques to problems of approximate seriation: The case of symmetric proximity measures", *British Journal of Mathematical and Statistical Psychology*, 27, 133-153.

Hubert, L., Arabie, P. & Meulman, J. (2006), *The structural representation of proximity matrices with MATLAB*, Philadelphia: SIAM.

Hubert, L., Arabie, P. & Meulman J. (2001), *Combinatorial Data Analysis: Optimization by Dynamic Programming*, Philadelphia: SIAM.

Hurley, C. (2004), "Clustering visualizations of multidimensional data", *Journal of Computational & Graphical Statistics*, 13(4), 788-806.

Iyer, V.R., Eisen, M.B, Ross, D.T., Schuler, G., Moore, T., Lee, J.C.F, Trent, J.M., Staudt, L.M., Hudson, J., Boguski, M.S., Lashkari, D., Shalon, D., Botstein, D. & Brown, P.O. (1999), "The transcriptional program in the response of human fibroblasts to serum", *Science*, 283, 83-87.

Johnson, D.S. and Papadimitriou, C.H. (1985), "Performance Guarantees for Heuristics", in *The Traveling Salesman Problem*, edited by Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B., 145-180. John-Wiley & Sons, Ltd.

Khan, J., Wei, J.S., Ringnér, M., Saal, L.H., Ladanyi, M., Westermann, F., Berthold, F., Schwab, M., Antonescu, C.R., Peterson, C. & Meltzer, P.S. (2001), "Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks", *Nature Medicine*, 7(6), 673-679.

Kendall, D.G. (1971), "Seriation from abundance matrices", in *Mathematics in the Archaeological and Historical Sciences*, edited by Hodson, F.R, Kendall, D.G., and Tautu, P., 215-252. Edinburgh: Edinburgh University Press.

Kohonen, T. (1984), *Self-Organization and Associative Memory*, Springer, Berlin.

Kruskal, J.B. (1964a), "Multidimensional scaling by optimizing goodness-of-fit to a nonmetric hypothesis", *Psychometrika*, 29, 1-27.

Kruskal, J.B. (1964b), "Nonmetric multidimensional scaling: a numerical method", *Psychometrika*, 29, 115-129.

Kruskal, J.B., & Landwehr, J.M., (1983), "Icicle Plots: Better Displays for Hierarchical Clustering", *The American Statistician*, 37(2), 162-168

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. (1985), *The Traveling Salesman Problem.* John-Wiley & Sons, Ltd.

Lin, S. & Kernighan, B.W. (1973), "An Effective Heuristic Algorithm for the Traveling-Salesman Problem", *Operations Research*, 21, 498-516.

McCormick, W.T., Schweitzer, P.J., & White, T.W. (1972), "Problem decomposition and data reorganization by a clustering technique", *Operations Research*, 20(5), 993-1009.

Morris, S.A., Asnake, B. & Yen, G.G. (2003), "Optimal dendrogram seriation using simulated annealing", *Information Visualisation*, 2, 95-104.

Niermann, S. (2005), "Optimizing the ordering of tables with evolutionary computation", *The American Statistician*, 59(1), 41-46.

Peng, W., Ward, M.O. & Rundensteiner, E.A. (2004), "Clutter reduction in multi-dimensional data visualization using dimension reordering", In *Proceedings of the IEEE Information Visualization 2004*, 89-96.

Petrie, W.M. (1899), "Sequences in prehistoric remains", *Journal of the Anthropological Institute*, 29, 295-301.

R Development Core Team (2010), R: A language and environment for statistical computing, R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

Robinson, W.S. (1951), "A method for chronologically ordering archaeological deposits", *American Antiquity*, 16, 293-301.

Rothkopf, E.Z. (1957), "A measure of stimulus similarity and errors in some paired associate learning tasks", *Journal of Experimental Psychology*, 53, 94-101.

Spellman, P.T., Sherlock, G., Zhang, M.Q., Iyer, V.R., Anders, K., Eisen, M.B., Brown, P.O., Botstein, D. & Futcher, B. (1998), "Comprehensive identification of cell cycle-regulated genes in the yeast *Saccharomyces cerevisiae* by microarray hybridization", *Molecular Biology of the Cell*, 9, 3273-3297.

Tien, Y-J., Lee, Y-S., Wu, H-M. & Chen, C-H. (2008), "Methods for simultaneously identifying coherent local clusters with smooth global patterns in gene expression profiles", *BMC Bioinformatics*, 9(155).

Tufte, E.R. (2001), *The Visual Display of Quantitative Information (2nd Ed.)*, Graphics Press, Cheshire, CT.

Tubb, A., Parker, A.J. & Nickless, G. (1980), "The analysis of Romano-British pottery by atomic absorption spectrophotometry", *Archaeometry*, 22(2), 153-171.

Unwin A., Theus, M. & Hofmann, H. (2006), *Graphics of Large Datasets: Visualizing a Million (Statistics and Computing)*, Springer.

Ward, M.O. (2002), "A taxonomy of glyph placement strategies for multidimensional data visualization", *Information Visualization*, 1, 194-210.

Wilkinson, L. (2005), *The Grammar of Graphics (2nd Ed.)*, Springer-Verlag, New York.

Wilkinson, L., Anand, A., & Grossman, R. (2005), "Graph-Theoretic Scagnostics", Proceedings of the IEEE Symposium on Information Visualization, October 2005, Minneapolis, MN, 157- 164.

Wishart, D. (1999), "ClustanGraphics 3 Interactive graphics for cluster analysis". In: Gaul W, Locarek-Junge H (Eds). *Classification in the Information Age*, 268-275, Springer.

Wright, K. (2006), "corrgram: Plot a correlogram", R package version 0.1.

Wu, H-M., Tien Y-J., & Chen, C-H., (2010), "GAP: A graphical environment for matrix visualization and cluster analysis", *Computational Statistics & Data Analysis*, 54(3), 767-778.

Zeileis, A., Hornik, K. & Murrell P., (2008), "Escaping RGBland: Selecting colors for statistical graphics", *Computational Statistics & Data Analysis*, 53(9), 3259-3270.