Preserving Measured Structure During Generation and Reduction of Multivariate Point Configurations

by

Adam Rahman

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Statistics

Waterloo, Ontario, Canada, 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner	Daniel B. Carr Professor Department of Statistics George Mason University
Supervisor	R. Wayne Oldford Professor Department of Statistics and Actuarial Science University of Waterloo
Internal Members	Greg Bennett Professor Emeritus Department of Statistics and Actuarial Science University of Waterloo
	Marius Hofert Assistant Professor Department of Statistics and Actuarial Science University of Waterloo
	Matthias Schonlau Professor Department of Statistics and Actuarial Science University of Waterloo
Internal-external Member	Grant Weddell Associate Professor David R. Cheriton School of Computer Science University of Waterloo

Author's Declaration

Statement of Contributions

Chapters 4 - 6 include work that was published in the paper **Euclidean Distance Matrix Completion and Point Configurations from the Minimal Spanning Tree** by Adam Rahman and R. Wayne Oldford [89].

Abstract

Inherent in any multivariate data is *structure*, which describes the general shape and distribution of the underlying point configuration. While there are potentially many types of structure that could be of interest, consider restricting interest to two general types: geometric structure, the general shape of a point configuration, and probabilistic structure, the general distribution of points within the configuration.

The ability to quantify geometric structure is an important step in many common statistical analyses. For instance, general neighbourhood structure is captured using a k-nearest neighbour graph in dimension reduction techniques such as isomap [103] and locally-linear embedding [93]. Neighbourhood graphs are also used in sensor network localization, which has applications in fields such as environmental habitat monitoring [72] and wildlife monitoring [101]. Another geometric graph, the convex hull, is also used in wildlife monitoring as a rough estimate of an animal's home range [65].

The identification of areas of high and low density is one example of measuring the probability structure of a configuration, which can be done using a wide variety of methods. One such method is using kernel density estimation, which can be viewed as a weighted sum of nearby points. Kernel density estimation has widely varying applications, including in regression analysis [80][116], and is used in general to assess certain features of the data (modality, skewness, etc.).

Related to the idea of measuring structure is the concept of "Cognostics" [110], which has been formalized as scatterplot diagnostics (or scagnostics) [119][120]. Scagnostics provides a framework through which interesting structure can be measured in a configuration. The central idea is to numerically summarize the structure of a large number of two-dimensional point configurations via measures calculated on geometric graphs. This allows the interesting views to be quickly identified, and ultimately examined visually, while the views deemed to be uninteresting are simply discarded. While a good starting point, several issues in the current framework need to be addressed. For instance, while each measure is designed to be in $[0, 1]$, there are some that, when measured over tens of thousands of configurations, fail to achieve this range. In addition, there is a lot of structure that could be considered interesting that is not captured by the current framework. These issues, among others, will be addressed and rectified so that the current scagnostic framework can continue to be built upon.

With tools to measure structure, attention is turned to making use of the structural information contained in the configuration. Consider the problem of preserving measured structure during the task of data aggregation, more commonly known as binning. Existing methods of data aggregation tend to exist on two ends of the structure retention spectrum. Through experimentation, methods such as equal width [121] and hexagonal binning [18] will be shown to tend to retain the shape of the configuration, at the expense of the density, while methods such as equal frequency [121] and random sampling tend to retain relative density at the expense of overall shape. *Tree-based binning*, a general binning framework inspired by classification and regression trees, is proposed to bridge the gap between these sets of specialist algorithms. *GapBin*, a specially designed tree-

based binning algorithm, will be shown through experimentation to provide a trade-off in low dimensional space between geometric structure retention and probabilistic structure retention. In higher dimensions, it will be shown to be the superior algorithm in terms of structure retention among those considered.

Next, the general problem of constructing a configuration with a given underlying structure is considered. For example, the minimal spanning tree is known to carry important clustering information [47]. Of interest then, is the generation of configurations with a given minimal spanning tree structure. The problem of generating a configuration with a known minimal spanning tree is equivalent to completing a Euclidean distance matrix where the only known entries are those in the minimal spanning tree. For this problem, there are several solutions, including those of Alfakih et. al. [2], Fang & O'Leary [34], and Trosset [107]. None of these algorithms, however, are designed to retain the structure of the minimal spanning tree. In addition, the sparsity of the Euclidean distance matrix containing only the minimal spanning tree results in completions that are not accurate as compared to the known completion. This leads to issues in the point configurations of the resulting completions. To resolve these, two new algorithms are proposed which are designed to retain the structure of the minimal spanning tree, leading to more accurate completions of these sparse matrices.

To complement the algorithms presented, implementation of these algorithms in the statistical programming language R will also be discussed. In particular, the R package *treebinr* for tree-based binning, and *edmcr* for Euclidean distance matrix completions will be presented.

Acknowledgments

I would like to thank my supervisor Wayne Oldford for all of his guidance, advice, probing questions, and thoughtful insight. His expertise and knowledge in statistics, visualization, data analysis, and teaching have had a profound impact on my academic development, and for that I am very grateful.

Second, to the giants upon whose shoulders I stand, thank you.

Most importantly, I would like to thank my mother and father, Sue and Ziad, and my grandmother Nancy. Without their constant love, support, and generosity I would not be where I am today. For that, I am forever in your debt.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Consider a set of observations $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n] \in \mathbb{R}^{n \times d}$, which creates a Cartesian point configuration in a d-dimensional Euclidean space. Inherent in this configuration is *structure* - measurable characteristics of the point configuration. Two examples of measurable structure include

1. Probabilistic Structure - the structure of a point configuration that can be measured via density estimation.
2. Geometric Structure - the structure of a point configuration that can be measured using a geometric graph.

Measuring probabilistic and geometric structure in a point configuration, retaining it during standard statistical analyses, and generating configurations containing this structure will be the major subject of this thesis.

1.1 Probabilistic Structure

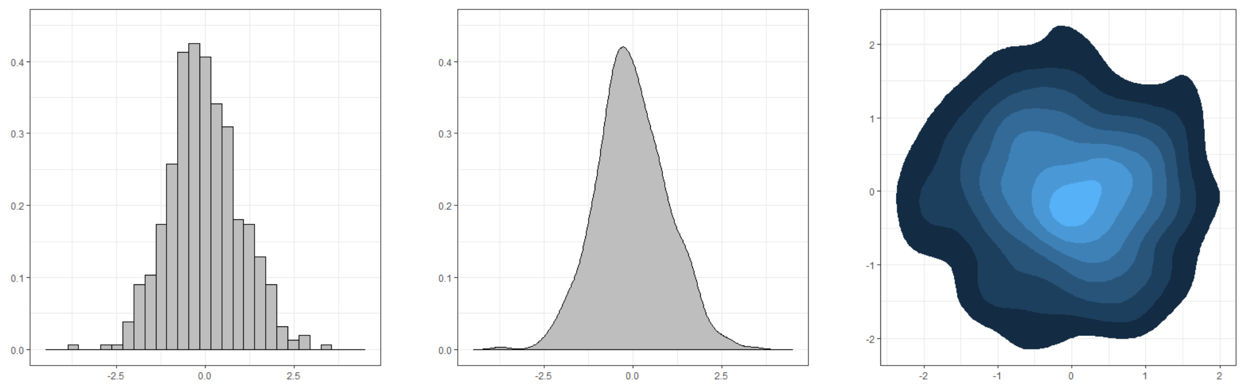
Probabilistic structure refers to the structure in a point configuration that can be measured using density estimation techniques. The histogram [85] is a classic density estimation tool, which first partitions the domain of the data into equally sized bins, and then counts the number of occurrences in each bin. The computed density in each partition is then

$$\text{Density over Domain of Partition} = \frac{\text{Number of Observations in Partition}}{\text{Total Number of Observations} \times \text{Width of the Partition}}$$

This approach leads to a “rough” estimate of the density of the observed data. A smoother estimate of the density can be achieved using kernel density estimation [84][91]. Using a kernel function K, the estimate of density at a point x is computed as

$$\widehat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where h is a smoothing parameter to be chosen. In addition to being a smoothed estimate of the density, kernel density estimation is not limited to univariate observations, and can be computed for arbitrary dimension d. Figure 1.1 illustrates the differences between a histogram and kernel density estimate, and provides an example of a two dimensional kernel density estimate.



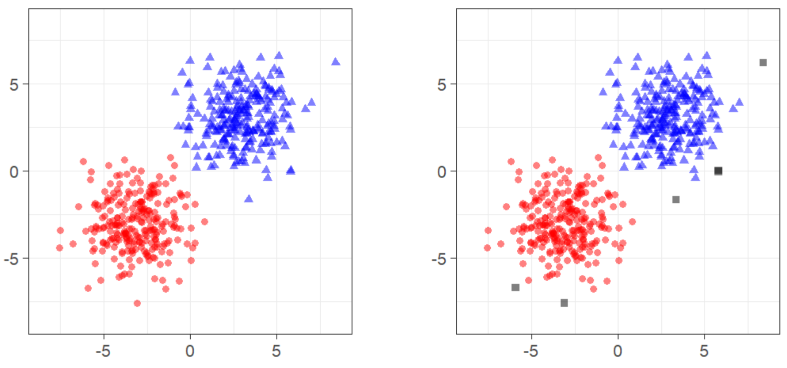
(a) (b) (c)

Figure 1.1: *(a) A histogram of a 500 point sample from a univariate standard normal distribution. (b) A kernel density estimate of a 500 point sample from a univariate standard normal distribution. (c) A two dimensional kernel density estimate of a 500 point sample from a standard bivariate normal distribution. Lighter areas represent areas of high density.*

In addition to numerical identification, probabilistic structure can also be captured by applying graph objects to the point configuration. For instance, identifying areas of high density via nearest neighbours is a key component in the density-based spatial clustering of applications with noise (DBSCAN) algorithm [33]. DBSCAN identifies points that are in the same high density region, and places them in a cluster. Beginning with a measure of distance ϵ and a minimum count m, DBSCAN proceeds as follows for each point p

1. Identify all other points that are within a distance ϵ of p.
2. If p has at least m points within a distance ϵ of it (including itself), then label it a *core point*.

A cluster is formed by beginning with a core point, and including all other points that are reachable from it (including all other core points). It should be noted that every point that is labelled as a core point will belong to a cluster, but not all non-core points in the configuration are necessarily part of a cluster. Points not placed in a cluster are considered to be noise. An example of using DBSCAN to identify two bivariate normal clusters is shown in Figure 1.2. The bivariate normal clusters are drawn from $\mathcal{N}(\mu_1, \Sigma)$, and $\mathcal{N}(\mu_1, \Sigma)$ distributions, where $\boldsymbol{\mu}_1 = [-3, -3]$, $\boldsymbol{\mu}_2 = [3, 3]$, and $\boldsymbol{\Sigma} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$.



(a) (b)

Figure 1.2: *(a) A sample of 500 points from a 50/50 mixture of two bivariate normal distributions. Points emanating from* $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$ *are red circles, and points from* $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$ *are blue triangles. (b) DBSCAN is able to identify both clusters quite clearly (using a value of* $\epsilon = 1.25$*, and* $m = 5$*), with a few points labelled as "noise" (grey squares). It should be noted that varying either* ϵ *or* m *results in varying outcomes.*

1.2 Geometric Structure

In addition to probabilistic structure, the identification and preservation of *geometric structure*, defined to be the shape information which can be captured by a geometric graph on a point configuration, is also of interest. For example, in sensor network localization, the goal is to determine the positions of a set of *sensors*, based only on the position of a few known *anchors*. Distances between sensors, as well as between sensors and anchors, are known only if they are within some (Euclidean) distance of one another, known as the *radio range*, denoted by R. The resulting *partial distance matrix* can be viewed as a weighted neighbourhood graph on the sensors and anchors.

An example of such a problem is described in [90], where access points (i.e. anchors) use received signal strength from wireless users (emitters, i.e. sensors) to attempt to locate the positions of access points with *unknown* locations. Figure 1.3 illustrates the problem.

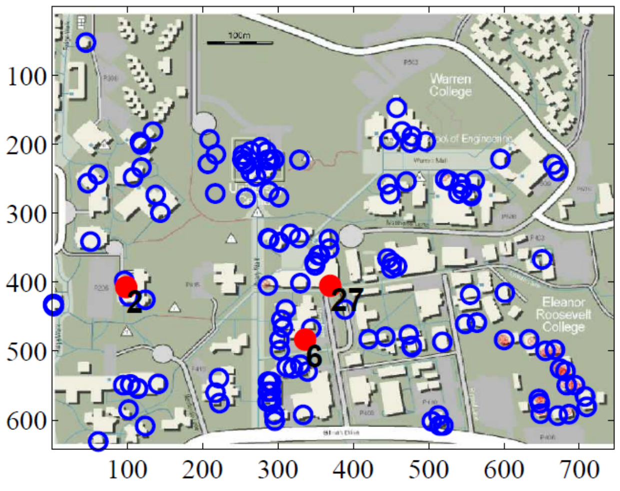


Figure 1.3: *From [90], a map of the 200 known access point locations (blue open circles). Also shown are access points sampled by a single user (red closed circles) at a certain moment in time, along with the strength of the received signal strength.*

To model this problem as a sensor network localization problem, the received signal strength is first converted to a distance. For emitter i and access point j, let s_{ij} represent the received signal strength at the access point. Then,

$$d_{ij} = d_0 * 10^{(s_0 - s_{ij})/(10 n_p)}$$

where s_0 is the received signal strength at the access point at a reference distance of d_0. The variable n_p is known as the "path-loss exponent", and is environment dependent. It can be viewed as a noise term, as it is generally unknown.

With these individual distances, the distance matrix $\mathbf{D}$ has the following form

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_{kk} & \mathbf{D}_{ke} & \mathbf{D}_{ku} \\ \mathbf{D}_{ek} & \mathbf{D}_{ee} & \mathbf{D}_{eu} \\ \mathbf{D}_{uk} & \mathbf{D}_{ue} & \mathbf{D}_{uu} \end{bmatrix}$$

where subscript k denotes the known access points, u the unknown access points, and e the emitters. $\mathbf{D}_{kk}$ represents the distance matrix containing the distances between the known access points, $\mathbf{D}_{ke}$ the distance matrix between the known access points and the emitters, etc.

Since each emitter will only be within range of a subset of the known and unknown access points, both $\mathbf{D}_{ek}$ and $\mathbf{D}_{uk}$ will be a partial distance matrix (denoted with a superscript $\star$). Also, since there are access points with unknown locations, the distance matrices $\mathbf{D}_{uu}$ and $\mathbf{D}_{uk}$ are both completely unknown. The result is the following partial distance matrix $\mathbf{D}^\star$ to be completed, with the end goal being to reconstruct the complete matrix $\mathbf{D}$ from $\mathbf{D}^\star$.

$$\mathbf{D}^\star = \begin{bmatrix} \mathbf{D}_{kk} & \mathbf{D}^\star_{ke} & \star \\ \mathbf{D}^\star_{ek} & \star & \mathbf{D}^\star_{eu} \\ \star & \mathbf{D}^\star_{ue} & \star \end{bmatrix}$$

As a practical application, [90] consider data collected at the University of California, San Diego [76]. The data represents received signal strength data for personal data assistants (PDAs) given to 275 freshman students. Over the course of the experiment, 300 unique access points were sensed, however only 200 had known locations.

The goal of the analysis was to locate the approximately 100 access points with unknown locations. Figure 1.4(a) demonstrates the accuracy of the algorithm by first finding the position of four access points whose positions were hidden from the algorithm. Figure 1.4(b) shows the 200 known access points (blue circles), as well as the estimated positions of some of the unknown access points (red triangles).

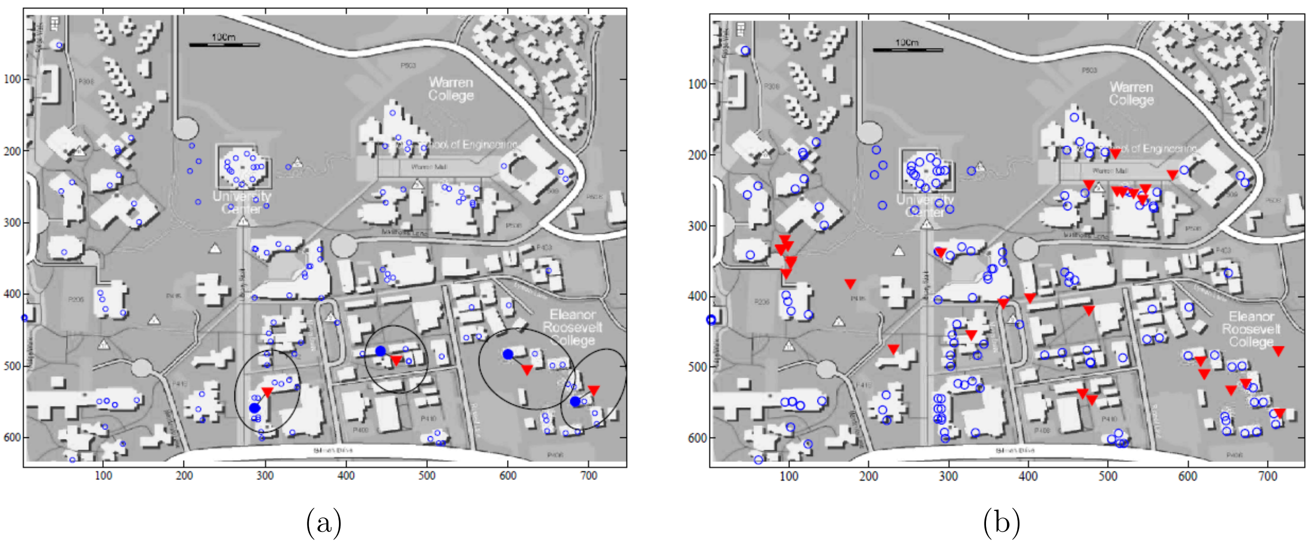


Figure 1.4: *From [90] (a) The proposed positions (red triangles) of four access points (of known location) whose locations were hidden from the algorithm. Actual locations are shown as blue circles. (b) The proposed positions of some of the unknown access points (red triangles), and the positions of the 200 known access points (blue circles).*

Figure 1.5 illustrates the general problem. Here, blue circles represent access points with known positions, red triangles access points with unknown positions, and grey squares emitters with unknown positions. Solid lines represent known distances, while dotted lines represent unknown distances, forming a partial distance matrix. The goal of the analysis is to find the positions of the emitters and access points with unknown locations.

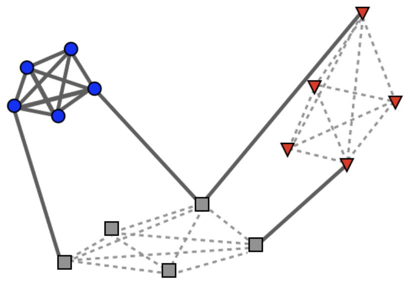


Figure 1.5: *The general sensor network localization problem of [90]. Blue circles represent access points with known positions, red triangles access points with unknown positions, and grey squares emitters with unknown positions. Solid lines represent known distances, while dotted lines represent unknown distances. The goal is to find the positions of the emitters and access points with unknown locations.*

In addition to geo-tracking problems such as this, sensor networks have a large range of applications, such as military/security applications [69], environmental habitat monitoring [72], and wildlife monitoring [101], amongst others. There are many proposed methods for solving sensor network localization problems, ranging from relaxations of semi-definite programming algorithms to iterative multidimensional scaling (see for example [32], [67], or [90]).

A much more constrained problem in this same space is the reconstruction of a protein molecule based on incomplete inter-atomic distances. Specifically, the *molecule problem* looks to reconstruct a three-dimensional molecule (most often a protein structure) using incomplete interatomic distances measured via nuclear magnetic resonance (nmr) spectroscopy. While some interatomic distances may be unknown, there is a vast amount of structure that exists in a molecule that can be exploited, as nature gives insights into the angles and distances at which certain atoms must bond. Having *a priori* knowledge of the atomic structure of a molecule will allow for the determination of bounds on the possible distances that can theoretically exist between atoms. Other phenomena such as natural repulsion/attraction forces, and known atomic structure enforce natural bounds on the possible distances that can exist between pairs of atoms.

Given all these restrictions, the problem is to then (as accurately as possible) propose a series of plausible configurations for the molecule - as a set of bounds rarely can completely specify a distance matrix, there are likely many valid solutions to a single conformation problem. There is a large volume of research done in this area, including [44], [51], [53], [78], [106], [122].

One of the first attempts to solve this problem was presented in [51]. Using nuclear magnetic resonance imaging, they extracted intramolecular, interproton distances from the molecular backbone of various protein structures. As they had the exact measurements (in Ångströms, $\mathring{A}$), the data were modified such that each known distance was placed in an appropriate range. For example, distances shorter than $2\mathring{A}$ were instead treated as unknown distances in the range $[0,2)\mathring{A}$. In total, 10 data sets were considered, each with varying levels of restrictions on bonding angles and distances placed on them. Figures 1.6 and 1.7 illustrate the results from two of the experiments.

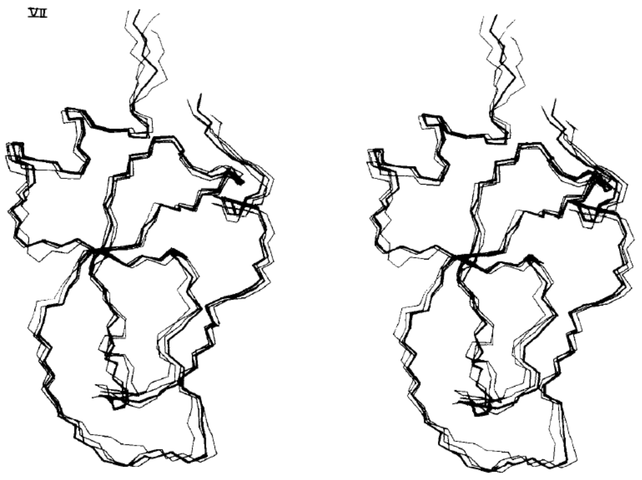


Figure 1.6: *A stereographic image of three attempts to replicate the crystal structure of a BPTI protein. The actual crystal structure is superimposed in darker black. This example shows a protein reconstruction with the most known structure of all protein reconstruction attempts in [51].*

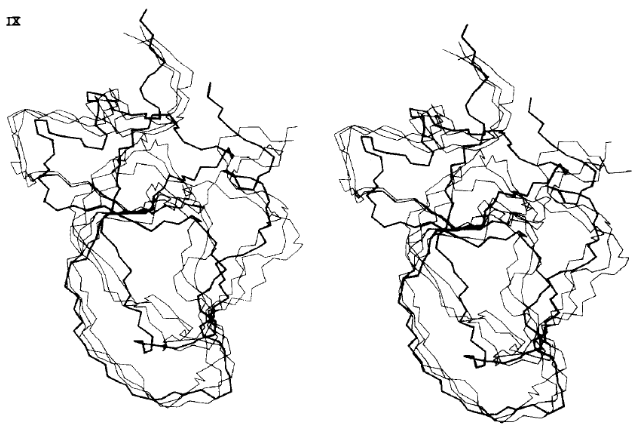


Figure 1.7: *A stereographic image of three attempts to replicate the crystal structure of a BPTI protein. The actual crystal structure is superimposed in darker black. This example shows a protein reconstruction with the least known structure of all protein reconstruction attempts in [51].*

The importance of preserving geometric structure is also present in fields such as dimension reduction, where, as the name implies, the goal is to reduce the dimension of a configuration, but to do so in such a way as to minimize the damage done to the underlying manifold. For instance, the isomap algorithm [103] first produces a (connected) neighbourhood graph in the native dimension of the data, thus preserving *local* distances between points. Remaining distances are calculated as the shortest path along the neighbourhood graph, approximating the *geodesic* distance between points. Figure 1.8 illustrates the construction of this distance matrix using a simple two dimensional point configuration.

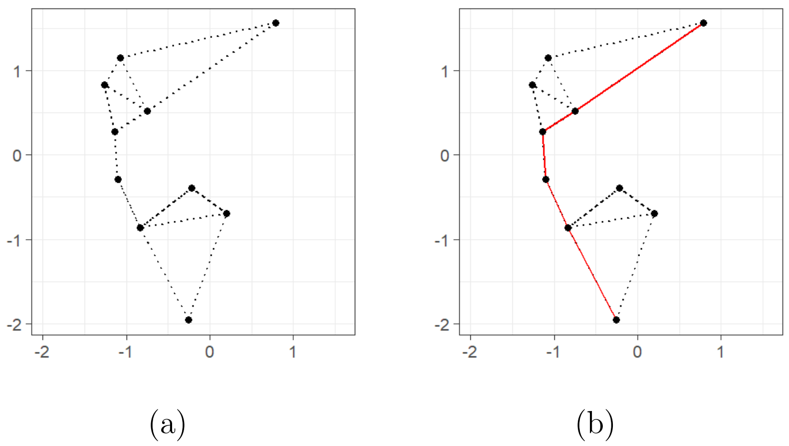


Figure 1.8: *(a) A 2-nearest neighbour graph that could be used in isomap, since the minimum requirement of connectedness is achieved. (b) An example of the shortest path between two points along the graph. This distance is used as an estimate of the geodesic distance between two points along the manifold, and is used in place of the Euclidean distance between the points.*

With an (approximate) distance matrix for the data in its original dimension, multidimensional scaling is used to decrease the dimension of the data to that desired. Figure 1.9 provides an example of a three dimensional manifold produced by isomap. The original data exists in 4096 dimensions.

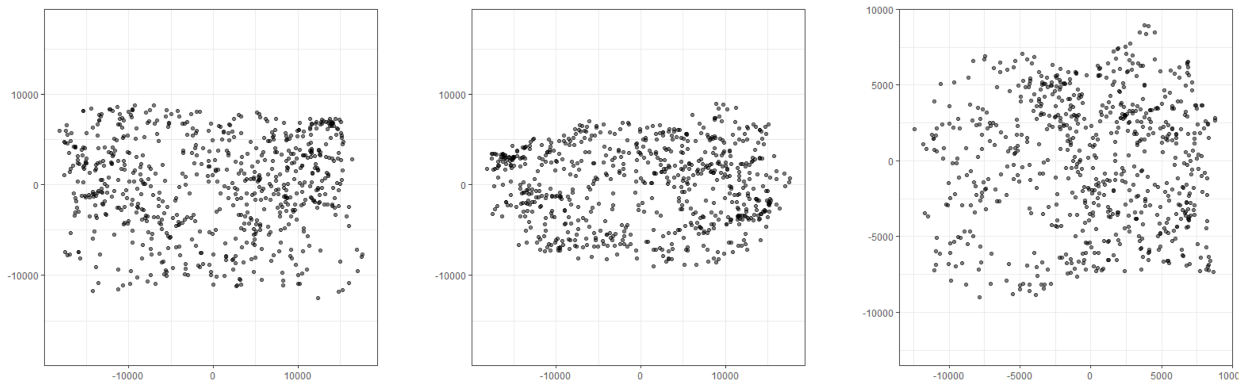


Figure 1.9: *Two dimensional views of each pair of principal directions for the three dimensional manifold of the statue data produced by isomap. From left to right: (Dimension 1, Dimension 2), (Dimension 1, Dimension 3), (Dimension 2, Dimension 3).*

Isomap is just one example of a dimension reduction algorithm that seeks to preserve geometric structure. Other examples include Laplacian Eigenmaps [9] and locally linear embedding [93], among many others.

1.3 Data Aggregation

In dealing with data that has not only a large number of dimensions, but also a large number of observations, it may first be beneficial to decrease the number of observations (a task typically referred to as *binning*) before undertaking statistical analyses, as simple tasks such as the construction of the distance matrix have computational complexity that scale exponentially with the number of observations. Much of the recent work in the field of binning has been centred around its use in classification and kernel density estimation, and not necessarily in its more general use of simply decreasing the number of observations in a configuration.

In classification, the goal of binning is to *discretize* a continuous attribute into a finite number of discrete subsets. This is done to accommodate continuous attributes in classification algorithms that require strictly discrete attributes. Of the large number of algorithms that exist (see [42] for a survey), some of the better overall performers include Multivariate Discretization [8], MODL [10], and ChiMerge [64]. The issue with these algorithms (and the vast majority of the algorithms developed for these classification problems) is that they are *supervised* algorithms - they require a class variable $\mathbf{y}$. In many data sets, such a class variable is not present, so a more general algorithm is required to decrease the number of observations in a general configuration.

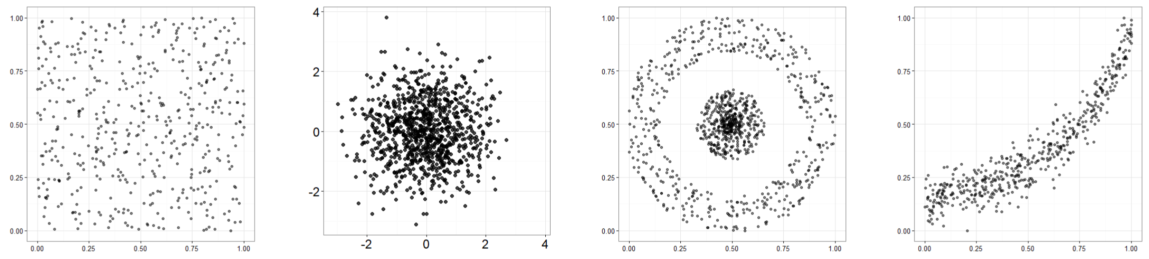


Figure 1.10: *Examples of simulated configurations on which geometric and probabilistic structure will be evaluated before and after binning.*

Unlike in classification, in kernel density estimation the goal of binning is to reduce the number of observations in the configuration before computing the density estimates. The effects of various binning algorithms on the accuracy of the density estimate has been studied (see [50] & [58]), and the method found to be best in terms of accuracy is *linear binning* [50], which aims to replace each point by its nearest *grid point* - essentially turning a point scatter into a weighted grid, decreasing the number of points at which the kernel density estimate must be computed. While this method may be appropriate as an approximation in the context of kernel density estimation, seeing use in algorithms such as the local polynomial regression algorithm RODEO [95], it is not a generally appropriate method to reduce the number of observations in a large data set, as data tend to not lie on a perfect grid.

What is left for use in the general case of decreasing the observation count is a narrow selection of unsupervised binning algorithms. Some of the more well known include equal width [121], equal frequency [121] (and the closely related fixed frequency [123]), and hexagonal binning [18]. While not a binning algorithm per se, random sampling can also be used to efficiently reduce the number of observations in a point configuration.

With multiple algorithms to choose from, of practical importance is to choose one that minimally damages the underlying probability and geometric structure of the data, limiting the impact of binning on any analyses undertaken. This can be explored empirically by first simulating data sets (such as those in Figure 1.10), decreasing the number of observations using various binning algorithms, and measuring the effect of binning on the structure. By including configurations with various types of underlying structure, such as configurations with high and low density areas, configurations that have potential outlying points, etc., the strengths and weaknesses of each of the algorithms can be examined.

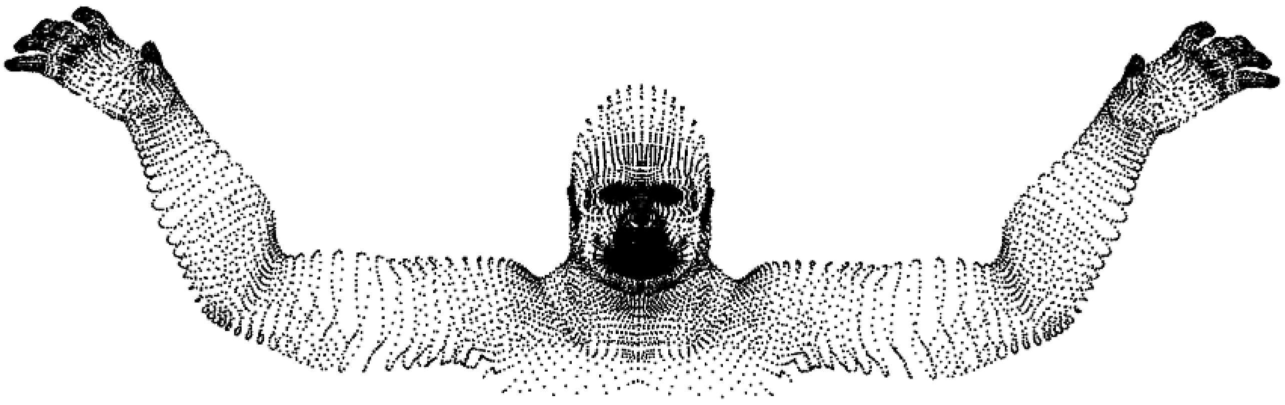


Figure 1.11: *An example of a three dimensional structure which contains both a general shape and fine detail that would ideally be preserved during data aggregation.*

In addition to the simulated data sets, having real data configurations with clear structure, such as those in the TOSCA high resolution data set [14], are very useful for visually confirming the results of the simulation. Using images such as those in Figure 1.11, the ability of various binning algorithms to retain the overall structure of the shape of each image can be visually assessed. The retention of fine detail, such as in the face and fingers of the images, can also be examined.

An algorithm's ability to bin in low dimensional space does not necessarily imply it will continue to be viable as dimension increases. For example, the number of bins under consideration in the equal width algorithm scales exponentially with the dimension of the data. To place k bins in each of d dimensions, the equal width algorithm must place k^d bins, increasing the computational complexity of the algorithm significantly. In turn, this likely impacts the accuracy of the algorithm as fewer bins can reasonably be considered in each dimension. Tukey, in the context of capturing density, takes a much more blunt stance on binning in high dimensions [112]

> *The number of potential combinations and subdivisions of the coordinates rises so fast with the number of dimensions that we cannot stick to "counts in bin." In fact, it is difficult to do well with bins in so few as two dimensions. Clearly, bins are for the birds!*

To see if bins are truly for the birds, an analysis of high dimensional data can be undertaken. The algorithms capable of binning in high dimensional space can be used to bin data in their native dimension, and then, using a method such as as isomap, the differences between the non-binned data and the binned data sets can be visually assessed. For this, image data sets in varying high dimensional space, such as that in Figure 1.12, could be considered.

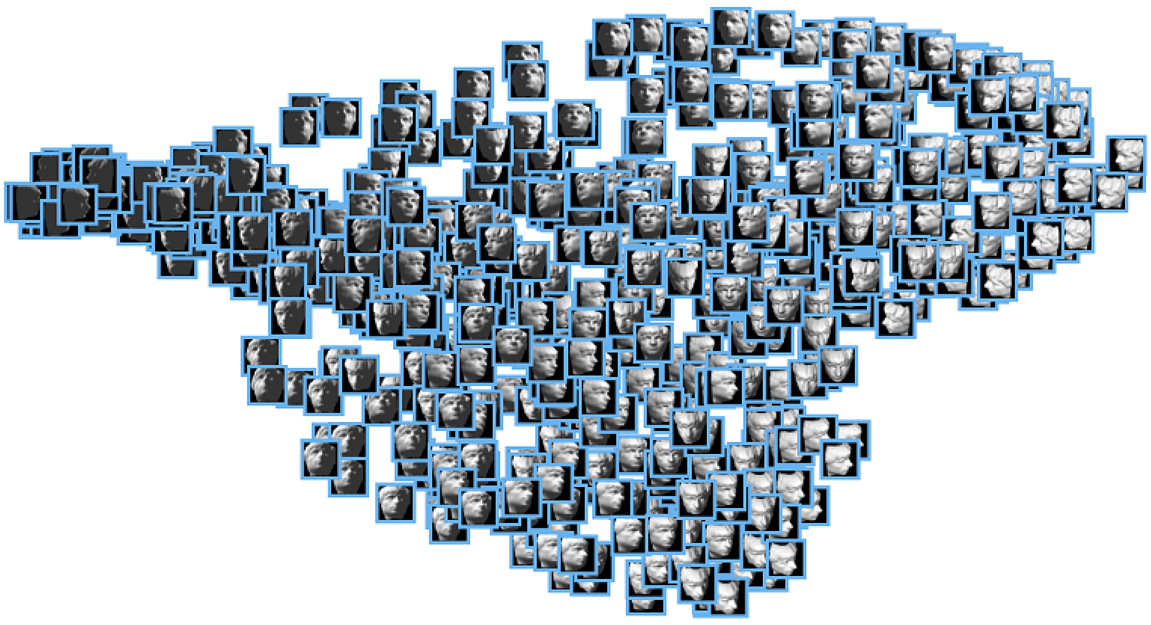


Figure 1.12: *Views of the first two dimensions of the three dimensional manifold produced by multidimensional scaling of an image data set. The data was originally of dimension 4096.*

1.4 Thesis Structure

The main theme of this thesis will revolve around the identification of structure, as well as the preservation of structure during both the reduction and generation of point configurations. The thesis will be structured as follows.

Chapter 2 introduces probabilistic and geometric structure by reviewing some of the ways in which each type of structure is measured. Also introduced is the concept of scatterplot diagnostics, or *scagnostics* [119], which are designed to summarize these types of structure in a point configuration. Some of the experiments in [120] are also reproduced. Chapter 3 discusses some of the issues with the current implementation of scagnostics, and offers some suggestions as to potential solutions.

Chapter 4 introduces Euclidean distance matrix completion by reviewing the underlying mathematics and some of the existing methods in this field. Examples of generating configurations with desired structure (such as those with extreme values of a chosen scagnostic) are presented both as a motivating example and as an additional application of a newly proposed Euclidean distance matrix completion method. Chapter 5 will discuss the novel implementation of these algorithms in the statistical software R, which is used to facilitate the experiments conducted in Chapter 6.

Chapter 7 explores existing techniques for data aggregation, including both supervised and unsupervised techniques and their role in data analysis. In addition, tree-based binning, a new binning framework, is proposed. Chapter 8 discusses the implementation of tree-based binning in R. Chapter 9 examines the strengths and weaknesses of the binning methods introduced in Chapter 7, specifically analyzing their ability to preserve both probabilistic and geometric structure over configurations with varying numbers of observations and dimension.

Finally, Chapter 10 introduces solutions to the problems identified in the scagnostic framework in Chapter 3. In addition, new scagnostics that capture additional interesting structure are proposed, and, building off of the work done in Chapter 4, methods for generating configurations with a given scagnostic level are introduced. As a future work project, a proposal for a new scagnostic framework is made, with a theoretical layout in R.

Chapter 2

Background

This chapter introduces the necessary background material to understand the concepts of probability and geometric structure, and the tools used to capture them. Section 2.1 introduces the tools that will be used to measure the probability structure of a point configuration. Section 2.2 introduces a wide range of geometric graph objects that can be used to measure geometric structure. Finally, Section 2.3 introduces *scagnostics* [119], a framework for numerically quantifying "interesting" structure in two-dimensional point configurations.

2.1 Probability Structure

Probability structure is measured by identifying points of interest that exist in the probability density of a point configuration. For example, identifying areas of high and low point density and transition points between these areas could lead to interesting structural discoveries in a configuration. Also, density comparisons between configurations allows for the structure of a "non-interesting" configuration, such as a uniform or normal point scatter, to be contrasted against configurations of unknown structure to test for deviations (and hence "interestingness").

A popular method for computing smoothed density estimates of a point configuration is *kernel density estimation*, computed at a point x using the traditional estimate

$$\widehat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where h is the bandwidth, and the function K is the kernel function, which can be viewed as a weighting function for the x_i. Table 2.1 lists some univariate kernel functions. A formal treatment of the subject is given in [84][91], and [97] provides a comprehensive introduction to the methodology, and introduces a number of applications.

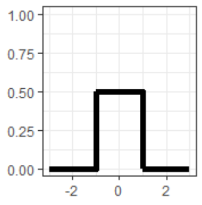
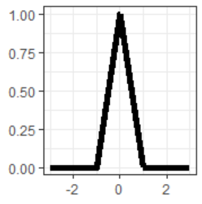
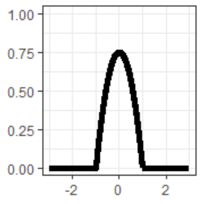
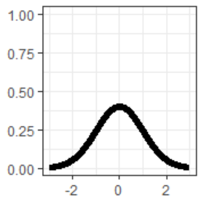
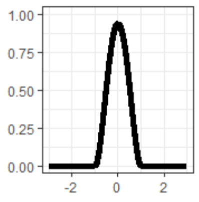
Kernel	**Functional Form**	**Support**	**Visualization**
Uniform	$K(x) = \frac{1}{2}$	$-1 \leq x \leq 1$	
Triangular	$K(x) = 1 - \lvert x \rvert$	$-1 \leq x \leq 1$	
Epanechnikov	$K(x) = \frac{3}{4}(1 - x^2)$	$-1 \leq x \leq 1$	
Gaussian	$K(x) = \frac{1}{\sqrt{2\pi}} e^{-0.5x^2}$	$-\infty < x < \infty$	
Biweight	$K(x) = \frac{15}{16}(1 - \lvert u \rvert^2)^2$	$-1 \leq x \leq 1$	

Table 2.1: *The functional form, range, and visualization for five kernels.*

Using different kernel functions results in slightly different estimates of the density. Figure 2.1 illustrates the effect of estimating the density of a sample from a standard normal distribution using each of the kernels listed Table 2.1.

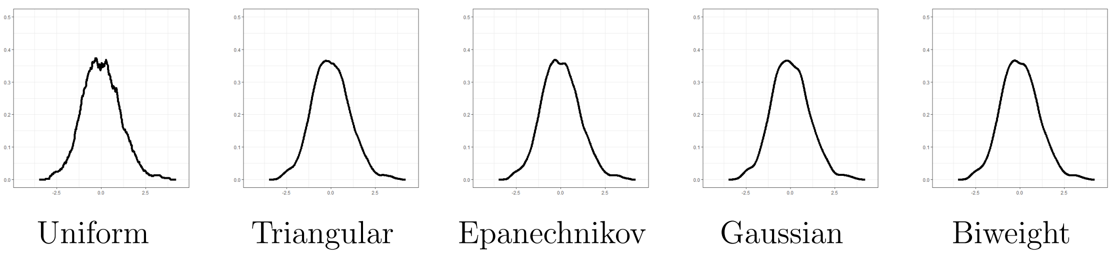


Figure 2.1: *Kernel density estimates using five different kernel functions for a sample of size 500 from a univariate standard normal distribution.*

Kernel density estimation can be easily generalized to include estimation in higher dimensions

$$\widehat{f}(\mathbf{x}) \;=\; \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}_i)$$

where the matrix $\mathbf{H}$ is a symmetric, positive definite *smoothing* matrix, and

$$K_{\mathbf{H}}(\mathbf{x}) \;=\; |\mathbf{H}|^{-0.5} K(\mathbf{H}^{-0.5}\mathbf{x})$$

In higher dimensions, the multivariate Gaussian kernel is a common choice for the kernel function

$$K_{\mathbf{H}}(\mathbf{x}) \;=\; (2\pi)^{-\frac{d}{2}}\;|\mathbf{H}|^{-0.5}\; e^{-0.5\mathbf{x}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{x}}$$

With the ability to compute estimates of the density for a given configuration of points, comparing the density of multiple configurations is a natural method to attempt to uncover interesting structure, as it allows uninteresting configurations, such as a uniform or normal configuration, to be compared to a configuration of unknown structure.

A very simple method to compare the densities of two configurations is to consider the absolute difference between their kernel density estimates. For two configurations P and Q, let $\widehat{f}_P$ and $\widehat{f}_Q$ be the kernel density estimates of P and Q respectively. Then, for randomly chosen points x_i, the absolute difference between the estimated densities is simply

$$\sum_{i=1}^{n} |\widehat{f}_P(x_i) - \widehat{f}_Q(x_i)|$$

A more sophisticated method of comparison between two probability distributions is Kullback-

Leibler divergence [68]. For two (discrete) probability distributions P and Q, the Kullback-Leibler divergence from P to Q is defined to be

$$D_{KL}(P||Q) \;=\; \sum_{i=1}^{n} p_i \; log \left(\frac{p_i}{q_i} \right)$$

An analogous definition exists for two continuous probability distributions. Note that this definition is an expectation with respect to the probability distribution P, resulting in a non-symmetric measure (i.e. $D_{KL}(P||Q) \neq D_{KL}(Q||P)$) . The original definition of Kullback-Leibler divergence was in fact

$$\begin{aligned} D_{KL}(P||Q) + D_{KL}(Q||P) &= \textstyle\sum_{i=1}^{n} p_i \; log \left(\frac{p_i}{q_i} \right) + \sum_{i=1}^{n} q_i \; log \left(\frac{q_i}{p_i} \right) \\ &= \textstyle\sum_{i=1}^{n} (p_i - q_i)(log(p_i) - log(q_i)) \end{aligned}$$

which symmetrizes the expectation. A related measure known as Jensen-Shannon divergence measures the similarity between two probability distributions by comparing each distribution to the average of both [73]

$$D_{JS} = \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M)$$

where M is defined to be the average of the two distributions, $M = \frac{1}{2}(P + Q)$.

Both Kullback-Leibler and Jensen-Shannon divergence methods have been used in various applications in the literature such as support vector machine classification [79], signal processing [4], and genome comparison [98].

2.2 Geometric Structure

Consider a point configuration of size n in arbitrary dimension d given by the $n \times d$ matrix $\mathbf{X} = [x_{ij}]$, with associated *squared* Euclidean distances given by $\mathbf{D} = [d_{ij}]$. For each pair of points $\mathbf{x}_i, \; \mathbf{x}_j \;\; i, j = 1, ...n, \; i \neq j, \; d_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||^2 = \sum_{k=1}^{d} (x_{ik} - x_{jk})^2$. If we consider $\mathbf{x}_i \in \mathbf{X}$ to be a node $\mathbf{v}_i \in \mathbf{V}$, with weighted edge length $\mathbf{e}_{ij} \in \mathbf{E}$ such that $\mathbf{e}_{ij} = d_{ij}$, we can treat this configuration as a weighted graph object, which allows for the computation of many important geometric graph objects.

2.2.1 Voronoi Diagrams & Delaunay Triangulations

Voronoi diagrams and Delaunay triangulations are the building blocks of many geometric graph objects. The Voronoi diagram provides nearest neighbour information for a set of nodes by creating a cell around each node in the set. The Voronoi cell surrounding the node $\mathbf{v}_i$ indicates that any point placed in the cell will be closer (by Euclidean distance) to $\mathbf{v}_i$ than any other node in the configuration $\mathbf{v}_j$, $j \neq i$. More formally, the Voronoi diagram is defined as

Definition 1. *Given a set of points $\mathbf{X} = [\mathbf{x}_i^T] \in \mathbb{R}^d$, the Voronoi cell of the point $\mathbf{x}_i$ is the set of points in $\mathbb{R}^d$ which are closer to $\mathbf{x}_i$ than to the other points in $\mathbf{X}$*

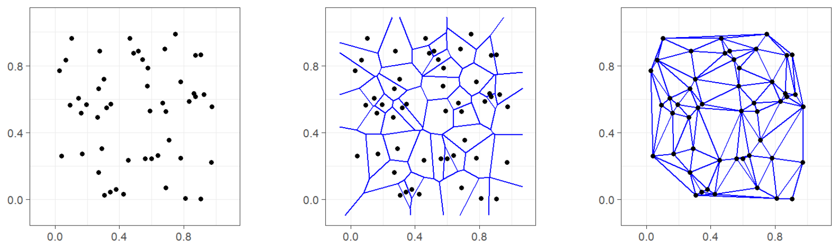
$$\mathbf{V}_i = \{\mathbf{x} \in \mathbb{R}^d : d(\mathbf{x}, \mathbf{x}_i) < \min_{j \neq i} ||x - x_j||\}$$

The Voronoi diagram is the collection of all such cells [74].

The Delaunay triangulation is closely related to the Voronoi diagram

Definition 2. *The Delaunay triangulation is the graph with node set $\mathbf{V} = \mathbf{X}$, where $\mathbf{v}_i$ and $\mathbf{v}_j$ share an edge if and only if the Voronoi cells for $\mathbf{x}_i$ and $\mathbf{x}_j$ share an edge.*

Figure 2.2 provides an example of the Voronoi diagram and Delaunay triangulation for a sample point configuration. Many graph objects, such as the minimal spanning tree, alpha hull, and convex hull, are all subsets of the Delaunay triangulation, allowing for each of these graph objects to be computed more efficiently.



(a) (b) (c)

Figure 2.2: *(a) The original point configuration. (b) The Voronoi diagram of the configuration. (c) The Delaunay triangulation of the configuration.*

2.2.2 Minimal Spanning Tree

A spanning tree of a set of nodes **V** is a set of edges **E** such that

1. no closed loops occur
2. each point is visited at least once
3. the tree is connected

The *minimal* spanning tree is then defined as

Definition 3. *The minimal spanning tree (MST) of a set of nodes* **V** *is the spanning tree with minimum edge weight. In the case of a point configuration* **X**, *these weights correspond to the Euclidean distances [47].*

It should be noted that the minimal spanning tree is not necessarily unique. Uniqueness is only guaranteed in the case of all edge lengths being unique. Algorithm 1, proposed in [87], is a well-known minimal spanning tree algorithm that has a worst-case complexity of $\mathcal{O}(|V|^2)$. Figure 2.3 provides an example of the minimal spanning tree of a set of points.

Algorithm 1 Prim's Algorithm

```
Structures
    $\mathbf{V}$ - Node set containing $\mathbf{v}_i$,    $i = 1, ..., n$;
    $\mathbf{E}$ - Edge set containing the edge weights, $e_{ij}$, of the nodes. If two nodes do not share an edge, then $e_{ij} = \infty$.
    $\mathbf{U} = \varnothing$ - Nodes currently in the minimal spanning tree
    $\mathbf{F} = \varnothing$ - Edges currently in the minimal spanning tree

procedure PRIM'S ALGORITHM($\mathbf{V}$, $\mathbf{E}$)
    Without loss of generality, initialize the tree by choosing a node at random, $\mathbf{v}_i$
    $\mathbf{U} \leftarrow \mathbf{U} \cup \{\mathbf{v}_i\}$                                   ▷ Add the node to the tree
    $\mathbf{V} \leftarrow \mathbf{V} \setminus \{\mathbf{v}_i\}$                                   ▷ Remove the node from the node set
    while $\mathbf{V} \neq \varnothing$ do                                   ▷ While all nodes are not in the tree
        $\{k, j\} = \{(k, j) : \min(\mathbf{e}_{kj}), \mathbf{v}_k \in \mathbf{U}, \mathbf{v}_j \in \mathbf{V}\}$   ▷ Find node in V with smallest edge to any node in U
        $\mathbf{U} \leftarrow \mathbf{U} \cup \{\mathbf{v}_j\}$                               ▷ Add the node to the tree
        $\mathbf{V} \leftarrow \mathbf{V} \setminus \{\mathbf{v}_j\}$                               ▷ Remove the node from the base set
        $\mathbf{F} \leftarrow \mathbf{F} \cup \mathbf{e}_{kj}$                               ▷ Add the edge to the tree edge set
    end while
    return $\{\mathbf{U}, \mathbf{F}\}$                                   ▷ Return the Node and Edge sets for the MST
end procedure
```

An application of the minimal spanning tree comes from its use in cluster analysis. Single-linkage clustering [47] is a simple algorithm in which nodes are clustered at level d_k if there exists an edge in the minimal spanning tree between the node and any other node in the cluster with weight $e_{ij} \leq d_k$. As the level increases, larger clusters are formed until all nodes belong to the same cluster. Clustering at any level d_k can be determined by breaking the minimal spanning tree at any edge $e_{ij} > d_k$ [47]. As the clustering level d_k increases, the clusters at previous levels remain, but some may combine with other clusters to form larger clusters. This allows the evolution of the clustering to be observed at each successive step d_k.

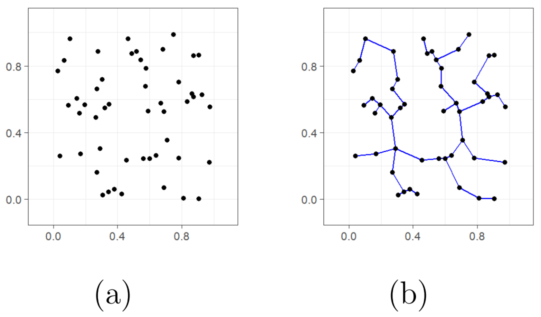


Figure 2.3: *(a) The original point configuration. (b) The associated minimal spanning tree.*

2.2.3 Nearest Neighbour Graphs

Formally, the nearest neighbour of a node $\mathbf{v}_i$ is defined in [74]

Definition 4. *Given a node set* $\mathbf{V}$ *and a corresponding edge set* $\mathbf{E}$*, define* $nn(\mathbf{v}_i)$ *to be the nearest neighbour of* $\mathbf{v}_i \in \mathbf{V}$

$$nn(\mathbf{v}_i) = \underset{\mathbf{v}_j \in \mathbf{V} \setminus \mathbf{v}_i}{\arg\min}\{\mathbf{e}_{ij}\}$$

The nearest neighbour graph is then defined as

Definition 5. *A nearest neighbour graph can be defined by an edge set* $\mathbf{E}$ *and a node set* $\mathbf{V}$ *such that:*

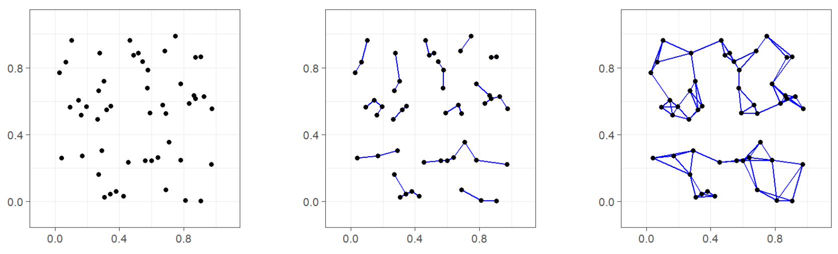
$$\mathbf{e}_{pq} \in \mathbf{E} \implies \mathbf{v}_q = nn(\mathbf{v}_p)$$

There are two distinct differences between the nearest neighbour graph and the minimal spanning tree:

1. The nearest neighbour graph is not necessarily connected.
2. The nearest neighbour graph is a directed graph.

The second point comes from the realization that $\mathbf{v}_q = nn(\mathbf{v}_p) \not\Rightarrow \mathbf{v}_p = nn(\mathbf{v}_q)$. Figure 2.4 provides an example of a nearest neighbour graph.

The nearest neighbour graph can be easily extended to include edges between a point and its k-nearest neighbours, resulting in a similarly directed graph known as the k-nearest neighbour graph. A k-nearest neighbour graph with $k = 3$ is shown in Figure 2.4.



(a) (b) (c)

Figure 2.4: *(a) The original point configuration. (b) The nearest neighbour graph. (c) The 3-nearest neighbour graph.*

The k-nearest neighbour graph has many applications, including in the field of dimension reduction. In particular, it is used in the well known dimension reduction technique isomap [103]. At a high level, isomap looks to preserve relative neighbourhoods by first constructing a neighbourhood graph using k-nearest neighbours. This is done on the configuration in high dimension, and then the lower dimensional embedding is computed by multidimensional scaling, which aims to preserve these neighbourhoods in the low dimensional realization.

2.2.4 Alpha and Convex Hulls

The alpha hull is defined formally in [74]

Definition 6. *For $\alpha > 0$, the alpha hull of a set of points $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ is the graph with vertices $\mathbf{V} = \mathbf{X}$ and edge set $\mathbf{E} = \{\mathbf{v}_i, \mathbf{v}_j : \exists D_\alpha$ with $\mathbf{v}_i, \mathbf{v}_j \in \partial D_\alpha$ and $\mathbf{X} \cap D_\alpha = \varnothing\}$, where D_α is an open disk of radius α, and ∂D_α denotes the boundary of the disk.*

In simpler terms, to construct the alpha hull, place an edge between any two nodes that are on the boundary of a disk of radius α which contains no other points. The value of α used to create the alpha hull greatly affects the resulting graph, which is illustrated in Figure 2.5.

The convex hull is a special case of the alpha hull, and can be computed in the same way by letting α approach infinity, resulting in a structure such as that in Figure 2.5.

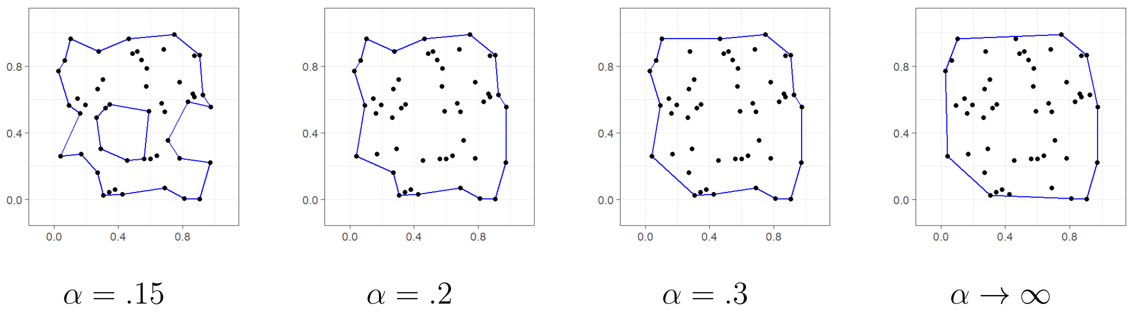


Figure 2.5: *Three examples of an alpha shape computed with varying levels of α. The limiting case ($\alpha \to \infty$), known as the convex hull, is also provided.*

2.3 Scagnostics

Proposed originally by John and Paul Tukey [111], and formalized by Wilkinson, Wills, Anand, and Grossman [119][120], *scagnostics* numerically summarize structure inherent in a point configuration using measures computed on various geometric graphs. The current two-dimensional implementation of [120] is now presented.

2.3.1 Pre-processing

Before the scagnostic measures are computed, a number of pre-processing steps are taken to ensure comparability of the measures between configurations.

Scaling

The configuration is first scaled to lie in the unit square, which is accomplished using min-max scaling. Let $x_{(1)}, ..., x_{(n)}$ and $y_{(1)}, ..., y_{(n)}$ represent the original data sorted from smallest to largest. Then, the data are scaled as follows

$$x_i^* = \frac{x_i - x_{(1)}}{x_{(n)} - x_{(1)}}$$
$$y_i^* = \frac{y_i - y_{(1)}}{y_{(n)} - y_{(1)}}$$

Scaling in this fashion ensures two things

- All computed scagnostics on a given configuration will be constrained to lie in [0,1], giving a common scale for each scagnostic measure to be compared to one another.

- Every configuration will be constrained to lie in the unit cube. This ensures that scagnostics computed on different configurations are directly comparable.

Binning

Hexagonal binning [18] is used to reduce the size of the configuration, reducing the overall run-time of the scagnostic algorithm. Beginning with a 50×50 hexagonal grid, the individual data points are binned into their corresponding hexagons, with the resulting center of mass of each hexagon, $(\bar{x}, \bar{y})$, being the representative point of all points in that bin. If more than 1000 cells are nonempty, the grid size is reduced by two thirds and re-binning occurs. This process continues until there are no more than 1000 nonempty cells. Note that this is how the implementation of scagnostics in R executes binning [118]. In the literature, a 40×40 grid is used, with a reduction of one half of the grid size and rebinning if there are more than 250 nonempty cells.

The hexagonal binning algorithm is more thoroughly discussed in Section 7.1.2.

Removal of Outliers

To improve the robustness of the scagnostic measures, outliers are removed from the configuration. While there are many existing results that flag potential outliers, many rely on a distributional assumption of the data. To avoid this, outliers are detected by peeling the minimal spanning tree. A node is considered an *outlier* if all minimal spanning tree edges attached to the node exceed a value w

$$w = q_{75} + 1.5(q_{75} - q_{25})$$

where q_{75} and q_{25} are the 75^{th} and 25^{th} percentiles of the minimal spanning tree edge weight distribution.

The peeling of the minimal spanning tree is an iterative process. Beginning with the minimal spanning tree, $\{\mathbf{V}, \mathbf{E}\}$, on the entire configuration, the value of w is determined. Any node $\mathbf{v}_i$ with all edges $\mathbf{e}_{ij} > w$ are removed from the configuration. This process is repeated by calculating the minimal spanning tree on the remaining nodes and recalculating w until no node is identified as an outlier.

This definition of outlying avoids the need to assume the data come from any particular distribution, allowing for more robust outlier detection if the assumptions are not met. Any vertex that meets the above criteria is removed from the data set after the outlying scagnostic has been computed, and the final minimal spanning tree produced is used in the computation of all but the outlying scagnostic.

2.3.2 Measures

Once the data has been pre-processed, geometric graphs are computed on the configuration. The geometric graphs used in the current implementation of scagnostics include the minimal spanning tree, alpha hull, and convex hull. The α value for computing the alpha hull is chosen to be the 90^{th} percentile of the minimal spanning tree edge lengths. From these geometric graphs, nine measures are computed which numerically summarize some of the structure in the configuration.

Outlying

The outlying scagnostic is defined as the total edge length contributed by those points that are identified as outliers to the total edge length of the minimal spanning tree

$$c_{outlying} = \frac{\sum T_{outliers}}{\sum T}$$

where the notation T denotes the edge lengths of the edges in the minimal spanning tree and $T_{outliers}$ denotes the edge lengths of the outlying nodes.

Skewed

The measure of skewness utilized depends on the non-parametric distribution of the edge lengths of the minimal spanning tree. This provides a relatively robust measure when compared to the parametric measures that could have been used in their place, while avoiding the need to impose a distribution on the data. The skewed scagnostic is given by

$$q_{skew} = \frac{q_{90} - q_{50}}{q_{90} - q_{10}}$$

where q_p is the p^{th} percentile of the minimal spanning tree edge length distribution. This measure is then scaled to combat the fact that it decreases in n after binning

$$c_{skew} = 1 - \omega(1 - q_{skew})$$

where $\omega = .7 + \frac{.3}{1+t^2}$, $t = n/500$, is the weight function defined by [120].

Sparse

The measure for sparse is given as the 90^{th} percentile of the minimal spanning tree edge length distribution, again scaled to combat bias in n

$$c_{sparse} = \omega q_{90}$$

Striated

The striated scagnostic seeks to capture "smooth" paths through the minimal spanning tree - such as a curve with a high signal-to-noise ratio. A non-parametric measure is again used, defining striation to be the proportion of adjacent edges in the minimal spanning tree with cosine less than -0.75 (equivalently, the angle between adjacent edges exceeds approximately 138°).

$$c_{striated} = \frac{1}{|V|} \sum_{v \in V^2} I(cos(\theta_{e(v,a)e(v,b)}) < -0.75)$$

where V^2 is the set of all vertices of degree two.

Clumpy

The clumpy scagnostic seeks to capture whether the points in the configuration are clustered, and is based on the RUNT statistic.

$$c_{clumpy} = max_j(1 - \frac{max_k(length(e_k))}{length(e_j)})$$

where e_j corresponds to a specific edge length in the minimal spanning tree, j runs over all edges in the minimal spanning tree, and k runs over all edges in R_j, which is defined to be the RUNT graph, the smaller of the two subsets of edges that are still connected to each of the two vertices in e_j after deleting edges in the minimal spanning tree with lengths greater than e_j.

Convex

The convex scagnostic is simply the ratio of the area of the alpha hull to the convex hull. Again, the weight factor is included to combat bias after binning.

$$c_{convex} = \omega\Big(\frac{area(A)}{area(H)}\Big)$$

where A represents the alpha hull, and H represents the convex hull.

Skinny

The skinny scagnostic is based on the ratio of the area of the alpha hull to its perimeter. Skinny shapes will tend to have a larger perimeter than area. This measure has been standardized so that a circle yields a value of 0 and a perfect square a value of 0.12.

$$c_{skinny} = 1 - \frac{\sqrt{4\pi \cdot area(A)}}{perimeter(A)}$$

Stringy

A stringy shape is defined as a skinny shape with no branches in the MST. As such, the number of vertices of degree 2 are counted and compared to the number of other vertices, less those of degree one:

$$c_{stringy} = \frac{|V^2|}{|V| - |V^1|}$$

This measure is cubed to adjust for negative skew in its conditional distribution on n [120].

Monotonic

The monotonic scagnostic is the square of Spearman's rank correlation, as the direction of correlation is not of interest.

$$c_{monotonic} = \rho^2$$

Straight

The straight scagnostic was proposed in [119] but later replaced with the stringy scagnostic. It aimed to measure the *linearity* of the configuration by considering a modified form of graph dilation. The straight scagnostic is defined as the ratio between the Euclidean distance of the two end nodes of the longest shortest path of the minimal spanning tree to the actual distance of the same longest shortest path, known as the *diameter* of the minimal spanning tree

$$c_{straight} = \frac{||t_j - t_k||}{diameter(T)}$$

where t_j and t_k are the aforementioned end nodes on the longest shortest path. Note that this scagnostic is no longer included, but may be of interest in a future work project.

2.4 Additional Work on Scagnostics

The results presented above constitute the scagnostic framework that will be considered throughout this thesis. There are however a number of published works that look to expand on this framework. They are presented here for completeness.

2.4.1 Transforming Scagnostics

While scagnostics are designed to capture interesting structure in a configuration, if the structure is obscured by high density regions, scagnostics will fail to capture it. This problem is addressed in [27] by first using a set of nonlinear transformations on the data, and then applying scagnostics. The following transformations are considered

- None: $x^* = x$ (leaves the points unchanged)
- Half: $x^* = x/2$ (squeezes points together)
- Square: $x^* = x^2$ (pulls points towards the left of the frame)
- Square Root: $x^* = \sqrt{x}$ (mildly pulls points towards the right of the frame)
- Log: $x^* = log(x)$ (strongly pulls points towards the right of the frame)
- Inverse: $x^* = 1/x$ (reverses scale and squeezes points towards the left of the frame)
- Logit: $x^* = (log(x/(1-x)) + 10)/20$ (squeezes points towards the middle of the frame)
- Sigmoid: $x^* = 1/(1 + exp(-20x + 10))$ (expands points away from the middle of the frame)

Note that the parameters of the logit and sigmoid transformations are designed to make them symmetric about $y = x$. Each transformation is applied independently to the x and y axes, resulting in 64 total possibilities to consider.

To illustrate their methods, the Madelon data set [31] is used, and more specifically the point configuration obtained by plotting variable 41 versus variable 48 [27]. They apply each data transformation independently to the x and y axes, and compute the striation scagnostic on the data. They note that clear striation becomes visible in the data only when the sigmoid function is applied to the x axis. Figure 2.6 illustrates the experiment.

In choosing a transformation, a scagnostic of interest is chosen and then used as a measure of goodness in selecting a transformation to be used. The transformation resulting in the most significant gain over the original configuration as measured by the scagnostic of interest is the transformation that is chosen for the data.

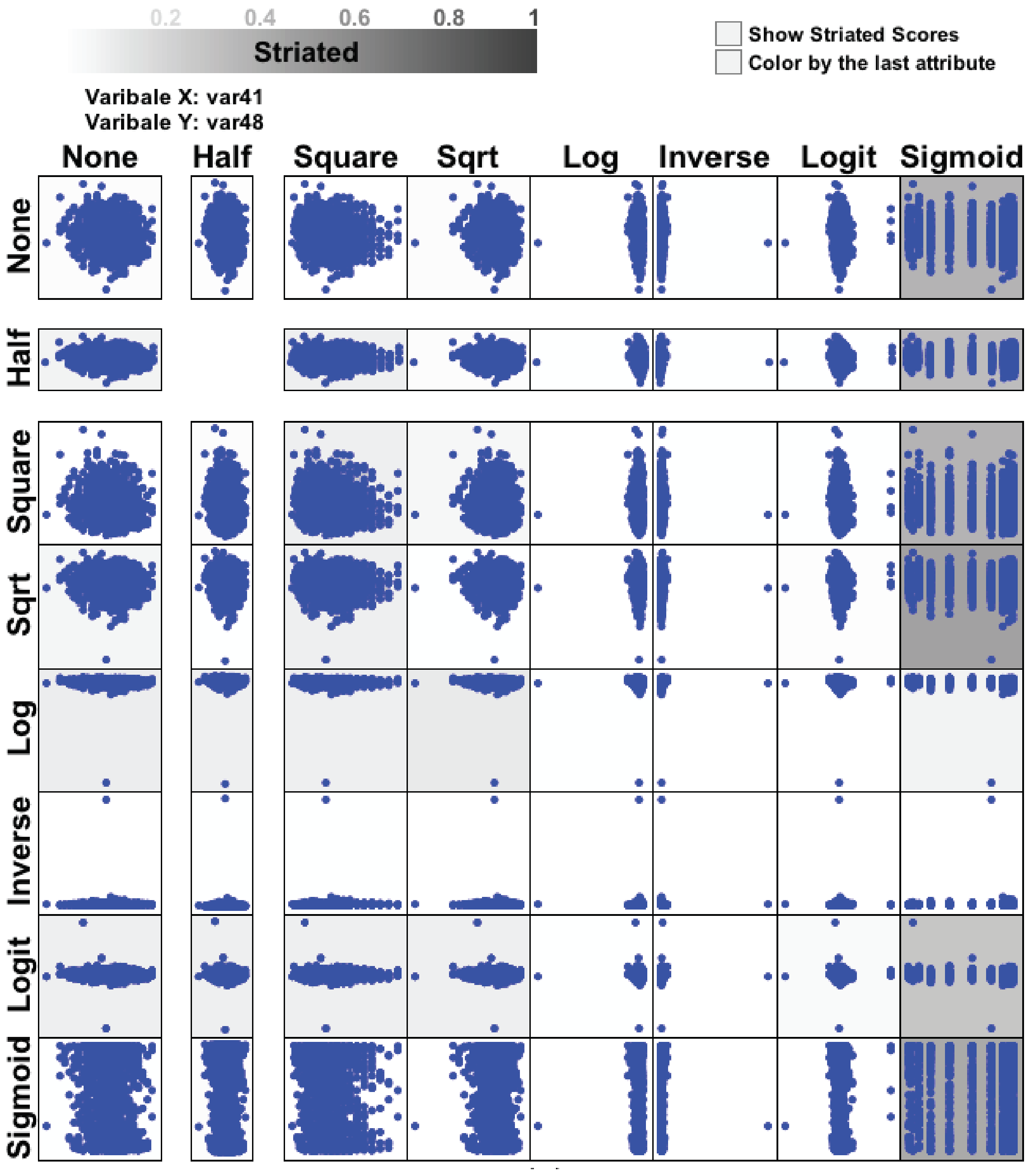


Figure 2.6: *From [27]: The result of computing striation on the 64 different data configurations produced by transforming the 41st and 48th variables in the Madelon data set. Grey level indicates the level of measured striation, with darker grey representing higher striation. High levels of striation can be observed when the sigmoid function is applied to the x axis variable.*

2.4.2 Pargnostics

While scagnostics compute measures on data arranged as a scatterplot, *pargnostics* [28] attempt to measure interesting structure in the data when the data are arranged as parallel coordinate plots. In addition, pargnostics can be used to order the axes in the parallel coordinate plot to decrease clutter and improve readability.

Like scagnostics, pargnostics begin by binning the data, separating each axis into h equally sized bins. With h bins (or pixel coordinates) on each axis, the number of points in bin i on the left axis and bin j on the right axis can be counted as

$$b_{ij} = |\{k \mid \lfloor l_k \rfloor = i \cap \lfloor r_k \rfloor = j\}|$$

where l_k is the pixel coordinate of the left axis, and r_k is the pixel coordinate of the right axis. With these counts, a number of measures of interest can be computed.

Number of Line Crossings

Crossing lines in a parallel coordinate plot typically indicate inverse relationships between the axes. Giving the binning done above, the number of crossing lines in a parallel coordinate plot can be calculated as

$$L = \sum_{i=1}^{h-1} \sum_{j=1}^{h-1} \sum_{k=i+1}^{h} \sum_{l=j+1}^{h} b_{ij} b_{kl}$$

Since each line can intersect every other line at most one time, the number of crossings is bounded above by $\frac{n(n-1)}{2}$, where n is the number of lines in the plot. The normalized measure is then

$$L_{norm} = \frac{2L}{n(n-1)}$$

Angles of Crossing

The angle of crossing in a parallel coordinate plot is of interest for two reasons. First, lines that cross at shallow angles can be hard to track, leading to visual cluttering in the plot. Second, lines that may be part of the same cluster tend to cross at shallower angles [28].

Each pair of crossing lines forms two angles that add to 180°, and the smaller of the two is taken, creating a distribution of angles. The measure of interest is taken to be

$$A = q_{50}$$

where q_p represents the p^{th} percentile of the crossing angle distribution.

Parallelism

A parallel coordinate plot that has many parallel (or near parallel) lines could be indicative of a high degree of correlation between the dimensions. To measure the degree of parallelism, the vertical distance between any two connecting points on adjacent axes are measured, with positive values indicating that the line is going up, and negative values indicating the line is going down.

Axis pairs with high levels of parallelism will have narrow distance distributions, while wide distributions indicate little to no parallelism. The measure is therefore taken to be the interquartile range of the distance distribution (normalized by dividing through by the largest measured distance)

$$P_{norm} = 1 - |q_{75} - q_{50}|$$

where q_p is the p^{th} percentile of the distance distribution.

Mutual Information

Mutual information measures the general level of dependency between variables. Letting X and Y be random variables, mutual information is calculated as

$$I(X;Y) = \sum_{i=1}^{h} p(x_i, y_j) log\Big(\frac{p(x_i, y_j)}{p(x_i)p(y_j)}\Big)$$

For pargnostics, these probabilities are simply equal to the proportion of the data in each bin. For the joint probability $p(x_i, y_j) = \frac{b_{ij}}{n}$, and for the marginal probability, $p(x_i) = \frac{b_i}{n}$, where $b_i = |\{k \mid \lfloor l_k \rfloor = i\}|$.

Convergence and Divergence

A parallel coordinate plot is said to be convergent if many lines from the left axis converge on a few points on the right axis. This can be measured as

$$C = \sum_{i=1}^{h} \sum_{j=1}^{h} I(b_{ji} > 0)$$

Conversely, a parallel coordinate plot is said to be divergent if the plot has only a few points on the left axis and spreads out to several on the right. This can be measured as

$$D = \sum_{i=1}^{h} \sum_{j=1}^{h} I(b_{ij} > 0)$$

Over-Plotting

Over-plotting leads to visual cluttering, and can therefore be used as a measure of the quality of the visualization. Every bin that has more than one entry contributes to over-plotting. Therefore, an appropriate measure is

$$O = \sum_{i=1}^{h} \sum_{j=1}^{h} b_{ij} I(b_{ij} > 1)$$

This measure is normalized by the total number of points n

$$O_{norm} = \frac{O}{n}$$

An Example using Real Data

Pargnostics are computed on wine data [31], and are used to optimally organize it by minimizing the angle of crossing and maximizing parallelism between adjacent axes [28]. This is done using a branch-and-bound algorithm, and using the pargnostic measures as a cost. The results are shown in Figure 2.7 [28].

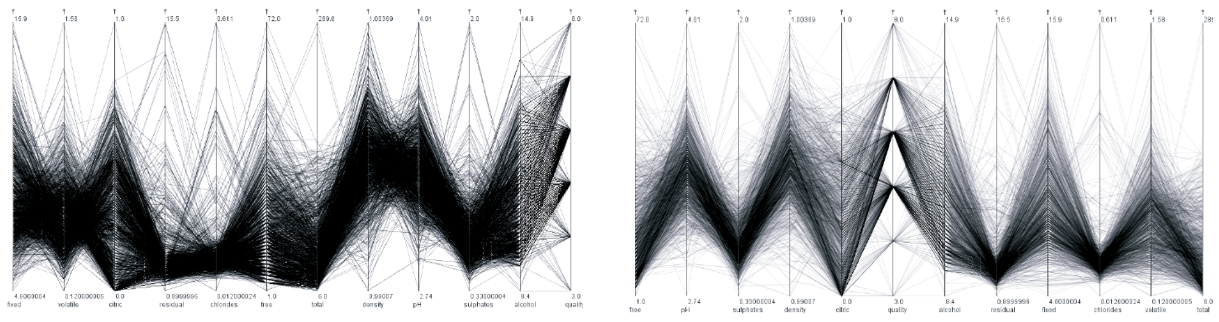


Figure 2.7: *From [28]: The result of sorting the axes of the wine data set in a parallel coordinate plot by minimizing the angle of crossing and maximizing parallelism. On left the original plot, and on right the optimally ordered plot.*

2.4.3 Density-based Cognostics

A very different approach to cognostics is taken in [17]. Instead of considering measures based on subgraphs of the Delaunay triangulation (which was the approach of [119]), density-based scagnostics consider computing cognostics by ignoring portions of the data using erosion and thinning techniques. Using erosion techniques, low density patterns can be produced by ignoring high density regions, which can call attention to infrequent events that are of interest [17].

Erosion and thinning are closely related. After being hexagonally binned, a configuration *erodes* as follows:

1. Begin with all cells under consideration.
2. For each cell, remove a number of counts from the cell in proportion to the number of exposed faces on the cell (0 - 6).
3. Cells with non-positive counts are removed from the configuration.
4. The number of exposed faces are updated each time a cell is removed.

Steps 2-4 are repeated until one cell remains. The last remaining cell in the configuration is deemed to be the *center* of the configuration.

Thinning [92] is very similar to erosion, with two key differences:

1. Cells are not eroded if they have only one neighbour.
2. Cells are not eroded if their removal would result in a set of disconnected cells.

Using thinning and erosion techniques, one pattern that draws attention is low density arms radiating out of high density regions [17]. The following cognostic is proposed to capture this type of behaviour

1. Identify areas of high density (i.e. high count cells) and exclude them from thinning.
2. Apply thinning to the remaining configuration, resulting in a skeletal structure.
3. Compute the measure of interest as

$$M = \frac{\text{Number of Cells in the Skeleton Structure}}{\text{Number of Cells under Consideration for Erosion}}$$

Figure 2.8 illustrates the type of structure being pursued. The image on the left is the original data, after applying hexagonal binning. Dark areas represent areas of high density. The picture on the right is the result of thinning the data. The dark center of the image (making up 96% of the data) was excluded from thinning. The measure of interest for this data was found to be 0.3.

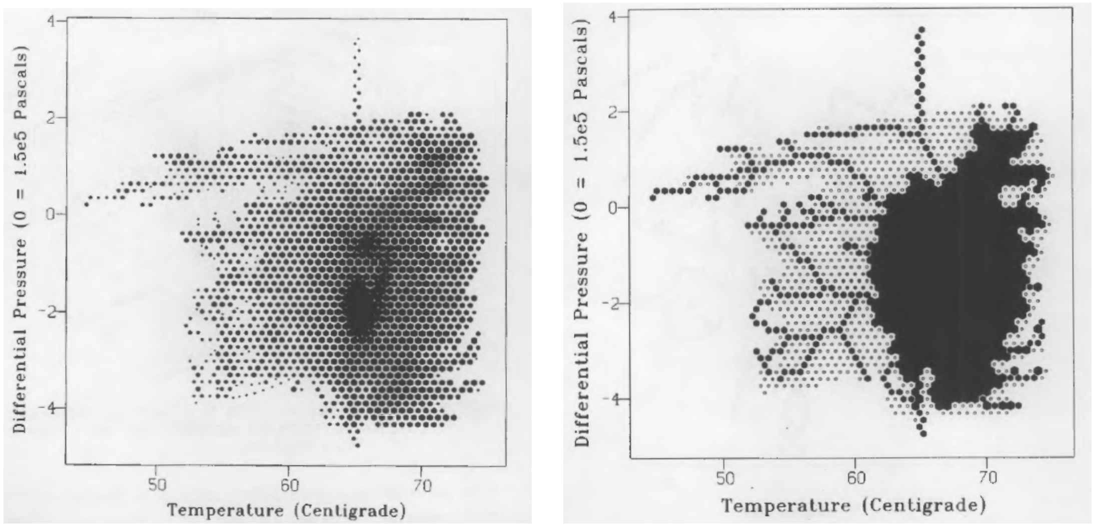


Figure 2.8: *From [17]: On left, the original data after applying hexagonal binning. On right, the result of thinning the configuration. Darker cells were retained during thinning, while the large central mass was not considered during the thinning process. The measure of interest is computed to be 0.3.*

2.5 Concluding Remarks

This chapter presented the background required to understand the methods used to capture both geometric and probabilistic structure. These methods were formalized in the scagnostic framework of [119], which was also presented in detail. This framework allows for a large number of two-dimensional point configurations to be assessed numerically in a short period of time, in turn allowing for more time to be spent on visually assessing those configurations which might be considered interesting.

The current scagnostic framework provides a rich environment to explore the structure present in many point configurations. The experiments performed in [119] & [120] provide more detail surrounding the types of structure that each scagnostic measure looks to capture, as well as a number of properties that are shared by all measures. The experiments also give insight into potential issues with the current implementation that warrant further exploration. These experiments will be discussed in detail and replicated in Chapter 3, and additional experiments will be conducted. Additionally, possible extensions of the scagnostic framework will be discussed.

Chapter 3

On Scagnostics

This chapter explores in more detail the current scagnostic framework. Section 3.1 repeats some of the experiments of [119][120], and also presents some new experiments which give important insight into the current scagnostic framework. Section 3.2 examines the inability to achieve the full range of [0,1] values that some scagnostic measures experience. Section 3.3 explores point configurations with interesting structure that is not captured by the current implementation of scagnostics. Finally, Section 3.4 explores the two-dimensional limitation of the current scagnostic framework, and briefly discusses some of the challenges associated with computing scagnostics in a higher dimensional space and why these measures may be of interest.

3.1 On the Distribution of Scagnostic Measures

As part of the discussion presented in [120], scagnostic measures on ten point configurations of varying sample size are computed to gauge the effect of sample size on the scagnostic measures. A similar analysis is undertaken here. From [120], the ten configurations under consideration will be

1. Bivariate Uniform
2. Spherical (Spherical Normal)
3. Binormal (bivariate normal with $\rho = 0.6$)
4. Funnel (bivariate log-normal with $\rho = 0.6$)
5. Exponential (exponential function plus random error)
6. Quadratic (negative quadratic function plus random error)

7. Clustered (three separated spherical normals at the vertices of an equilateral triangle)
8. Doughnut (two polar uniform separated by a moat of white space)
9. Stripe (product of Uniform and integer [1,5])
10. Sparse (product of integer [1,3] with itself)

Figure 3.1 provides a 500 point example of each of the ten configurations. Note that the scaling applied during the pre-processing step of the scagnostic framework has also been applied to each configuration, confining each to the unit square.

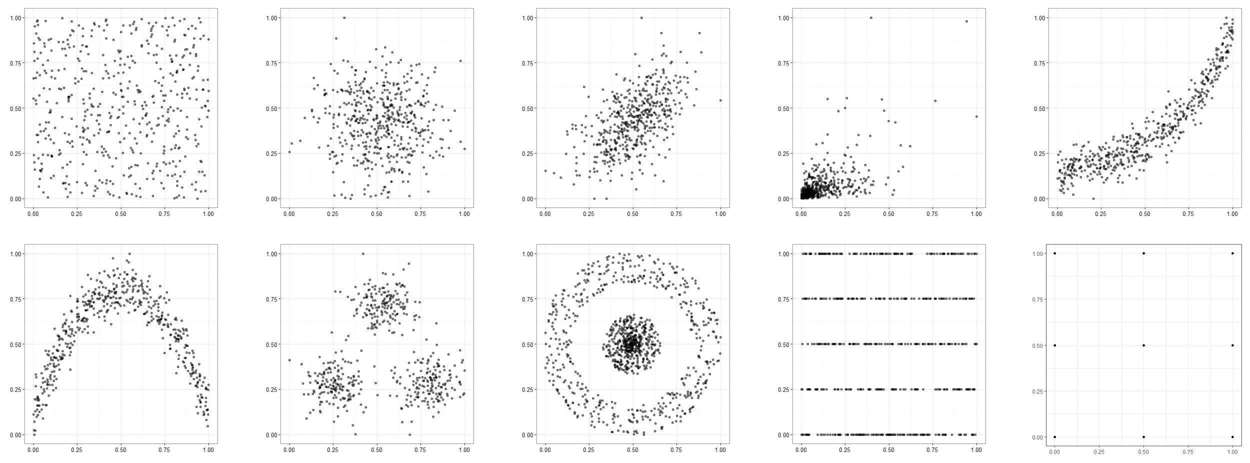


Figure 3.1: *A visualization of the point configurations to be used to perform a numerical analysis of the distribution of the various scagnostic measures.*

The first experiment presented in [120] creates 100 samples of size 100, 200, 300, 400, 500, 600, 700, 800, and 900 of each of the ten configurations. The aim is to observe the effect of sample size on the scagnostic measures, with the intention of showing that the measures don't tend to change with sample size. The original results are shown in Figure 3.2, and the reproduced results in Figure 3.3.

While mostly similarities are observed between the two images, some differences are evident. Difference are due in part to the inability to exactly replicate each point configuration in the original experiment. For instance, the exponential distribution used in the original experiment reads "Exponential function with random error". This leads to two questions - how was the exponential function produced, and how is the random error produced. For the reproduction in Figure 3.3, the exponential plot was created by simulating 500 x-axis values from a Uniform[0,3] configuration, and producing the y-axis values as $e^x + \mathcal{N}(0,3)$. Using a simple exponential function seemed reasonable, and the choice of simulating from Uniform[0,3] was to accentuate the exponential relationship between x and y. Normal(0,3) noise was chosen to tone down the monotonicity of

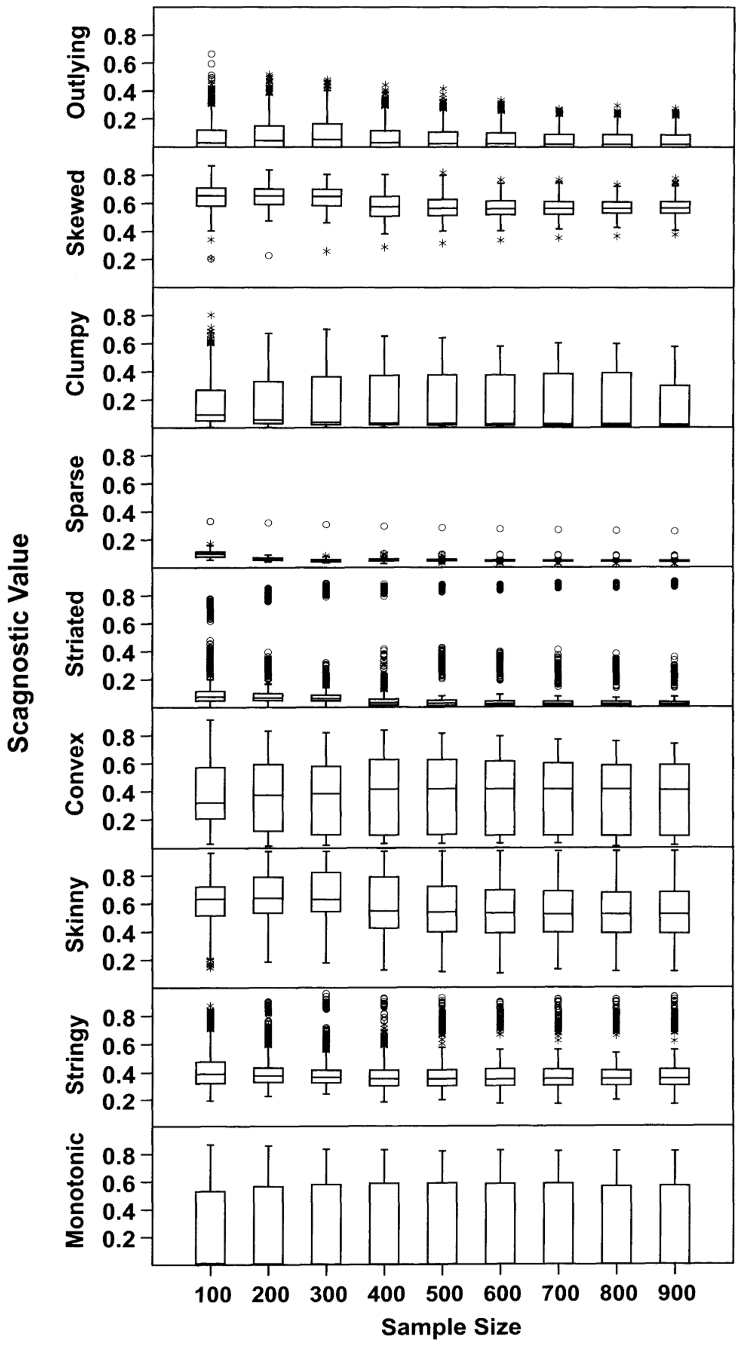


Figure 3.2: *The original scagnostic consistency experiment of [120]: "Boxplots of scagnostics measures for different samples from a wide variety of distributions. The horizontal axis represents sample size. The vertical axis is stratified by type of scagnostic. Scagnostic values vary between 0 and 1. The general lack of an overall trend across sample size indicates the scagnostic measures are consistent."*

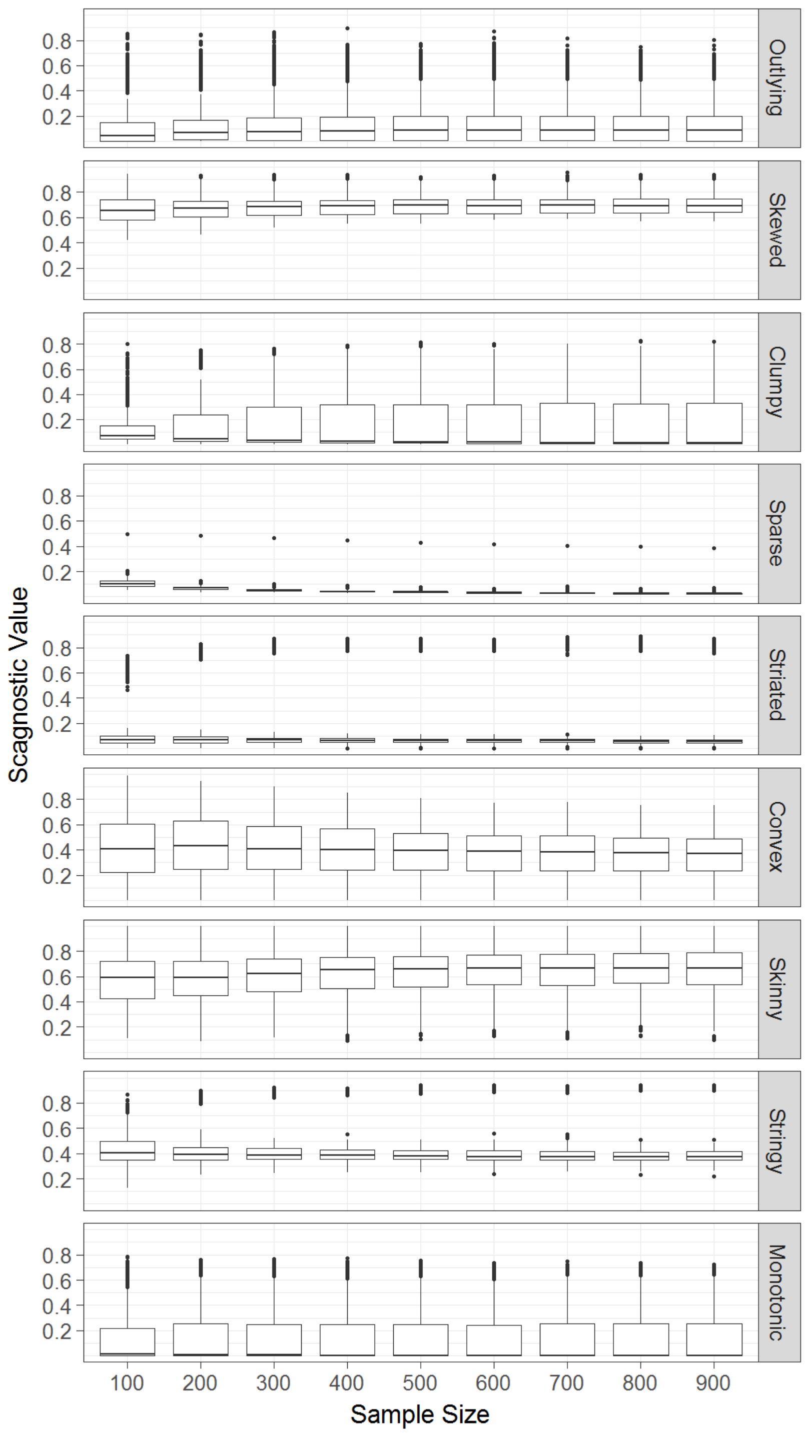


Figure 3.3: *The reproduced scagnostic consistency experiment. Despite the inability to reproduce the original configurations exactly, the observed results are very similar to the original experiment.*

the configuration - from the original image, there were very few (if any) configurations with a monotonicity above 0.8, and using Normal(0,1) noise (for example) resulted in almost every exponential plot having monotonicity above 0.8. It is for reasons like this that small differences between plots are observed, but overall, the two plots are fairly similar, indicating that the configurations considered are reasonably close to the original.

Despite these small differences, both plots tell a similar story. Varying the number of points between 100 and 900 has very little effect on the scagnostic measures computed. This can be seen by the relatively unchanged boxplots as the sample size increases. Another observation here is that, overall, the adjustments made to combat biases due to binning seem to have been effective - there is no apparent bias in any of the box plots that would indicate binning is adversely affecting the measures as sample size increases.

While observing the change in scagnostic measures as sample size increases is certainly a worthwhile experiment, the methodology used leads to two additional experiments that could give insight into the strengths and weaknesses of the current scagnostic framework.

First, recall that each boxplot computed in Figure 3.2 for any given sample size contains scagnostic measures from ten very different configurations. An interesting extension to this experiment is to view the empirical distribution of the scagnostics on each configuration individually, which allows the range of each scagnostic to be observed. This in turn gives insight into the type of structure that each scagnostic is designed to capture. For example, one would expect the distribution of the monotonic scagnostic measure to be clustered at high value for distributions such as the exponential distribution, and low values for distributions such as the normal or uniform. Observing similar trends for the other scagnostics could be quite informative.

To execute such an experiment, 100 samples of size 500 for each of the ten configurations are produced randomly and the nine scagnostic measures are computed on each. To compare the value of the scagnostics between configurations, histograms of each scagnostic measure separated by point configuration are produced. Figures 3.4 - 3.12 illustrate the results.

The histograms of the individual scagnostics lead to some interesting insights about the various scagnostic measures. In particular, they allow for a more clear indication of what type of structure each is chasing. A similar idea is presented in [120], who present Figure 3.13 to demonstrate the types of structure each of the scagnostics are designed to capture.

The most obvious, of course, is the measure of monotonicity. It is highest for the exponential distribution, moderate for the log-normal and bivariate normal, and low for every other configuration. This fits perfectly with what it's trying to capture. The stringy and striated scagnostics are also ones that emerge clearly in this experiment. They are both high for only one configuration - the stripe configuration. This helps solidify what the names imply - the striated scagnostic is searching for straight lines, while the stringy scagnostic is searching for lines in general.

While these images provide clarity for some of the scagnostics, there are others for which it provides more questions than answers. The most obvious issue, perhaps, lies with the skewed

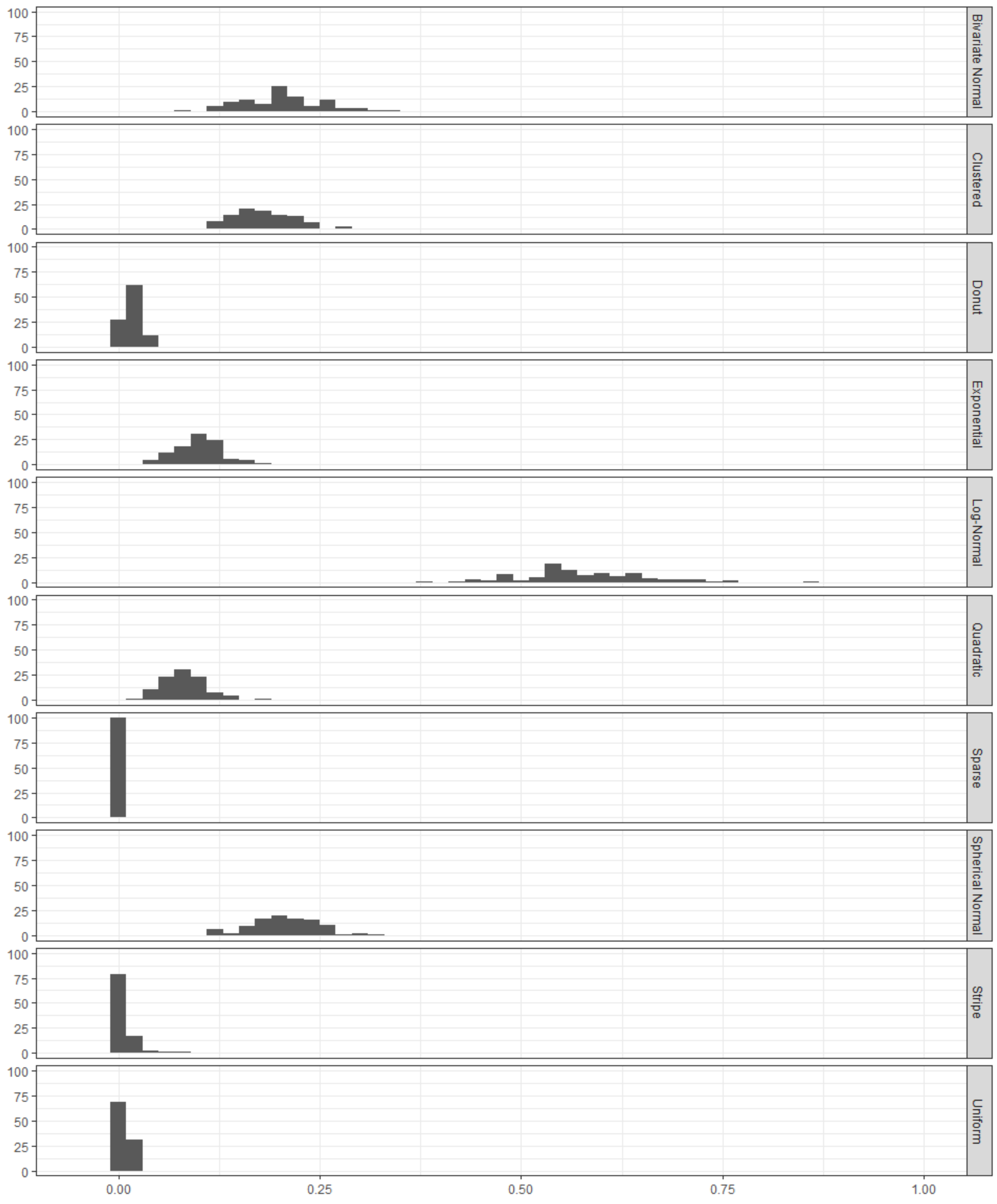


Figure 3.4: *The empirical distributions of the outlying scagnostic measured over the ten configurations listed in [120].*

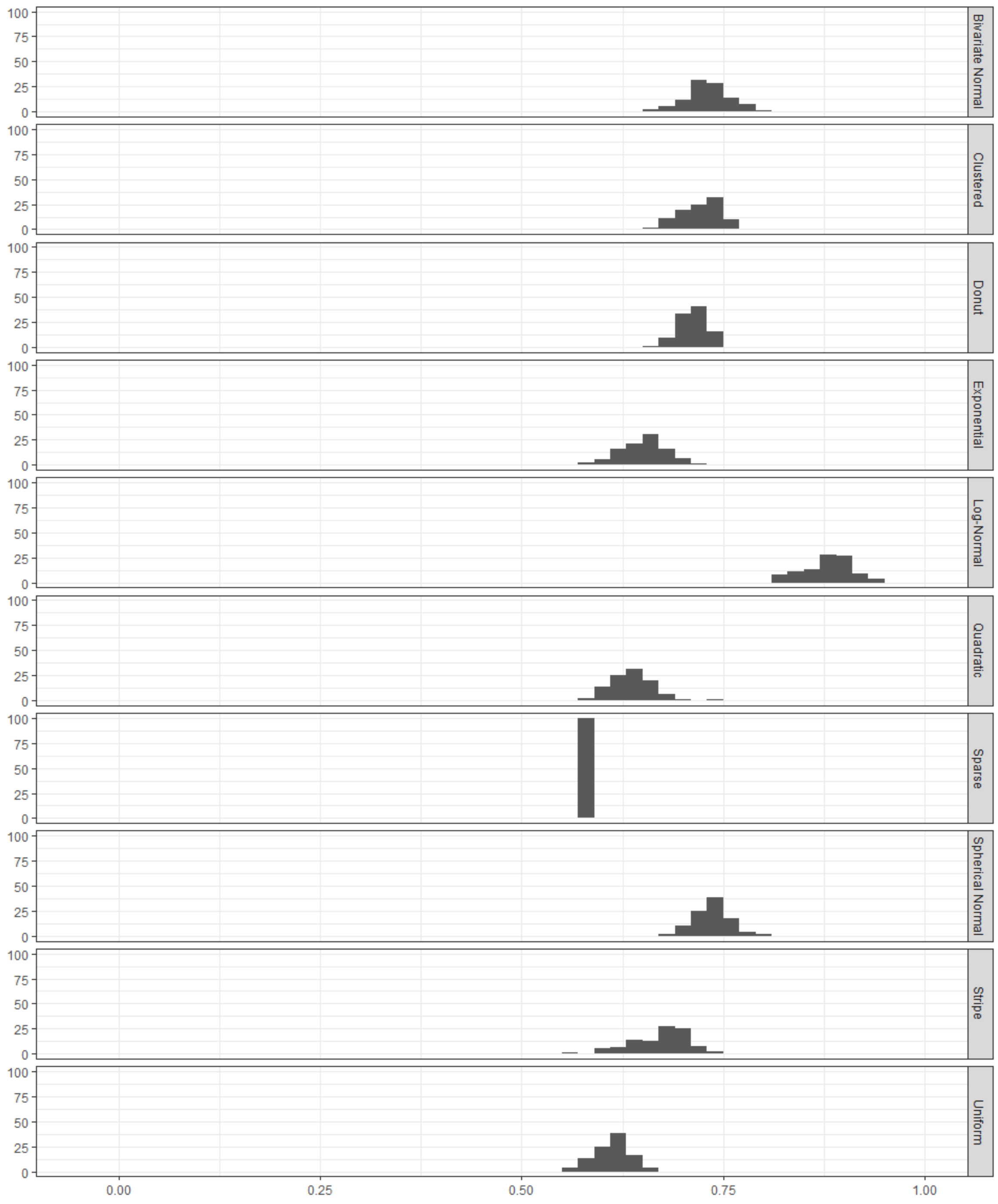


Figure 3.5: *The empirical distributions of the skewed scagnostic measured over the ten configurations listed in [120].*

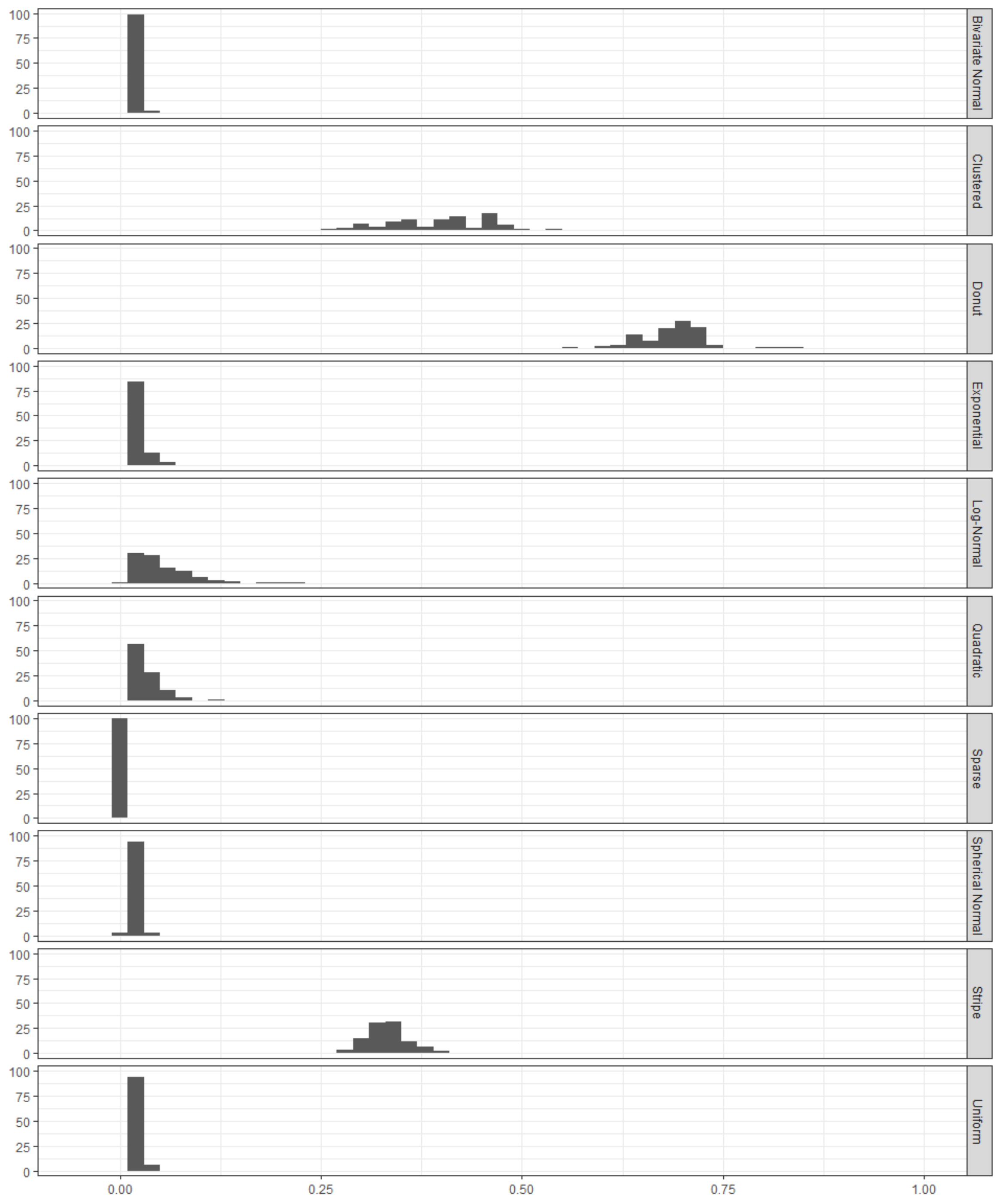


Figure 3.6: *The empirical distributions of the clumpy scagnostic measured over the ten configurations listed in [120].*

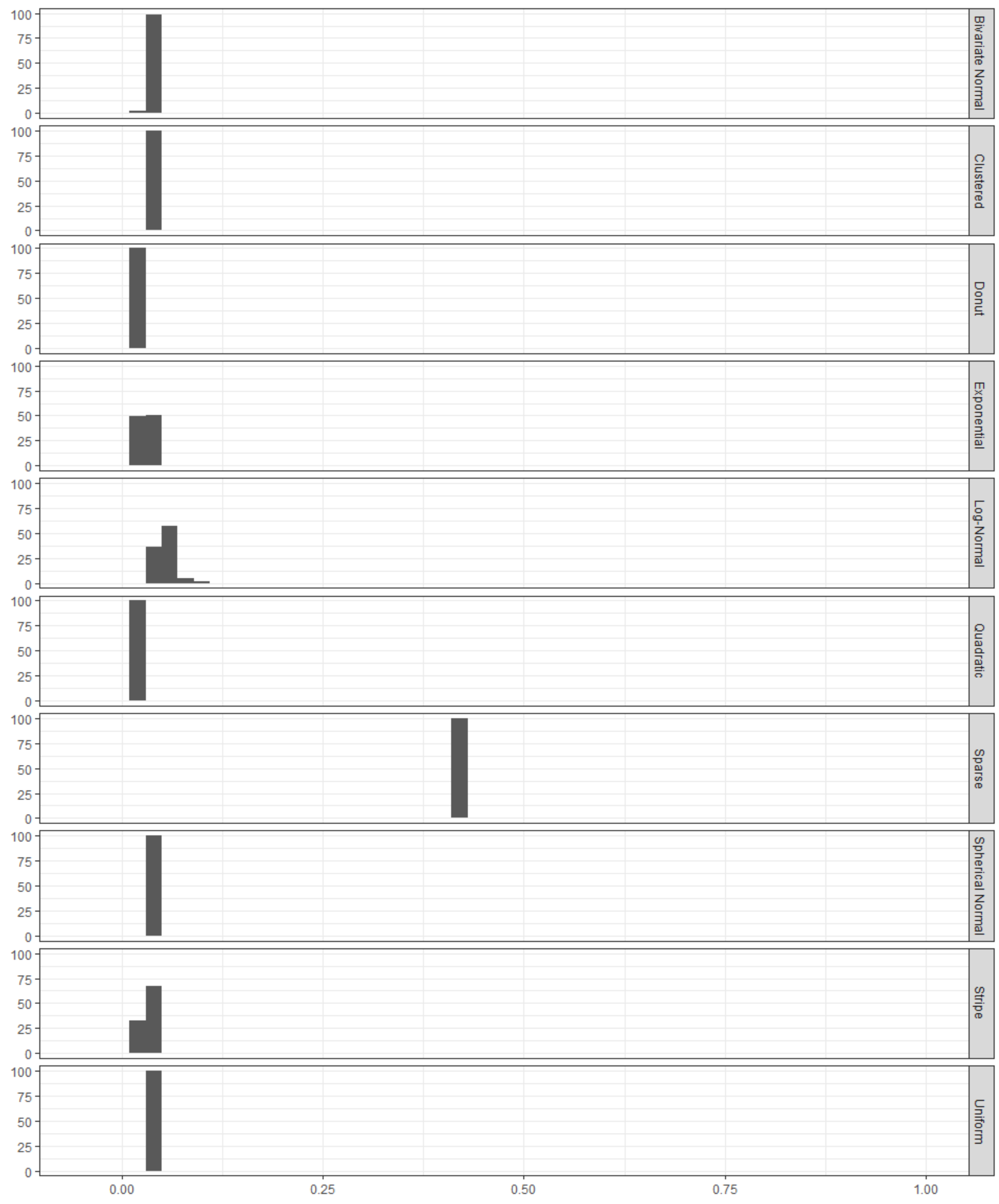


Figure 3.7: *The empirical distributions of the sparse scagnostic measured over the ten configurations listed in [120].*

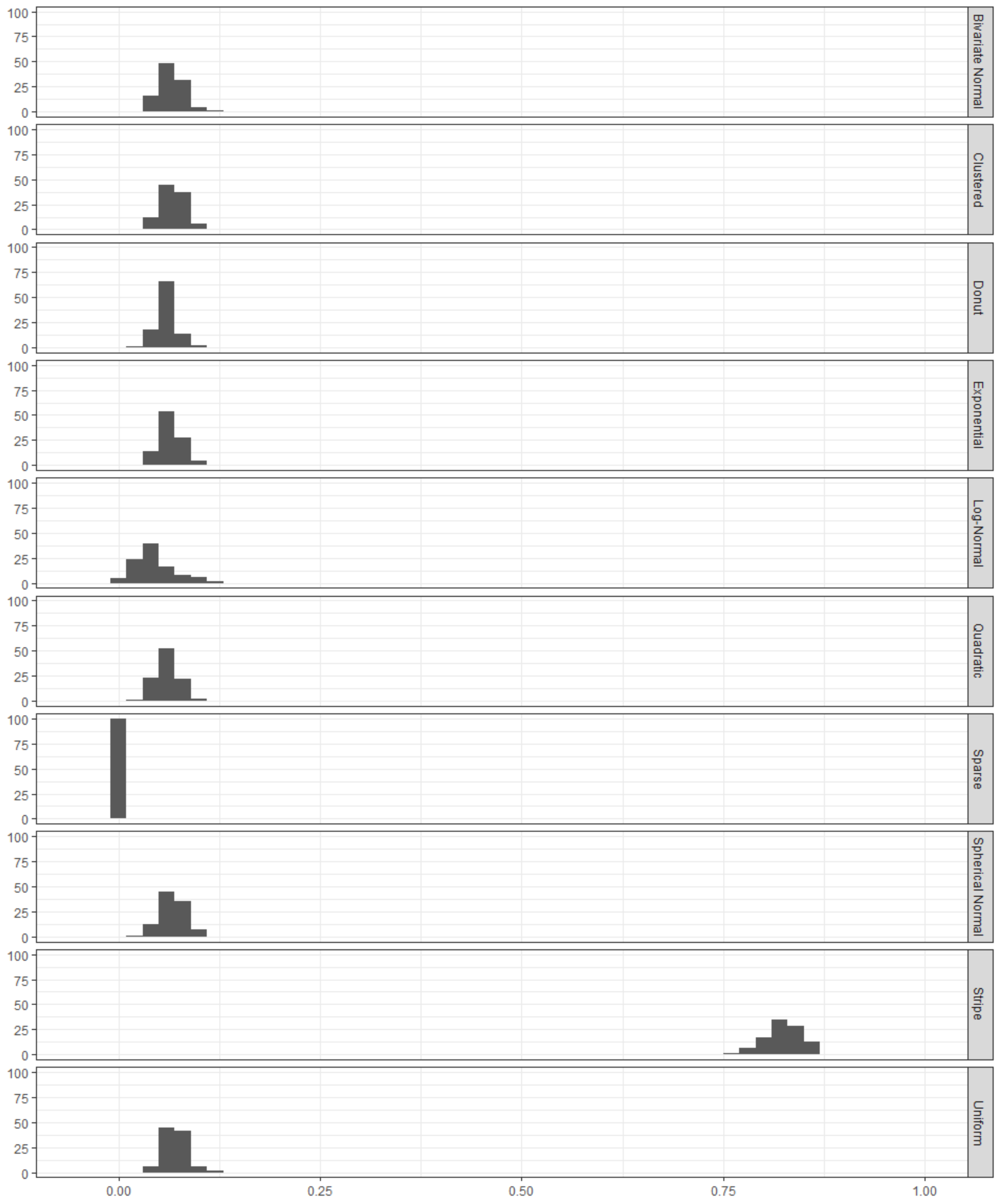


Figure 3.8: *The empirical distributions of the striated scagnostic measured over the ten configurations listed in [120].*

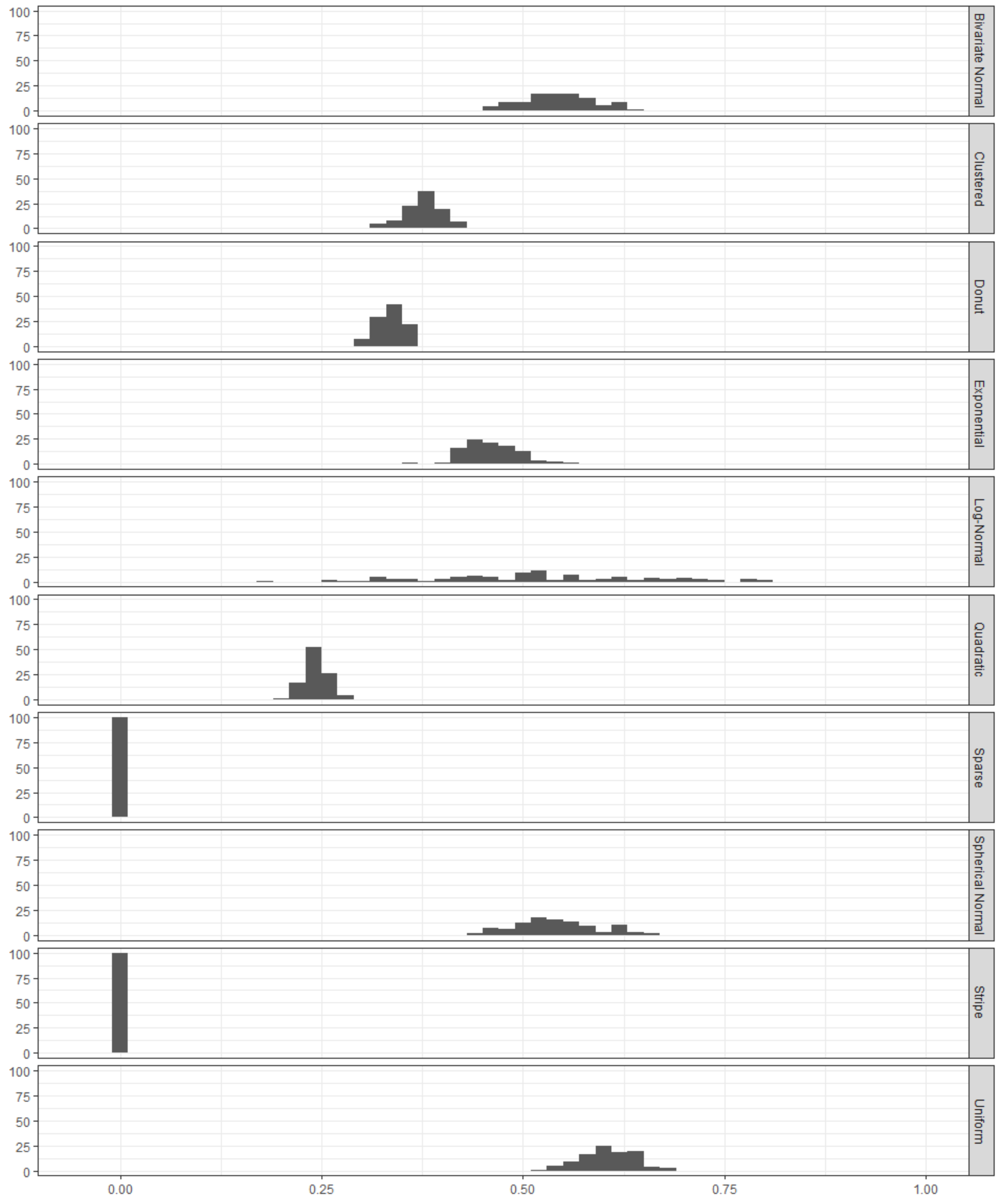


Figure 3.9: *The empirical distributions of the convex scagnostic measured over the ten configurations listed in [120].*

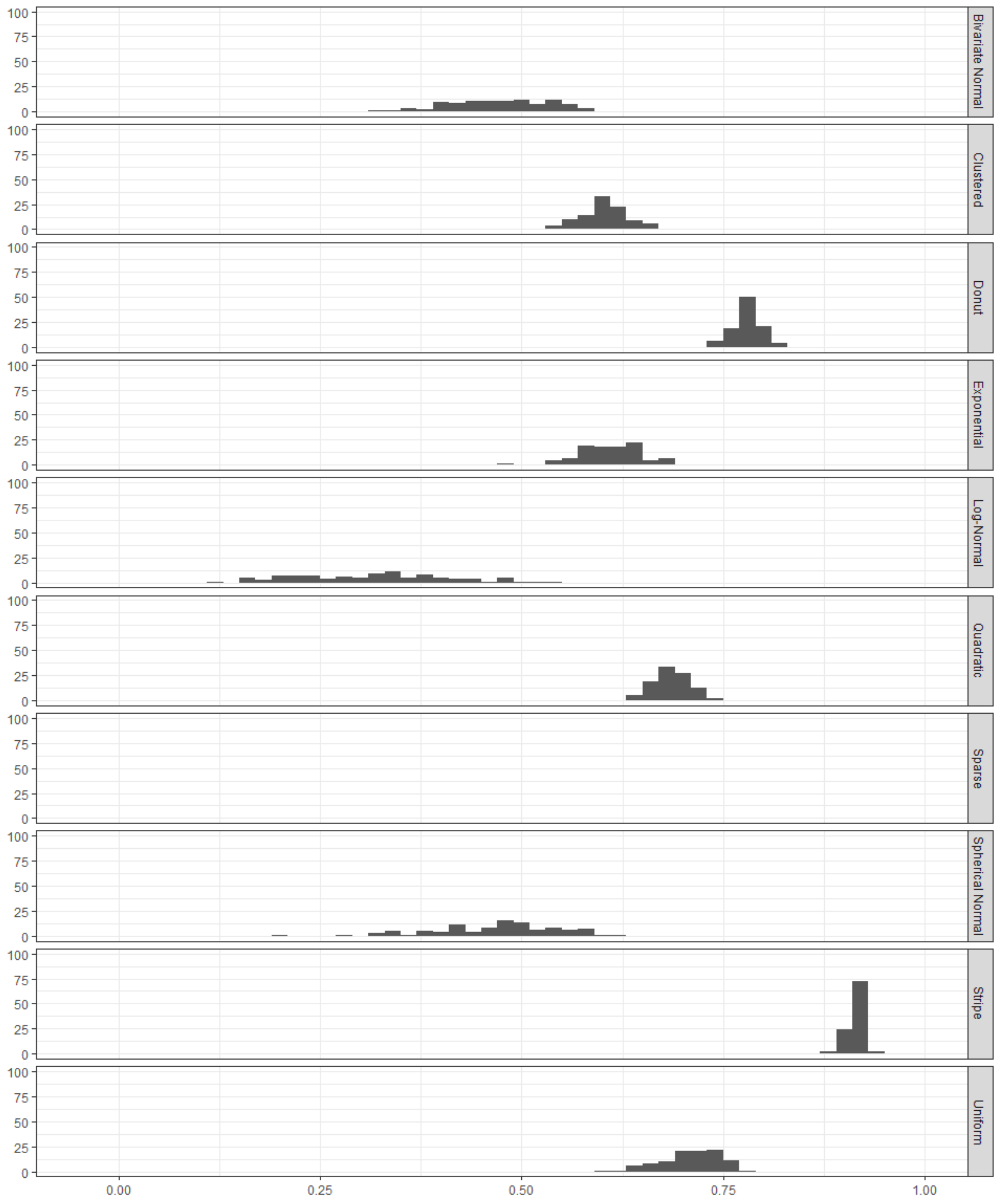


Figure 3.10: *The empirical distributions of the skinny scagnostic measured over the ten configurations listed in [120].*

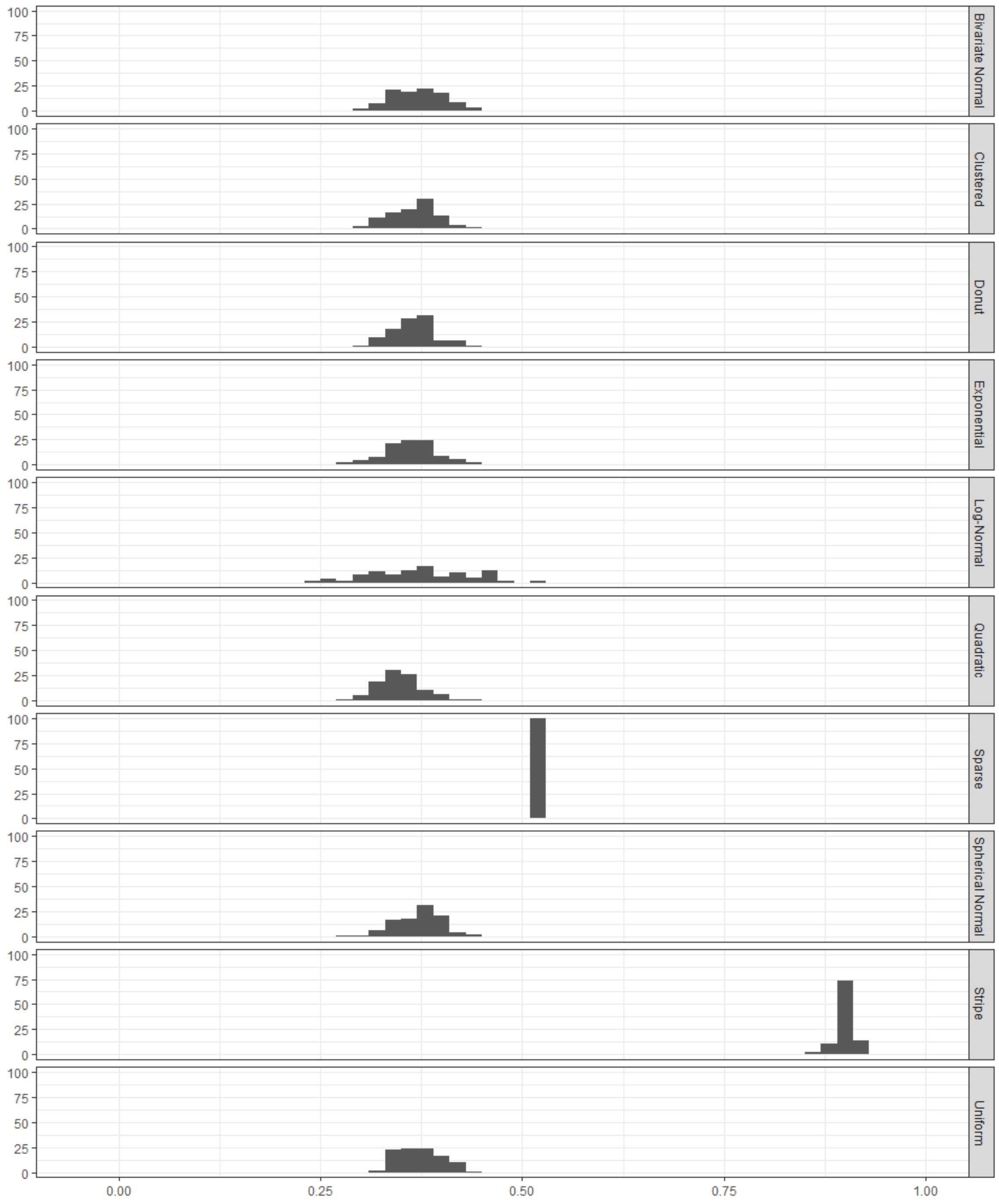


Figure 3.11: *The empirical distributions of the stringy scagnostic measured over the ten configurations listed in [120].*

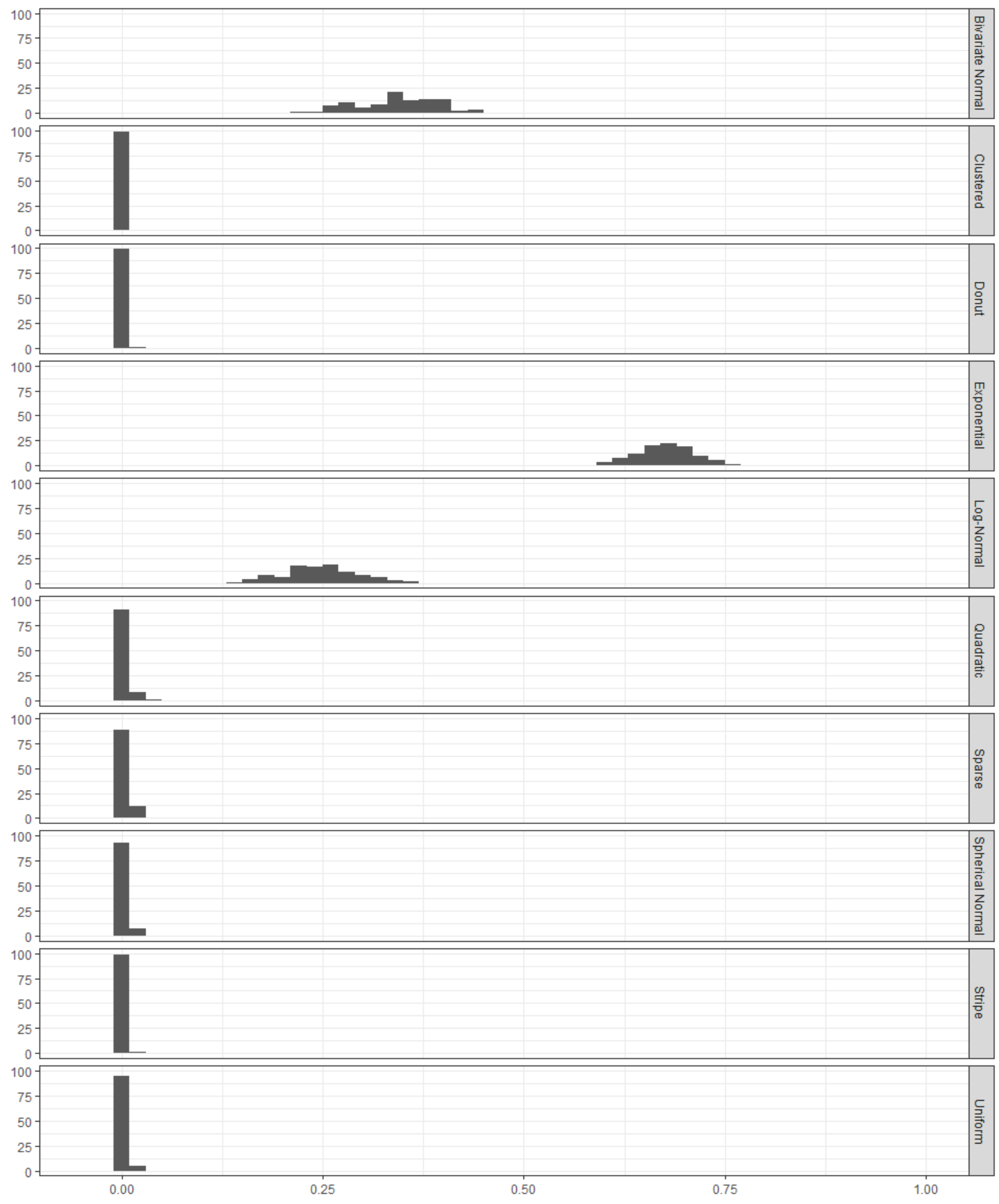


Figure 3.12: *The empirical distributions of the monotonic scagnostic measured over the ten configurations listed in [120].*

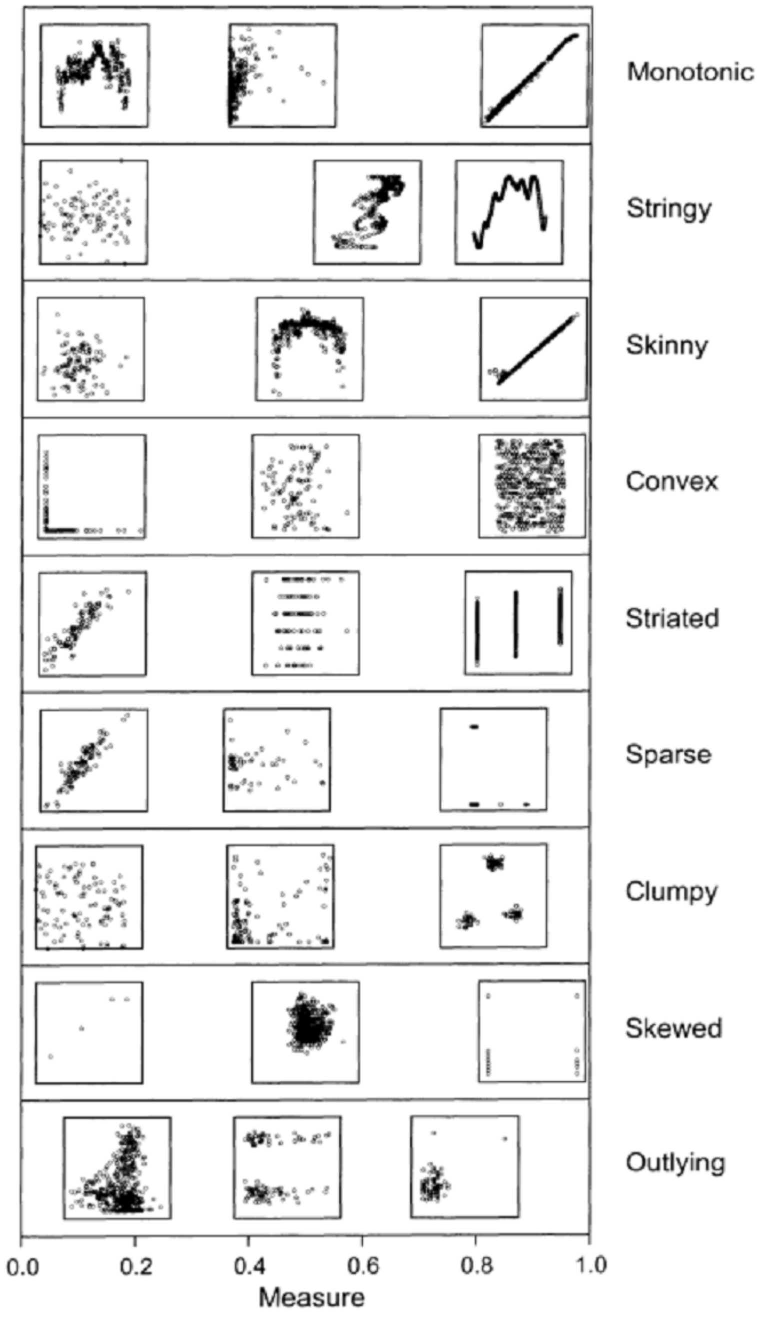


Figure 3.13: *From [120]: "Scatterplots from a variety of real data sets aligned on scagnostics scale for each scagnostic. The scatterplots were selected for having a relatively low, medium, or high value on each scagnostic. This figure shows that high-value scatterplots are reasonable exemplars for the descriptive names (Monotonic, Stringy, etc.) and low-value scatterplots correspondingly lack the feature described by each scatterplot name."*

scagnostic. From Figure 3.5, the skewed scagnostic does not have a single value (across any configuration) that is below 0.5. This is a problem if the scagnostic is to be used - if "low" skewness configurations are never encountered, what then is the significance of "high" skewness? Examining Figure 3.13, the configuration representing a low skewed value is quite underwhelming, as it contains only four unique points. Combined, these observations bring in to question the legitimacy of the skewed scagnostic. It also raises an interesting question - what does a configuration containing 100 unique points (or more) with a low value of skewed look like?

Another problematic scagnostic, although not as blatant perhaps as the skewed scagnostic, is the skinny scagnostic. The skinny scagnostic measures the ratio of the perimeter of the configuration to its area, and states that a square should yield a value near 0.12, while a skinny polygon should yield a value near 1.0. Counter-intuitively then, referring to Figure 3.10, the skinny scagnostic of the uniform distribution is in the $[0.6, 0.8]$ range - clearly not all that close to 0.12, despite the configuration being very similar to a square. Similar issues are clear in both the donut and stripe data sets - neither would be categorized as skinny, yet they have exceptionally high values of the skinny scagnostic. Like the skewed scagnostic, this brings the legitimacy of the skinny scagnostic into question, and further investigation is warranted. This will be undertaken in Chapter 10.

The second experiment looks to simply broaden the range of the original experiment performed in [120], where the chosen point configurations are designed to portray a wide range of important structure - representing, in effect, the "universe" of potential structure that could be encountered. Instead of artificially simulating data sets, instead consider the possibility of retrieving real data, and observing the range of scagnostics computed on them. This would, in effect, gauge the overall ability of each scagnostic measure to achieve its entire range of values. Given a sufficient "universe" of data, one would expect each scagnostic to be able to achieve values close to 0 and 1, indicating that the measure has identified configurations that both lack and have the structure they are designed to capture. Also of interest would be to see where the majority of the data lie for any given scagnostic, giving insight into when a rare event occurs for each scagnostic. For example, if the majority of the data in the "universe" have measured convexity in [0.5, 0.75], observing a value of 0.33 may be of interest, while observing a value of 0.66 may not.

The data which makes up the universe under consideration were retrieved from the UCI Machine Learning Repository [31], based on a very simple criteria

1. The data be in file format directly compatible with R (.csv/xls or .txt).
2. The data contains minimal missing values.

Using this criteria, 50 data sets were chosen and are listed in Appendix B. The data sets range in terms of dimension, the smallest data set has dimension 5, while the largest has dimension 281, and also in terms of size, the smallest data set has 100 observations, while the largest has 928,991. For each data set, scagnostics are computed on every pair of dimensions, resulting in a total of

91,331 observations for each scagnostic measure. The letter-value boxplot [57] for each scagnostic is shown in Figure 3.15.

Traditional boxplots create a display by visually displaying the median and upper and lower quartiles of the data as a box. Data that fall outside of this range are represented using a "whisker", which stretches from the first and third quartiles to $q_{25} - 1.5(q_{75} - q_{25})$ and $q_{75} + 1.5(q_{75} - q_{25})$ respectively. All data that fall outside of the range of the whiskers are considered to be outliers, and are plotted individually. For large data sets, this results in a disproportionately large number of outliers, making an accurate analysis of the boxplot difficult.

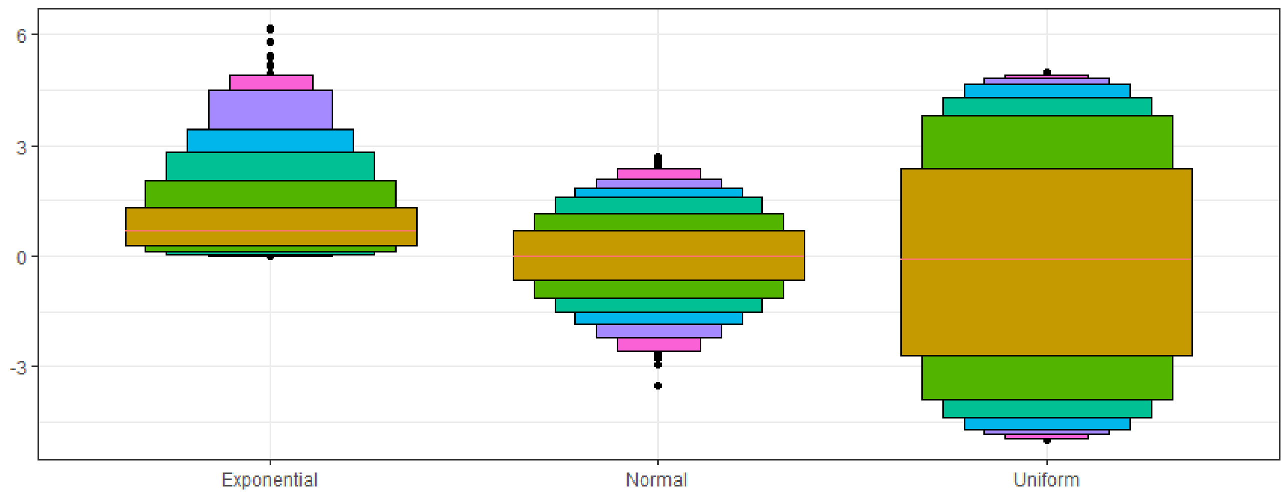


Figure 3.14: *Three examples of letter value boxplots for 1000 observations from Exponential(1), Normal(0,1), and Uniform(-5,5). Boxes of similar size and colour represent the same quantile range of the data.*

Letter value boxplots [57] are one proposed solution. In a traditional boxplot, a single box is created with boundaries at q_{25} and q_{75}. In letter value boxplots, several boxes are created, with boundaries at the *letter values* of the data. Letter values are the order statistics of the data at a given depth. The median, d_1, is defined as $d_1 = (1 + n)/2$. Successive depths are defined recursively, $d_i = (1 + \lfloor d_{i-1} \rfloor)/2$, and the lower and upper letter values are then defined as $L_i = X_{d_i}$ and $U_i = X_{n-d_i+1}$. Figure 3.14 provides examples of letter value boxplots for a few well known distributions. Note that in each, boxes that are the same size and colour represent the same range of data .

The results of the scagnostic universe experiment provide some interesting insight into the distribution of the scagnostics. The outlying, clumpy, sparse, striated, stringy, and monotonic scagnostics follow very closely to what might be expected - the entire range from [0,1] is mostly covered, with fewer occurrences in (at least) one of the two tails. The letter-value boxplots for skewed, convex, and (to a lesser extent) skinny seem to indicate potential issues with their respective scagnostics, as they don't come near the extreme value at one of the tails. For the skewed

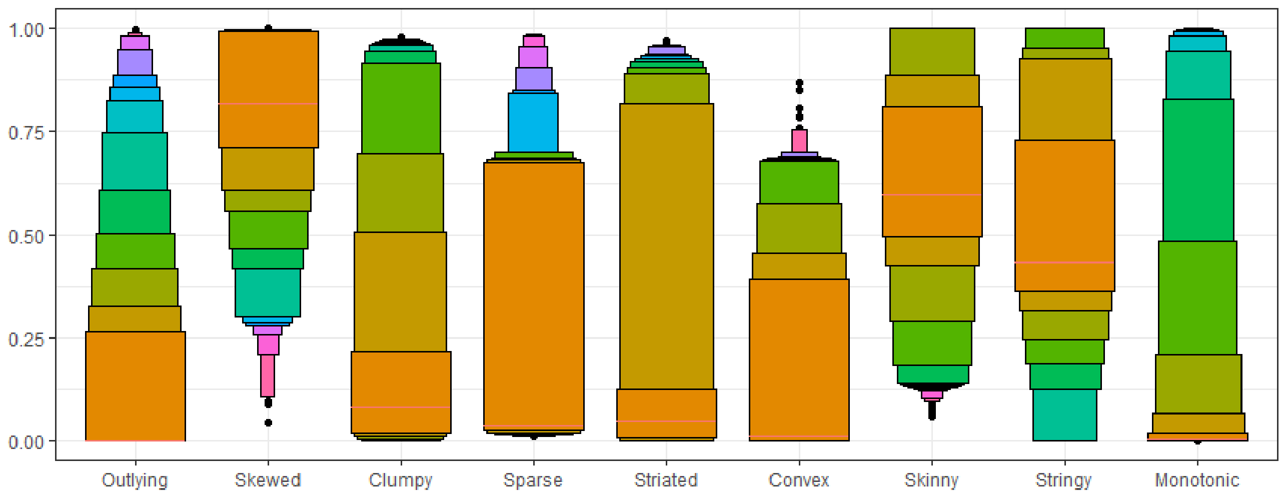


Figure 3.15: *The letter-value boxplots for each scagnostic measure computed on the "scagnostic universe" data. Ideally, each scagnostic would be able to achieve a full range of values from [0,1], however, that appears to not be the case for the skewed and convex scagnostics.*

scagnostic, there are only 15 observations in the range of $[0, .25]$. If, in close to 100,000 point configurations, only 15 have been identified as having skewness below 0.25, this is an indication that these low values of skewness are either incredibly rare and something that should be considered interesting, or the way in which the scagnostic is computed is not actually capturing this end of the range. In either case, further investigation is certainly necessary, as at current it's not clear of what interest the skewed scagnostic is - 64,410 configurations had a value of skewed above 0.75, so clearly the upper range of the scagnostic does not represent a rare occurrence. Similar issues arise with both the convex scagnostic (only 6 values above 0.75), and to a lesser extent, the skinny scagnostic (the smallest value observed is 0.057, and there are only 67 observations below 0.125). Each of these scagnostics need to be examined to ensure they are capturing interesting structure. This is undertaken in Chapter 10.

3.2 On Extreme Index Values

The core concept of scagnostics revolves around identifying interesting structure in a configuration via extreme indices in the observed scagnostic measures. These extreme indices indicate supposed structure in the configurations that the scagnostics are designed to capture [120]. Each scagnostic, then, should be able to achieve values at these extremes. Figure 3.15 indicates that this may not be the case.

The two clear offenders in this case are the skewed and convex scagnostics. Examining Figure

3.15, in over 90,000 data configurations, only 15 have skewed values below 0.25, with the minimum observed value being 0.043. This is indicative of one of two things - either the low skewed distributions are incredibly rare and thus interesting, or the skewed scagnostic is inherently flawed. Viewing four configurations that have low values of skewness (Figure 3.16) should give some insight into the structure (or lack thereof) of the low skew configurations.

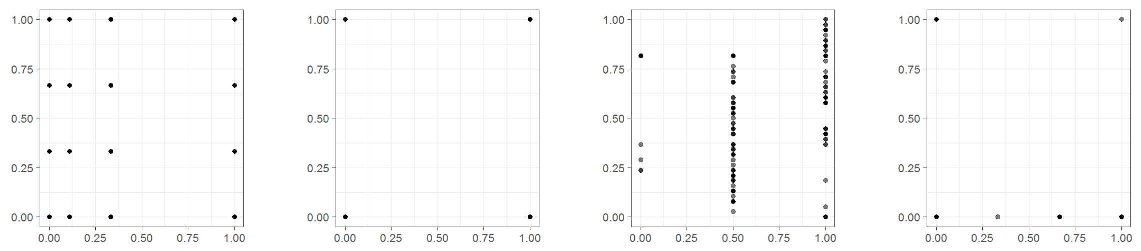


Figure 3.16: *Four configurations with measured skewness below 0.25 in the scagnostic universe experiment.*

From Figure 3.13, each low skewness configuration should lack the feature the skewed scagnostic is designed to capture. Examining the third image in figure 3.16 however, the third image looks very similar to the example of high skewness given in Figure 3.13. If both low skewed configuration exhibit the same type of structure has high skewed configurations, what type of structure is skewness attempting to capture?

The other scagnostic where extreme index values are not observed is the convex scagnostic, where only six configurations are observed to have a value above 0.75. Figure 3.17 illustrates the six configurations.

None of the configurations identified as having high levels of convexity appear to be particularly interesting. It appears as though the interesting structure in these configurations is better categorized by other scagnostics - for the first four, outlying appears to better define the interesting structure, while for the last two, striation and stringiness appear to be more accurate descriptions. In any case, it appears as though having a high level of convexity isn't particularly interesting.

In the case of either the skewed or convex scagnostics, the inability to identify interesting structure in the extremes of their bounds is problematic, as it is unclear then exactly what type of structure either is trying to capture. This leads to two potential avenues to explore

1. The scagnostic does not capture interesting structure that is not better summarized by another scagnostic measure.
2. The current mathematical definition of the scagnostic prevents adequate coverage of the range of the scagnostic.

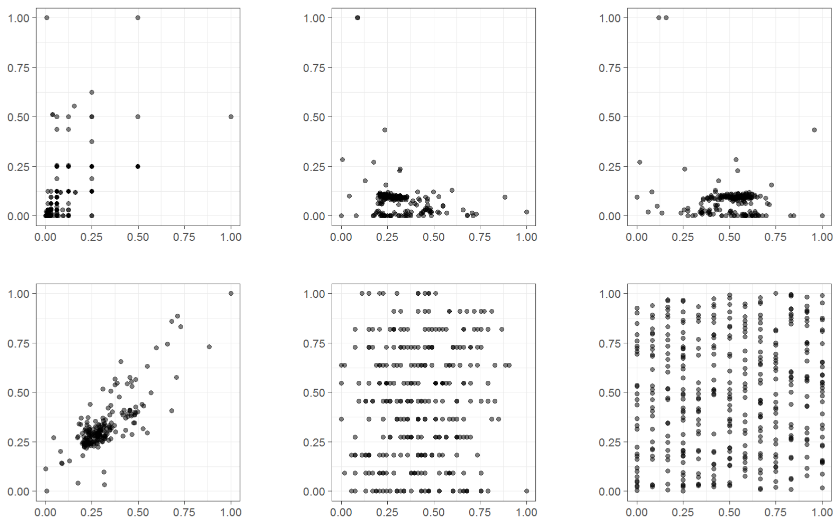


Figure 3.17: *Six configurations with measured convexity above 0.75 in the scagnostic universe experiment.*

Further exploration into the distribution of these scagnostics, as well as potential adjustments made to them, will be discussed in Chapter 10.

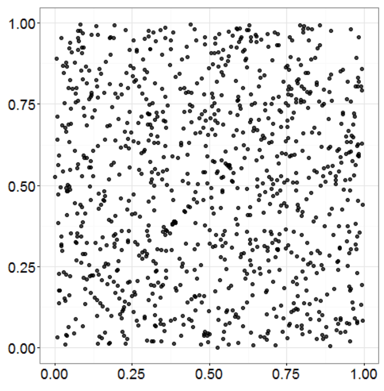
3.3 On Missing Structure

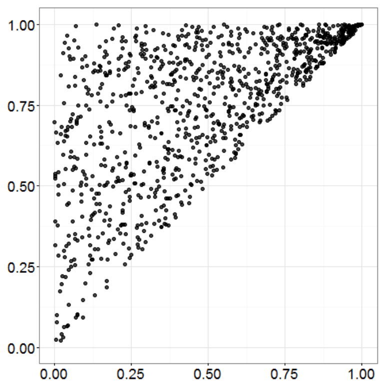
The current implementation of scagnostics computes nine measures designed to capture a wide variety of interesting structure. Naturally, however, some structure will not be captured. Consider the configurations and corresponding scagnostic summaries in Figure 3.18. Figure 3.18(a) represents a configuration that is perfectly symmetric about the line $y = x$, while Figure 3.18(b) represents perfect asymmetry about the same line.

Of the nine scagnostics in Figure 3.18(c), only monotonicity differs significantly between the two configurations. While monotonicity does differentiate the two configurations, it obviously does not distinguish between these two end cases of symmetry.

The distinction between these configurations is relevant, as in many applications the symmetry of the point configuration is very important. For instance, identification of symmetry in a configuration is a problem that occurs in fitting symmetric copulas, such as the t-copula [30]. Symmetric copulas are commonly used in fields such as portfolio theory when evaluating the relationships between the return of stocks.

Of course, symmetry isn't the only type of undiagnosed structure that may be of interest.


(a)


(b)

Scagnostic	3.18(a)	3.18(b)
Outlying	0.005	0.033
Skewed	0.648	0.624
Clumpy	0.014	0.015
Sparse	0.027	0.022
Striated	0.095	0.057
Convex	0.562	0.496
Skinny	0.781	0.752
Stringy	0.370	0.375
Monotonic	0.003	0.427

(c)

Figure 3.18: *In examining symmetry about the line Y=X, interest lies in identifying two cases, (a) perfect symmetry and (b) perfect asymmetry. (c) The scagnostic summaries for these two configurations, which seems to indicate that apart from monotonicity, these configurations are very similar.*

There is a wide variety of additional structure that could be pursued. Scagnostics to potentially capture this previously uncaptured structure are discussed in Chapter 10.

3.4 On Higher Dimensional Views

The original concept of cognostics [111] was to have a computer compute diagnostic measures on the $\binom{d}{2}$ pairs of a d-dimensional data set, allowing the user to choose to view only those two dimensional scatterplots that they would consider interesting. An equally valid interpretation is to use scagnostics as a tool to identify interesting configurations regardless of their dimension. That is, instead of choosing between two-dimensional views, scagnostics could be used to choose between three (or higher) dimensional views.

A natural extension then of the current scagnostic framework is to extend the concept to higher dimensional space. Many of the core concepts of scagnostics generalize quite naturally. For example, the minimal spanning tree, convex hull, and alpha hull all have higher dimensional analogues. There are, however, a number of issues that must be addressed for an effective scagnostic framework to exist in higher dimensional space.

Recall that before the geometric graphs are computed, the data are first pre-processed via scaling, binning, and removing outliers. The methods used to scale and remove outliers both generalize naturally to higher dimensional space. The ability to generalize the hexagonal binning algorithm is not as straight forward. With simple tasks such as the computation of the distance matrix having computational complexity dependent exponentially on the number of observations (and linearly on dimension), having an effective binning algorithm is crucial to the viability of scagnostics in higher dimensional space. A proposal for a binning algorithm that easily scales to higher dimensions will be made in Chapter 7.

Since each of the geometric graph objects used generalize to higher dimensions, once pre-processing has been accomplished, the next task is to generalize the scagnostic measures themselves. Many generalize quite easily - for instance, the measures of outlying, skewed, clumpy, sparse, convex, and stringy are already completely generalized, as they rely only on the computation of the underlying geometric graphs. Some of the measures require only slight tweaks to generalize easily to high dimensions. For example, the skinny scagnostic is scaled so that a two-dimensional circle yields a value of 0. As dimension increases, this scaling factor would need to be adjusted accordingly, if an n-sphere was to continue to have a value of 0.

Other measures, such as monotonicity (which uses Spearman's rank correlation) require thoughtful generalization to higher dimensions. Three dimensional scagnostics were explored in [41], who defined three dimensional monotonicity as the maximum of the associated two dimensional partial correlations - the correlation between two variables, while controlling for the effects of the third

$$\rho_{XY \cdot Z} = \frac{\rho_{XY} - \rho_{XZ}\rho_{YZ}}{\sqrt{1-\rho_{XZ}^2}\sqrt{1-\rho_{YZ}^2}}$$

This interpretation of three dimensional monotonicity, while potentially valid, is perhaps not ideal. Even though it controls for the effect of the third variable, it considers only the relationship between two variables at a time. This implies that if (for instance) (X, Y) are highly correlated, while (X, Z) and (Y, Z) are not, the monotonicity of the configuration would be considered to be high. A more desirable interpretation of monotonicity in high dimensions would be to consider the entire configuration at once.

While some scagnostics can be easily generalized to higher dimensions, perhaps a more important question to consider is if each scagnostic will continue to be relevant in higher dimensional spaces. For instance, is measuring the angle between branches in the minimal spanning tree still a relevant measure of striation in dimension larger than two? Is the RUNT dendrogram still an appropriate way to measure clumpiness? Is the upper fence of a boxplot an appropriate measure to determine if a point should be considered an outlier? Are there more interesting measures in higher dimensional space that could be considered? These are only some of the questions that will need to be addressed before an adequate generalization of scagnostics can exist in higher dimensional space. These questions will be explored as a future work project in Chapter 10.

Chapter 4

On Euclidean Distance Matrix Completion

This chapter begins by introducing the problem of Euclidean distance matrix completion via motivating example in Section 4.1. Section 4.2 introduces the background mathematics of the Euclidean distance matrix completion problem, as well as three solution methods found in the literature. Section 4.3 considers completions when the underlying Euclidean distance matrix is sparse, and more specifically, when the underlying Euclidean distance matrix represents a minimal spanning tree. Two new solutions are proposed to solve this problem.

4.1 Motivating Examples

Two applications of Euclidean distance matrix completion - sensor network localization and molecular conformation - were introduced in Chapter 2. Additionally, Figure 3.15 in Chapter 3 illustrated the difficulty of finding a point configuration that had a low value of the skewed scagnostic. Unable to find many such configurations, instead consider the possibility of *generating* a configuration with low skewness.

Recall that the skewed scagnostic is defined as

$$c_{skewed} = 1 - \omega\,(1 - \frac{q_{90} - q_{50}}{q_{90} - q_{10}})$$

Then, the problem of generating a configuration with a given level of skewness is equivalent to a problem with the following steps

1. Generate minimal spanning tree edge lengths with a given level of skewness.

2. Generate a minimal spanning tree preserving configuration from the minimal spanning tree edge lengths.

The first problem will be addressed in Section 4.4. Assuming it can be done, the second problem can be addressed by arranging the minimal spanning tree edge lengths as an undirected graph object. From the graph object, a *partial distance matrix*, such as that in Figure 4.1, can be created. A partial distance matrix is one in which all entries are either specified or unspecified, every diagonal element is zero, all specified entries are non-negative, and every complete principal submatrix is also a Euclidean distance matrix [67].

$$\begin{bmatrix} 0 & 0.56 & 0.98 & ? & ? & ? & ? & ? & ? & ? \\ 0.56 & 0 & ? & ? & ? & 0.82 & ? & ? & ? & ? \\ 0.98 & ? & 0 & ? & 0.66 & ? & ? & 0.87 & ? & ? \\ ? & ? & ? & 0 & ? & ? & ? & 0.81 & 0.23 & ? \\ ? & ? & 0.66 & ? & 0 & ? & ? & ? & ? & ? \\ ? & 0.82 & ? & ? & ? & 0 & 1.08 & ? & ? & ? \\ ? & ? & ? & ? & ? & 1.08 & 0 & ? & ? & ? \\ ? & ? & 0.87 & 0.81 & ? & ? & ? & 0 & ? & 0.68 \\ ? & ? & ? & 0.23 & ? & ? & ? & ? & 0 & ? \\ ? & ? & ? & ? & ? & ? & ? & 0.68 & ? & 0 \end{bmatrix}$$

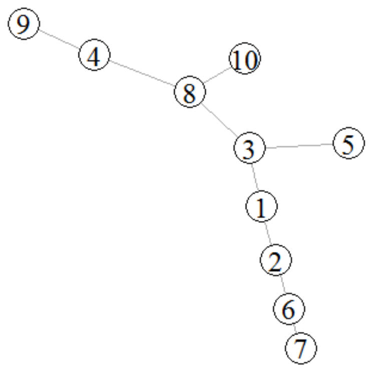


Figure 4.1: *On right, a minimal spanning tree arranged as a graph object. On left, the corresponding weighted adjacency matrix with weights equivalent to the Euclidean distances between nodes in the minimal spanning tree.*

With a partial distance matrix constructed, the problem becomes one of completion. That is, given the partial distance matrix $\mathbf{\Delta}^\star$, with known values $\delta^\star_{ij}$, can a Euclidean distance matrix $\mathbf{D}$ be found such that $d_{ij} = \delta^\star_{ij}$ for all known values in $\mathbf{\Delta}^\star$. The problem of completing a partial distance matrix is known as the *Euclidean distance matrix completion problem* (edmcp). Once such a matrix $\mathbf{D}$ can be found, creating a point configuration is then trivial. As was mentioned in Chapter 1, a point configuration can be created from a distance matrix $\mathbf{D}$ via eigendecomposition. Section 4.2 introduces the general Euclidean distance matrix completion problem, and presents three solutions from the literature. Section 4.3 discusses the problem of Euclidean distance matrix completion when only the minimal spanning tree is known, and introduces two new completion algorithms to solve this problem. One is a modification of an existing algorithm, and the second is a novel solution. Finally, Section 4.4 briefly introduces the problem of *generating* configurations with a set structure.

4.2 The Euclidean Distance Matrix Completion Problem

This section will introduce the underlying mathematics and relevant existing methods for the Euclidean distance matrix completion problem. Section 4.2.1 introduces the mathematics of the Euclidean distance matrix and its relationship to the Gram matrix, and also introduces four matrix sets of interest as well as four linear functions to move between them. Section 4.2.2 introduces the general Euclidean distance matrix completion problem and Sections 4.2.3 - 4.2.5 introduce three solutions - the semidefinite programming completion algorithm [2], the non-convex position formulation algorithm [34], and the dissimilarity parameterization formulation algorithm [107].

4.2.1 Background Mathematics

Suppose there are n points in arbitrary dimension p, denoted $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n] \in \mathbb{R}^{n\times p}$, centred such that $\sum_{i=1}^n \mathbf{x}_i = \mathbf{0}$. The $n \times n$ *squared* distance matrix is given by $\mathbf{D} = [d_{ij}]$, where

$$d_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||^2$$

and $||\mathbf{x}|| = \sqrt{x_1^2 + ... + x_p^2}$. Let $\mathbf{G} = [g_{ij}] = \mathbf{X}\mathbf{X}^\mathsf{T}$ be the Gram matrix. The squared distances d_{ij} can be written in terms of entries in the Gram matrix as

$$\begin{aligned} d_{ij} &= ||\mathbf{x}_i - \mathbf{x}_j||^2 \\ &= \mathbf{x}_i^\mathsf{T}\mathbf{x}_i + \mathbf{x}_j^\mathsf{T}\mathbf{x}_j - 2\mathbf{x}_i^\mathsf{T}\mathbf{x}_j \\ &= g_{ii} + g_{jj} - 2g_{ij} \end{aligned}$$

Thus, given a Gram matrix $\mathbf{G}$, the squared distance matrix $\mathbf{D}$ can be written as

$$\mathbf{D} = \mathcal{K}(\mathbf{G}) = \mathbf{1}\mathbf{g}^\mathsf{T} + \mathbf{g}\mathbf{1}^\mathsf{T} - 2\mathbf{G} \tag{4.1}$$

where $\mathbf{g} = diag(\mathbf{G}) = [g_{11}, ..., g_{nn}]^\mathsf{T}$. Rearranging Equation (4.1), the Gram matrix $\mathbf{G}$ can be written in terms of the squared distance matrix $\mathbf{D}$ as

$$\mathbf{G} = \frac{1}{2}(\mathbf{1}\mathbf{g}^T + \mathbf{g}\mathbf{1}^\mathsf{T} - \mathbf{D}) \tag{4.2}$$

To write $\mathbf{G}$ fully in terms of $\mathbf{D}$, the vector $\mathbf{g}$ must be written as a function of $\mathbf{D}$. Recall that the $\mathbf{x}_i$ are centred, so the average over the column index of $\mathbf{D}$ is

$$
\begin{aligned}
\bar{d}_{i\cdot} &= \frac{1}{n}\sum_{j=1}^{n} d_{ij} \\
&= \frac{1}{n}\sum_{j=1}^{n}(\mathbf{x}_i^{\mathsf{T}}\mathbf{x}_i + \mathbf{x}_j^{\mathsf{T}}\mathbf{x}_j - 2\mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j) \\
&= \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_i + \frac{1}{n}\sum_{j=1}^{n}\mathbf{x}_j^{\mathsf{T}}\mathbf{x}_j - 2\mathbf{x}_i^{\mathsf{T}}\sum_{j=1}^{n}\mathbf{x}_j \\
&= \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_i + \frac{1}{n}\sum_{j=1}^{n}\mathbf{x}_j^{\mathsf{T}}\mathbf{x}_j
\end{aligned}
$$

Similarly, the overall average can be determined

$$
\begin{aligned}
\bar{d}_{\cdot\cdot} &= \frac{1}{n}\sum_{i=1}^{n}\bar{d}_{i\cdot} \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^{\mathsf{T}}\mathbf{x}_i + \frac{1}{n}\sum_{j=1}^{n}\mathbf{x}_j^{\mathsf{T}}\mathbf{x}_j \\
&= \frac{2}{n}\sum_{i=1}^{n}\mathbf{x}_i^{\mathsf{T}}\mathbf{x}_i
\end{aligned}
$$

Using these, g_{ii} can be written as a function of the d_{ij}

$$
\begin{aligned}
g_{ii} &= ||\mathbf{x}_i||^2 \\
&= \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_i \\
&= \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_i + \frac{1}{n}\sum_{j=1}^{n}\mathbf{x}_j^{\mathsf{T}}\mathbf{x}_j - \frac{1}{n}\sum_{j=1}^{n}\mathbf{x}_j^{\mathsf{T}}\mathbf{x}_j \\
&= \bar{d}_{i\cdot} - \frac{1}{2}\bar{d}_{\cdot\cdot}
\end{aligned}
$$

which is generalized to the vector $\mathbf{g}$ as

$$
\begin{aligned}
\mathbf{g}\mathbf{1}^\mathsf{T} &= \frac{1}{n}\mathbf{D}\mathbf{1}\mathbf{1}^\mathsf{T} - \frac{1}{2}\mathbf{1}\bar{d}_{..}\mathbf{1}^\mathsf{T} \\
&= \frac{1}{n}\mathbf{D}\mathbf{1}\mathbf{1}^\mathsf{T} - \frac{1}{2}\mathbf{1}(\frac{1}{n}\mathbf{1}^\mathsf{T}\mathbf{D}\mathbf{1}\frac{1}{n})\mathbf{1}^\mathsf{T} \\
&= \mathbf{D}\mathbf{H} - \frac{1}{2}\mathbf{H}\mathbf{D}\mathbf{H}
\end{aligned}
$$

where $\mathbf{H} = \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$. With $\mathbf{g}$ now defined as a function of the d_{ij}, the Gram matrix $\mathbf{G}$ can be written as a function of the squared distance matrix $\mathbf{D}$

$$
\begin{aligned}
\mathbf{G} &= \frac{1}{2}(\mathbf{1}\mathbf{g}^\mathsf{T} - \mathbf{D} + \mathbf{g}\mathbf{1}^\mathsf{T}) \\
&= \frac{1}{2}[(\mathbf{D}\mathbf{H} - \frac{1}{2}\mathbf{H}\mathbf{D}\mathbf{H})^\mathsf{T} - \mathbf{D} + (\mathbf{D}\mathbf{H} - \frac{1}{2}\mathbf{H}\mathbf{D}\mathbf{H})] \\
&= \frac{1}{2}(2\mathbf{D}\mathbf{H} - \mathbf{D} - \mathbf{H}\mathbf{D}\mathbf{H}) \\
&= -\frac{1}{2}(\mathbf{D} - \mathbf{D}\mathbf{H} - \mathbf{H}\mathbf{D} + \mathbf{H}\mathbf{D}\mathbf{H}) \\
&= -\frac{1}{2}(\mathbf{I} - \mathbf{H})\mathbf{D}(\mathbf{I} - \mathbf{H})
\end{aligned}
$$

Letting $\mathbf{P} = \mathbf{I} - \mathbf{H}$, define the function $\mathcal{T}(\mathbf{D})$ as the function that converts a squared distance matrix to a Gram matrix

$$
\mathcal{T}(\mathbf{D}) = -\frac{1}{2}\mathbf{P}\mathbf{D}\mathbf{P} \tag{4.3}
$$

Using Equations (4.1) and (4.3), given a squared distance matrix $\mathbf{D}$, a point configuration can be determined by finding the associated Gram matrix $\mathbf{G}$, and applying a simple eigendecomposition,

$$
\begin{aligned}
\mathbf{G} &= \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\mathsf{T} \\
&= (\mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}})(\mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}})^\mathsf{T} \\
&= \mathbf{X}\mathbf{X}^\mathsf{T}
\end{aligned}
$$

where $\mathbf{U}$ is the matrix containing the eigenvectors of $\mathbf{G}$, and $\mathbf{\Lambda}$ is a diagonal matrix containing the corresponding eigenvalues.

The functions $\mathcal{T}$ and $\mathcal{K}$ play an important role in not only moving between a Gram matrix and squared Euclidean distance matrix, but more generally between two distinct spaces of matrices. Let $\mathcal{S}_n$ denote the set of $n \times n$ real symmetric matrices, and define a subset of this space to be all those matrices that are both symmetric and positive semidefinite

$$\mathcal{S}_n^+ = \{\mathbf{S} \in \mathcal{S}_n : \mathbf{S} \succeq 0\}$$

Similarly, define the subsets of symmetric and centred, and symmetric and hollow matrices as

$$\begin{aligned} \mathcal{G}_n &= \{\mathbf{G} \in \mathcal{S}_n : \mathbf{G1} = \mathbf{0}\} \\ \mathcal{D}_n &= \{\mathbf{D} \in \mathcal{S}_n : diag(\mathbf{D}) = \mathbf{0}\} \end{aligned}$$

respectively. Finally, define

$$\begin{aligned} \mathcal{G}_n^+ &= \{\mathbf{G} \in \mathcal{G}_n : \mathbf{G} \succeq 0\} \\ \mathcal{D}_n^- &= \{\mathbf{D} \in \mathcal{D}_n : \mathbf{z}^\mathsf{T}\mathbf{Dz} \leq 0 \text{ when } \mathbf{z}^\mathsf{T}\mathbf{1} = 0\} \end{aligned}$$

to be the set of symmetric, centred, and positive semidefinite matrices, and the set set of symmetric and hollow matrices that are negative semidefinite on the space spanned by $\mathbf{z}^\mathsf{T}\mathbf{1} = 0$ respectively. Each of these spaces are positive cones of some intrinsic interest, and provide a means of moving back and forth through different representation of the same problem. For instance, [34] and [62] show that $\mathcal{G}_n^+ = \mathcal{T}(\mathcal{D}_n^-)$ and $\mathcal{D}_n^- = \mathcal{K}(\mathcal{G}_n^+)$, implying these functions have an inverse relationship on these spaces (proof in Appendix A.1). This means that any problem written in terms of a matrix $\mathbf{D} \in \mathcal{D}_n^-$ could be re-written in terms of a matrix $\mathbf{G} \in \mathcal{G}_n^+$.

With the underlying mathematics now defined, the general Euclidean distance matrix problem can be stated.

4.2.2 The General Completion Problem

For a partial squared Euclidean distance matrix $\mathbf{\Delta}^\star$, define a matrix $\mathbf{A} = [a_{ij}]$ such that $a_{ij} = 1$ if $\delta_{ij}^\star$ is known, and 0 otherwise. The known entries of $\mathbf{\Delta}^\star$ can then be determined through a simple element-wise multiplication of $\mathbf{A}$ and $\mathbf{\Delta}^\star$, known as a Hadamard product, written as $\mathbf{A} \circ \mathbf{\Delta}^\star$. The general Euclidean distance matrix completion problem can then be stated in terms of the following minimization problem

$$\mathbf{\Delta}_0 = \underset{\mathbf{\Delta} \in \mathcal{D}_n^-}{\arg\min} \ ||\mathbf{A} \circ (\mathbf{\Delta}^\star - \mathbf{\Delta})||_F^2 \tag{4.4}$$

where the norm $||\cdot||_F$ is the Frobenius norm, computed as $||\mathbf{M}||_F = \sqrt{trace(\mathbf{M}^\mathsf{T}\mathbf{M})}$. The resulting squared Euclidean distance matrix $\mathbf{\Delta}_0$ is a *completion* of the partial distance matrix $\mathbf{\Delta}^\star$. In words, the problem seeks to find the squared Euclidean distance matrix $\mathbf{\Delta} = [\delta_{ij}]$ which minimizes the difference between δ_{ij} and $\delta_{ij}^\star$ for all i, j such that $a_{ij} = 1$ (i.e. for all known entries in $\mathbf{\Delta}^\star$).

Since $\mathbf{\Delta} \in \mathcal{D}_n^-$, the general completion problem can also be written in terms of the Gram matrix

$$\begin{aligned} \mathbf{G}_0 = & \underset{\mathbf{S} \in \mathcal{G}_n^+}{\arg\min} \ ||\mathbf{A} \circ (\mathbf{\Delta}^\star - \mathcal{K}(\mathbf{S}))||_F^2 \\ = & \underset{\mathbf{S} \in \mathcal{G}_n^+}{\arg\min} \ ||\mathbf{A} \circ \mathcal{K}(\mathbf{\Delta}^\star - \mathbf{S})||_F^2 \end{aligned} \tag{4.5}$$

with the second line following due to the linearity of the $\mathcal{K}$ operator.

The problem of Euclidean distance matrix completion has been approached using many different techniques by re-stating the general completion problem in Equation (4.4) or (4.5). In subsections 4.2.3-4.2.5, three such solutions are discussed.

4.2.3 The Semidefinite Programming Algorithm

One method to solve Equation (4.4) is to first convert it to a semidefinite programming problem [2]. A semidefinite program is a convex optimization problem with a linear objective function to be maximized over the cone of positive semidefinite matrices. Note that the cone of positive semidefinite matrices is the set of all positive semidefinite matrices such that if $\mathbf{A}$ is in the set, $\alpha\mathbf{A}$, $\alpha > 0$ is also in the set. Further, define the convex cone of positive semidefinite matrices to have the additional constraint that if $\mathbf{A}, \mathbf{B}$ are in the set, then so to is $\alpha\mathbf{A} + \beta\mathbf{B}$, $\alpha, \beta > 0$.

To transform Equation (4.4), first recall the linear operator $\mathcal{T}(\mathbf{D}) = -\frac{1}{2}\mathbf{PDP}$. Letting $\mathbf{P} = \mathbf{VV}^\mathsf{T}$, where $\mathbf{V}$ is an $n \times (n-1)$ matrix which satisfies

$$\begin{aligned} \mathbf{V}^\mathsf{T}\mathbf{V} = & \ \mathbf{I}_{n-1} \\ \mathbf{V}^\mathsf{T}\mathbf{1} = & \ \mathbf{0} \end{aligned}$$

define the linear operators

$$\begin{aligned} \mathcal{T}_v(\mathbf{D}) = \mathbf{V}^\mathsf{T}\mathcal{T}(\mathbf{D})\mathbf{V} \ &= \ -\frac{1}{2}\mathbf{V}^\mathsf{T}\mathbf{D}\mathbf{V} \\ \mathcal{K}_v(\mathbf{S}) = \mathcal{K}(\mathbf{V}\mathbf{S}\mathbf{V}^\mathsf{T}) & \end{aligned}$$

Then, for any matrix $\mathbf{D} \in \mathcal{D}_n^-$, $\mathcal{T}_v(\mathbf{D}) \in \mathcal{S}_{n-1}^+$ where $\mathcal{S}_{n-1}^+$ is the space of $(n-1) \times (n-1)$ symmetric positive semidefinite matrices. Further, for any matrix $\mathbf{S} \in \mathcal{S}_{n-1}^+$, $\mathcal{K}_v(\mathbf{S}) \in \mathcal{D}_n^-$. That is, $\mathcal{T}_v$ and $\mathcal{K}_v$ are inverse operators on these spaces [2].

With the ability to move between $\mathcal{D}_n^-$ and $\mathcal{S}_{n-1}^+$, [2] write the general Euclidean distance matrix completion problem of Equation (4.4) as

$$S_0 \ = \ \underset{\mathbf{S} \,\in\, \mathcal{S}_{n-1}^+}{\arg\min} \ ||\mathbf{A} \ \circ \ (\mathbf{\Delta}^\star \ - \ \mathcal{K}_v(\mathbf{S}))||_F^2 \tag{4.6}$$

The search space $\mathcal{S}_{n-1}^+$ can be broadened so that the problem can be formulated as a semidefinite programming algorithm. The operators $\mathcal{T}_v$ and $\mathcal{K}_v$ relate the set of Euclidean distance matrices $\mathcal{D}_n^-$ to the less restrictive set $\mathcal{P}_{n-1}^+$ of real $(n-1) \times (n-1)$ positive semidefinite matrices [2], which allows the problem to be cast as one of semidefinite programming

$$\begin{array}{ll} \text{Minimize} & f(\mathbf{S}) = ||\mathbf{A} \ \circ \ (\mathbf{\Delta}^\star \ - \ \mathcal{K}_v(\mathbf{S}))||_F^2 \\ \text{subject to} & \\ & \begin{aligned} a(\mathbf{S}) \ &= \mathbf{c} \\ \mathbf{S} \ &\succeq 0 \end{aligned} \end{array} \tag{4.7}$$

Here, the equality constraint $a(\mathbf{S}) \ = \ \mathbf{c}$ enforces the constraints on the known squared distances - $\mathcal{K}(\mathbf{S})_{ij} \ = \ \delta_{ij}^\star$ for all known distances in $\mathbf{\Delta}^\star$.

While this problem is convex, an attribute of semidefinite programming that is not shared by the other completion algorithms under consideration, it has two flaws. First, the embedding dimension of the solution cannot be fixed, potentially resulting in the Gram matrix $\mathbf{G}$ (and corresponding point configuration $\mathbf{X}$) residing in high dimension. In addition, the algorithm has complexity $\mathcal{O}(n^2)$, since it works directly with the distance matrix. Due to this, the time needed to find a solution increases drastically as the size of the distance matrix increases.

4.2.4 The Non-convex Position Formulation

Unlike the semidefinite programming algorithm, the non-convex position formulation [34] allows the Euclidean distance matrix problem to be solved with a fixed embedding dimension by reformulating Equation (4.4) in terms of a Gram matrix $\mathbf{G}$ with constrained rank

$$\begin{aligned} &\underset{\mathbf{G} \in \mathcal{G}_n^+}{\text{Minimize}} \quad ||\mathbf{A} \circ (\mathbf{\Delta}^\star - \mathcal{K}(\mathbf{G}))||_F^2 \\ &\text{subject to} \\ &\qquad rank(\mathbf{G}) = p \end{aligned} \tag{4.8}$$

For a fixed embedding dimension p, Equation (4.8) can be re-formulated in terms of the point configuration matrix $\mathbf{X}$ as

$$\underset{\mathbf{X} \in \mathcal{R}^{n \times p}}{\text{Minimize}} \, f(\mathbf{X}) = ||\mathbf{A} \circ (\mathbf{\Delta}^\star - \mathcal{K}(\mathbf{X}\mathbf{X}^\mathsf{T}))||_F^2$$

which can be minimized using standard numerical optimization techniques [34]. A major advantage of formulating the problem in this way is the size of the search space, which is $\mathcal{O}(np)$ as opposed to the $\mathcal{O}(n^2)$ of the semidefinite programming algorithm.

The ability to specify the rank of the Gram matrix comes at the cost of the convexity of the objective function [34], meaning that the algorithm may converge on a local minimum during optimization. The non-convex position formulation algorithm uses three methods to avoid local minima during optimization.

First, a random start algorithm is implemented as follows. For a partial distance matrix $\mathbf{\Delta}^\star$, define the matrix $\mathbf{B}$ such that $b_{ij} = \delta_{ij}^\star$ for all known entries in $\mathbf{\Delta}^\star$. Since the matrix $\mathbf{\Delta}^\star$ must form a connected graph, the remaining unknown entries in $\mathbf{B}$ can be determined as the shortest path distances along the connected graph. For convenience, call these distances f_{ij}. In practice, these f_{ij} tend to be much larger than the actual distances [34]. To correct for this, each b_{ij} is scaled by a random realization, s_{ij}, from a $\mathcal{N}(1.5, 0.3)$ distribution, truncated between 1 and the number of segments along the shortest path in computing f_{ij} [34]

$$\widehat{b}_{ij} = \frac{f_{ij}}{s_{ij}}$$

Since the $\widehat{b}_{ij}$ are computed with a random component, several instances of the matrix $\widehat{\mathbf{B}}$ are created. Using a spectral decomposition on the resulting matrix, a p-dimensional point configuration $\widehat{\mathbf{X}}$ is created (much like in isomap), and the one resulting in the smallest value of Equation (4.8) is used as the initial guess in the optimization.

The second method used to avoid local minima is referred to as stretching, numerical scaling of the point configuration $\mathbf{X}$ generated from the spectral decomposition of $\widehat{\mathbf{B}}$. For some $\alpha > 0$, stretching would result in the matrix $\alpha\widehat{\mathbf{X}}$. While it is noted that there is no theoretical basis for stretching in this context, through empirical observation it was observed that an increased chance of landing on the global minimum occurred when stretching was employed [34].

Finally, and perhaps most importantly, is the implementation of dimension relaxation. By artificially inflating the dimension of the point configuration during the spectral decomposition of $\widehat{\mathbf{B}}$, the non-convex position formulation algorithm may be able to reach the global minimum with a higher probability [34]. For instance, in the higher dimensional space, there may be a way to pass between local minima that would have been converged on in lower dimensional space on the way to the global minimum.

Once a solution is found in the higher dimensional space, it must be brought down to the desired dimension p. This is done in one of two ways [34]

- Principal Component Analysis
- Nonlinear Dimension Reduction

Under the second technique, reduction of the dimension of the optimal configuration from d to p is done through solving the optimization problem

$$\begin{array}{rcl} \underset{\mathbf{W}}{\text{Minimize}} & ||\mathbf{W}_2||_F^2 & \\ \text{subject to} & f(\mathbf{W}) & = & 0 \end{array}$$

where $\mathbf{W} = [\mathbf{W}_1,\ \mathbf{W}_2] \in \mathbb{R}^{n\times d}$ is a point configuration matrix in d-dimensional space, and $\mathbf{W}_1$ is a point configuration matrix in dimension p, where $p < d$. Requiring $f(\mathbf{W}) = 0$ ensures the resulting p dimensional point configuration matrix $\mathbf{W}_1$ is a global minimum of the original minimization.

4.2.5 The Dissimilarity Parameterization Formulation

The final Euclidean distance matrix completion algorithm is the dissimilarity parameterization formulation [107]. Recall that the matrix $\mathbf{A}$ is an indicator matrix with $a_{ij} = 1$ if $\delta^\star_{ij}$ is known, and 0 otherwise. Let $\mathcal{C}_n$ be the set of all $n \times n$ dissimilarity matrices, and further define the set $\mathcal{C}_n(\mathbf{A}\circ\mathbf{\Delta}^\star)$ as the set of all completions of the partial dissimilarity matrix $\mathbf{A}\circ\mathbf{\Delta}^\star$. Mathematically, $\mathcal{C}_n(\mathbf{A}\circ\mathbf{\Delta}^\star)$ is defined as

$$\mathcal{C}_n(\mathbf{A}\circ\mathbf{\Delta}^\star) = \{\mathbf{\Delta} \in \mathcal{C}_n : \mathbf{A}\circ\mathbf{\Delta} = \mathbf{A}\circ\mathbf{\Delta}^\star\} \tag{4.9}$$

Recall that $\mathcal{D}_n^-$ is the set of all $n \times n$ Euclidean distance matrices, and define $\mathcal{D}_n^-(p)$ as the set of $n \times n$ Euclidean distance matrices from p dimensional point configurations.

Then, a solution to the p dimensional embedding problem exists only if $\mathbf{\Delta} \in \mathcal{C}_n(\mathbf{A}\circ\mathbf{\Delta}^\star)$ and $\mathbf{\Delta} \in \mathcal{D}_n^-(p)$. That is, a solution to the problem exists only if $\mathcal{C}_n(\mathbf{A}\circ\mathbf{\Delta}^\star) \cap \mathcal{D}_n^-(p) \neq \varnothing$. This intersection is non-empty if and only if the following minimization has a global minimum of zero [107]

$$\begin{aligned} \underset{\mathbf{G},\boldsymbol{\Delta}}{\text{Minimize}} \quad & ||\mathbf{G} - \mathcal{T}(\boldsymbol{\Delta})||_F^2 \\ \text{subject to:} \quad & \mathbf{G} \in \mathcal{S}_n^+ \quad \text{and} \quad \text{rank}(\mathbf{G}) \le p \\ & \boldsymbol{\Delta} \in \mathcal{C}_n(\mathbf{A} \circ \boldsymbol{\Delta}^\star) \end{aligned} \tag{4.10}$$

Note that while this formulation is similar to both the semidefinite programming formulation and the nonconvex position formulation, there is a large distinction in that the masking matrix $\mathbf{A}$ and the known distances $\mathbf{A} \circ \boldsymbol{\Delta}^\star$ now appear as part of the constraints given by the set of allowed dissimilarity matrices $\mathcal{C}_n(\mathbf{A} \circ \boldsymbol{\Delta}^\star)$. This allows restrictions to be imposed on the unknown values.

Letting

$$\mathcal{C}_n(\boldsymbol{\Delta}^L, \boldsymbol{\Delta}^U) = \{\boldsymbol{\Delta} \in \mathcal{C}_n : \delta_{ij}^L \le \delta_{ij} \le \delta_{ij}^U\}$$

where $\boldsymbol{\Delta}^L = [\delta_{ij}^L]$ and $\boldsymbol{\Delta}^U = [\delta_{ij}^U]$ are both in $\mathcal{C}_n$, formulation (4.10) is a special case of

$$\begin{aligned} \underset{\mathbf{G},\boldsymbol{\Delta}}{\text{Minimize}} \quad & ||\mathbf{G} - \mathcal{T}(\boldsymbol{\Delta})||_F^2 \\ \text{subject to:} \quad & \mathbf{G} \in \mathcal{S}_n^+ \quad \text{and} \quad \text{rank}(\mathbf{G}) \le p \\ & \boldsymbol{\Delta} \in \mathcal{C}_n(\boldsymbol{\Delta}^L, \boldsymbol{\Delta}^U) \end{aligned} \tag{4.11}$$

where $\boldsymbol{\Delta}^L$ and $\boldsymbol{\Delta}^U$ are fixed matrices chosen such that $\mathbf{A} \circ \boldsymbol{\Delta}^L = \mathbf{A} \circ \boldsymbol{\Delta}^U$. That is, when $\delta_{ij}^\star$ is known, the upper and lower bounds are set equal to $\delta_{ij}^\star$. Whenever $a_{ij} = 0$, then the default value of the bounds are $\delta_{ij}^L = 0$ and $\delta_{ij}^U = +\infty$. An advantage of this formulation is that tighter bounds can be imposed on the unknown $\delta_{ij}^\star$, restricting the space of possible solutions. For example using the structure of the graph given by $\mathbf{A}$, upper bounds δ_{ij}^U can be determined for all unknown δ_{ij} by simply invoking the triangle inequality.

Equation (4.11) is a standard optimization problem, which can be solved efficiently using the L-BFG-S optimization algorithm [129][107]. Like the non-convex position formulation algorithm, the dissimilarity parameterization algorithm is not convex, meaning that it may converge on a local minimum, instead of the global minimum. A random start algorithm is again implemented in the dissimilarity parameterization formulation.

4.3 Completions from the Minimal Spanning Tree

For any Euclidean distance matrix completion problem, a minimum requirement is that the graph underlying the partial distance matrix be connected. Otherwise, the problem separates into two disconnected groups with no information to locate the relationship between them. For an $n \times n$ distance matrix, the spanning tree creates a connected graph object with only $n - 1$ edges (or known distances), the minimum number possible. Interest lies in completions of a Euclidean distance matrix when the graph given by $\mathbf{A}$ determines a spanning tree on its n nodes and the known distances given by $\mathbf{A} \circ \mathbf{\Delta}^{\star}$ are also the minimal spanning tree distances of the completion.

All methods in Section 4.2 can be applied when only minimal spanning tree is known, though there may be computational challenges given that $1 - \frac{2}{n}$ of the distances are missing. None, however, are *specifically* designed to ensure that the minimal spanning tree remains unchanged in the completion. Since the minimal spanning tree is known to contain important clustering information [33][47][100], the goal here is to find *minimal spanning tree preserving* completions.

More formally, let $\mathcal{A}_n \subset \mathcal{D}_n$ denote the set of symmetric $n \times n$ adjacency matrices, $\mathcal{A}_n^{\star} \subset \mathcal{A}_n$ the subset corresponding to spanning trees, and $\mathcal{A}_n^{\star}(\mathbf{A}) \subset \mathcal{A}_n^{\star}$ the set of spanning tree adjacency matrices restricted to a subgraph of $\mathbf{A}$ (i.e. $\mathbf{A} - \mathbf{A}^{\star} \in \mathcal{A}_n$ whenever $\mathbf{A}^{\star} \in \mathcal{A}_n^{\star}$). Further let $amst(\mathbf{A}, \mathbf{\Delta}) = \{\mathbf{A}_1, \ldots, \mathbf{A}_k\}$ denote the set of adjacency matrices $\mathbf{A}_i \in \mathcal{A}_n^{\star}(\mathbf{A})$ which produce a minimal spanning tree for the graph given by $\mathbf{A}$ and $\mathbf{\Delta} \in \mathcal{C}_n$. This set will typically be a singleton, but could be larger whenever there are tied values within a dissimilarity matrix $\mathbf{\Delta}$. The product $\mathbf{A}_i \circ \mathbf{\Delta}$ will determine a minimal spanning tree for any $\mathbf{A}_i \in amst(\mathbf{A}, \mathbf{\Delta})$.

Now let

$$\mathcal{M}_n(\mathbf{A}, \mathbf{A}^{\star}, \mathbf{\Delta}^{\star}) = \{\mathbf{\Delta} \in \mathcal{C}_n : amst(\mathbf{A}, \mathbf{\Delta}) = amst(\mathbf{A}^{\star}, \mathbf{\Delta}^{\star})\}$$

denote all those dissimilarity matrices $\mathbf{\Delta}$ which with a given adjacency matrix $\mathbf{A}$ will have the same minimal spanning tree adjacency matrix set as that for the target $\mathbf{\Delta}^{\star}$ and $\mathbf{A}^{\star}$. When non-empty, the set $\mathcal{M}_n(\mathbf{A}, \mathbf{A}^{\star}, \mathbf{\Delta}^{\star}) \subset \mathcal{C}_n$ is a convex cone.

Theorem 1. *The set* $\mathcal{M}_n := \mathcal{M}_n(\mathbf{A}, \mathbf{A}^{\star}, \mathbf{\Delta}^{\star}) = \{\mathbf{\Delta} \in \mathcal{C}_n : amst(\mathbf{A}, \mathbf{\Delta}) = amst(\mathbf{A}^{\star}, \mathbf{\Delta}^{\star})\}$ *is a convex cone.*

Proof. Two things need to be shown:

1. if $\mathbf{\Delta} \in \mathcal{M}_n$ then $\alpha\mathbf{\Delta} \in \mathcal{M}_n$ for any real $\alpha > 0$, and
2. if $\mathbf{\Delta} \in \mathcal{M}_n$ and $\mathbf{\Lambda} \in \mathcal{M}_n$ then $\mathbf{\Gamma} = \alpha\mathbf{\Delta} + \beta\mathbf{\Lambda} \in \mathcal{M}_n$ for any reals $\alpha, \beta > 0$.

First, note that $\mathcal{C}_n$ is clearly a convex cone from its definition, so the requirement that members of $\mathcal{M}_n$ also be members of $\mathcal{C}_n$ is trivially satisfied for both items above, so we need only check that the minimal spanning tree requirements are met.

Item 1 is also trivially true. If $amst(\mathbf{A}, \mathbf{\Delta}) = amst(\mathbf{A}^\star, \mathbf{\Delta}^\star)$ then a common rescaling of all elements in $\mathbf{\Delta}$ will make no change to the adjacency of any minimal spanning tree.

Item 2 is proved by showing that $amst(\mathbf{A}, \mathbf{\Delta}) \subseteq amst(\mathbf{A}, \mathbf{\Gamma})$, then that $amst(\mathbf{A}, \mathbf{\Gamma}) \subseteq amst(\mathbf{A}, \mathbf{\Delta})$, implying $amst(\mathbf{A}, \mathbf{\Gamma}) = amst(\mathbf{A}, \mathbf{\Delta})$.

To show $amst(\mathbf{A}, \mathbf{\Delta}) \subseteq amst(\mathbf{A}, \mathbf{\Gamma})$, let $\mathbf{M} \in amst(\mathbf{A}, \mathbf{\Delta})$ and $\mathbf{B} \in \mathcal{A}_n^\star$ be any spanning tree of both $\mathbf{A}$ and $\mathbf{A}^\star$. We write (twice) the sum of dissimilarities of $\mathbf{M} \circ \mathbf{\Gamma}$ as

$$\begin{aligned}
\mathbf{1}^\mathsf{T}(\mathbf{M} \circ \mathbf{\Gamma})\mathbf{1} &= \mathbf{1}^\mathsf{T}(\mathbf{M} \circ (\alpha\mathbf{\Delta}))\mathbf{1} + \mathbf{1}^\mathsf{T}(\mathbf{M} \circ (\beta\mathbf{\Lambda}))\mathbf{1} \\
&\leq \mathbf{1}^\mathsf{T}(\mathbf{B} \circ (\alpha\mathbf{\Delta}))\mathbf{1} + \mathbf{1}^\mathsf{T}(\mathbf{B} \circ (\beta\mathbf{\Lambda}))\mathbf{1} \\
&= \mathbf{1}^\mathsf{T}(\mathbf{B} \circ \mathbf{\Gamma})\mathbf{1}
\end{aligned}$$

which implies that $\mathbf{M} \in amst(\mathbf{A}, \mathbf{\Gamma})$ and so $amst(\mathbf{A}, \mathbf{\Delta}) \subseteq amst(\mathbf{A}, \mathbf{\Gamma})$.

To show $amst(\mathbf{A}, \mathbf{\Gamma}) \subseteq amst(\mathbf{A}, \mathbf{\Delta})$, let $\mathbf{M} \in amst(\mathbf{A}, \mathbf{\Delta})$ and $\mathbf{B} \in amst(\mathbf{A}, \mathbf{\Gamma})$. The result is proved by contradiction. Suppose $\mathbf{B} \notin amst(\mathbf{A}, \mathbf{\Delta})$. Then we have both that $\mathbf{1}^\mathsf{T}(\mathbf{B} \circ (\alpha\mathbf{\Delta}))\mathbf{1} > \mathbf{1}^\mathsf{T}(\mathbf{M} \circ (\alpha\mathbf{\Delta}))\mathbf{1}$ and that $\mathbf{1}^\mathsf{T}(\mathbf{B} \circ (\beta\mathbf{\Lambda}))\mathbf{1} > \mathbf{1}^\mathsf{T}(\mathbf{M} \circ (\beta\mathbf{\Lambda}))\mathbf{1}$.

Together these imply that

$$\begin{aligned}
\mathbf{1}^\mathsf{T}(\mathbf{B} \circ \mathbf{\Gamma})\mathbf{1} &= \mathbf{1}^\mathsf{T}(\mathbf{B} \circ (\alpha\mathbf{\Delta}))\mathbf{1} + \mathbf{1}^\mathsf{T}(\mathbf{B} \circ (\beta\mathbf{\Lambda}))\mathbf{1} \\
&> \mathbf{1}^\mathsf{T}(\mathbf{M} \circ (\alpha\mathbf{\Delta}))\mathbf{1} + \mathbf{1}^\mathsf{T}(\mathbf{M} \circ (\beta\mathbf{\Lambda}))\mathbf{1} \\
&= \mathbf{1}^\mathsf{T}(\mathbf{M} \circ \mathbf{\Gamma})\mathbf{1}
\end{aligned}$$

which means that $\mathbf{M}$ has a shorter spanning tree and hence $\mathbf{B} \notin amst(\mathbf{A}, \mathbf{\Gamma})$, a contradiction. Therefore $\mathbf{B} \in amst(\mathbf{A}, \mathbf{\Delta})$ and hence $amst(\mathbf{A}, \mathbf{\Gamma}) \subseteq amst(\mathbf{A}, \mathbf{\Delta})$. $\square$

The set of mst-preserving completions of $\mathbf{A}^\star \circ \mathbf{\Delta}^\star$ is now simply the intersection of $\mathcal{C}_n(\mathbf{A}^\star \circ \mathbf{\Delta}^\star)$ and $\mathcal{M}_n(\mathbf{A}, \mathbf{A}^\star, \mathbf{\Delta}^\star)$ when $\mathbf{A} = \mathbf{K} = \mathbf{1}\mathbf{1}^\mathsf{T} - \mathrm{diag}(\mathbf{1})$ is the adjacency matrix for a complete graph on n nodes.

Analogous to Equation (4.9), define this set to be

$$\mathcal{M}_n(\mathbf{A} \circ \mathbf{\Delta}^\star) = \{\mathbf{\Delta} \in \mathcal{C}_n(\mathbf{A} \circ \mathbf{\Delta}^\star) : amst(\mathbf{K}, \mathbf{\Delta}) = amst(\mathbf{A}, \mathbf{\Delta}^\star)\}$$

and note now that mst-preserving completions with embedding dimension p exist if and only if $\mathcal{M}_n(\mathbf{A} \circ \mathbf{D}) \cap \mathcal{D}_n^-(p) \neq \varnothing$. Note also that the minimal spanning tree fixes only $n-1$ dissimilarities/distances and leaves $(n-1)(n-2)/2$ to be determined. Moreover, the fixed distances are the smallest that produce a spanning tree. The set $\mathcal{M}_n(\mathbf{A} \circ \boldsymbol{\Delta}^\star)$ is very large.

As in the dissimilarity parameterization formulation [107], an mst-preserving completion problem in p dimensions can now be expressed as

$$\begin{aligned} \underset{\mathbf{G},\boldsymbol{\Delta}}{\text{Minimize}} \quad & ||\mathbf{G} - \mathcal{T}(\boldsymbol{\Delta})||_F^2 \\ \text{subject to:} \quad & \mathbf{G} \in \mathcal{S}_n^+ \ \text{ and } \ \operatorname{rank}(\mathbf{G}) \leq p \\ & \boldsymbol{\Delta} \in \mathcal{M}_n(\mathbf{A} \circ \boldsymbol{\Delta}^\star) \end{aligned} \tag{4.12}$$

achieving a zero global minimum. This differs from the Equation (4.10) only in restricting $\boldsymbol{\Delta}$ to $\mathcal{M}_n(\mathbf{A} \circ \boldsymbol{\Delta}^\star)$, a subset of $\mathcal{C}_n(\mathbf{A} \circ \boldsymbol{\Delta}^\star)$, which suggests that the methods used in the dissimilarity parameterization formulation could be adapted to find an mst-preserving completion by solving a minimization problem. That is, Equation (4.12) could be reduced to Equation (4.11) exactly as before, provided that $\boldsymbol{\Delta}^L$ and $\boldsymbol{\Delta}^U$ are chosen so as to ensure the other constraints hold.

4.3.1 Judicious Choice of Bounds

In the dissimilarity parameterization formulation, solving Equation (4.11) only specified $\delta_{ij}^L = 0$; no greater lower bound is used. If lower bounds can be determined so that the minimal spanning tree is maintained then solving Equation (4.11) will also solve Equation (4.12). This observation suggests that an adaptation of the dissimilarity parameterization formulation algorithm [107] using the correct non-zero lower bounds will produce an mst-preserving completion.

Algorithm 2 is used in conjunction with the dissimilarity parameterization formulation algorithm to construct lower bounds for all distances that will preserve the minimal spanning tree in the completion. The insight in constructing the lower bound is drawn from single-linkage clustering. Every edge in a spanning tree separates the vertices into two different groups, depending on which points remain connected to either one vertex or the other of that edge. Because the tree is a minimal spanning tree, if we select the largest edge, then the distance between any vertex of one group and any vertex of the other group must be at least as large as that of the largest edge. This gives a lower bound for these distances that will preserve that edge in the minimal spanning tree. The same reasoning is applied recursively to each separate group, thus producing a lower bound on all edges.

With these lower bounds computed in advance, we need only use these in the dissimilarity parameterization formulation algorithm to find a solution to Equation (4.12). This algorithm will be denoted as the dissimilarity parameterization formulation with lower bound algorithm so as

Algorithm 2 MST lower bounds algorithm

```
Structures
    tree: $T = (V, E)$ is a spanning tree with vertex set $V = \{v\}$ and edge set $E = \{e\}$;
    edges: $e$ will be a set of two indices $\{i, j\} = nodes(e)$ and have a weight $wt(e) = \delta_{ij} \geq 0$;
    $\mathbf{\Delta}^L = [\delta^L_{ij}]$ is the matrix of dissimilarity lower bounds to be determined;

procedure splitTree($T, splitEdge$)                                   ▷ $T$ is a spanning tree
    $restEdges \leftarrow edges(T) - \{splitEdge\}$          ▷ Remove $splitEdge$ from the edge set of $T$
    $(v_1, v_2) \leftarrow nodes(splitEdge)$
    $V_1 \leftarrow \{v_1\}$; $E_1 \leftarrow \{e \in restEdges : v_1 \in nodes(e)\}$          ▷ $E_1$ could be empty
    $V_2 \leftarrow \{v_2\}$; $E_2 \leftarrow \{e \in restEdges : v_2 \in nodes(e)\}$          ▷ $E_2$ could be empty
    while $restEdges \neq \varnothing$ do
        $e \leftarrow restEdges[1]$
        $restEdges \leftarrow restEdges - \{e\}$
        if $nodes(e) \cap nodes(E_1)$ then
            $E_1 \leftarrow E_1 \cup \{e\}$
        else
            $E_2 \leftarrow E_2 \cup \{e\}$
        end if
    end while
    $V_1 \leftarrow V_1 \cup nodes(E_1)$; $V_2 \leftarrow V_2 \cup nodes(E_2)$
    return $\{T_1 := (V_1, E_1),\ \ T_2 := (V_2, E_2)\}$                  ▷ Return the two trees
end procedure

procedure MSTLowerBounds($T, \mathbf{\Delta}^L$)             ▷ Recursively determines the lower bounds
    if $edges(T) \neq \varnothing$ then                             ▷ Ensure there are edges left in $T$
        $maxEdge \leftarrow \arg\max_{e \in edges(T)} wt(e)$                  ▷ Split on the biggest edge
        $Trees \leftarrow$ splitTree($T, maxEdge$)
        for $v_1 \in nodes(Trees[T_1])$ do
            for $v_2 \in nodes(Trees[T_2])$ do
                $\mathbf{\Delta}^L[v_1, v_2] \leftarrow \mathbf{\Delta}^L[v_2, v_1] \leftarrow wt(maxEdge)$          ▷ Set the lower bound
            end for
        end for
        for $Tree \in Trees$ do
            $\mathbf{\Delta}^L \leftarrow$ MSTLowerBounds($Tree, \mathbf{\Delta}^L$)          ▷ Recursively get lower bounds
        end for
    end if
    return $\mathbf{\Delta}^L$                                  ▷ Return the matrix of lower bounds
end procedure
```

to note its dependence on the dissimilarity parameterization formulation algorithms, but with a specified minimal spanning tree preserving lower bound.

4.3.2 A Constructive Solution

For any $\mathbf{A}$ and $\mathbf{\Delta}^{\star}$, the set $\mathbf{\Delta} \in \mathcal{M}_n(\mathbf{A} \circ \mathbf{\Delta}^{\star})$ is large; for a connected graph it is at its largest when $\mathbf{A}$ specifies a minimal spanning tree. In this case, it should be possible to find a completion in $\mathbf{\Delta} \in \mathcal{M}_n(\mathbf{A} \circ \mathbf{\Delta}^{\star})$. Consider simply constructing such a completion by locating points $\mathbf{x}_i \in \mathbb{R}^p$ (p being the embedding dimension) one at a time, while checking that the (partial) minimal spanning tree is preserved as each point is added.

More formally, define $\mathbf{X}_k = [\mathbf{x}_1, \ldots, \mathbf{x}_k]^{\mathsf{T}}$ to be a $k \times p$ matrix whose rows are point locations $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^p$. The locations are chosen so that the minimal spanning tree from the Euclidean distances of these k locations in $\mathbb{R}^p$ is identical to that of k connected nodes from the minimal spanning tree $\mathbf{A} \circ \mathbf{\Delta}^{\star}$. The matrices $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ are constricted by growing (and preserving) the minimal spanning tree one node at a time. The distance matrix from $\mathbf{X}_n$ provides an mst-preserving completion.

The construction begins by choosing the node of maximal degree from the minimal spanning tree of $\mathbf{A} \circ \mathbf{\Delta}^{\star}$ and locating it at $\mathbf{x}_1 = \mathbf{0}$. The second node to locate will be that of maximal degree amongst those connected to the first. If the dissimilarity between these two nodes is, say, δ_{12}, then the location of $\mathbf{x}_2$ is chosen at random from a uniform distribution on the surface of a sphere S^{p-1} in $\mathbb{R}^p$ of radius $\sqrt{\delta_{12}}$ centred at $\mathbf{x}_1$ (assuming squared distances for the completion matrix). The two points $\mathbf{x}_1$ and $\mathbf{x}_2$ are trivially a subtree of the minimal spanning tree. The remaining nodes with connections to $\mathbf{x}_1$ are then added in similar fashion.

As each location is proposed, its (squared) distance to all other placed points is calculated and the resulting distance matrix checked to see whether the minimal spanning tree (so far) is preserved. If it is, the point is accepted; if not, points are generated until one is acceptable. When new nodes are added, they are chosen amongst those without locations that share an edge in the minimal spanning tree with nodes already located. At each step, available nodes of highest degrees are added before nodes of low degree in $\mathbf{A} \circ \mathbf{\Delta}^{\star}$; since these are harder to place, they appear earlier. Algorithm 3 describes the method in detail.

4.4 Generating More Constrained Configurations

The original motivation of exploring Euclidean distance matrix completions from the minimal spanning tree was to be able to generate a configuration with a given level of the skewed scagnostic. Incidentally, the minimal spanning tree also controls several other scagnostic measures, as it determines the degree of each node (controlling the *stringy* scagnostic), the size of the largest edges

Algorithm 3 Guided Random Search Algorithm

```
Structures
    $T_V = (V, E)$ is a tree spanning the vertex set $V = \{1, \ldots, n\}$ with edge set $E$;
    $\mathbf{A} = [a_{ij}]$ and $\mathbf{A}^* = [a^*_{ij}]$ are $n \times n$ symmetric adjacency matrices;
    $\boldsymbol{\Delta} = [\delta_{ij}]$, is an $n \times n$ symmetric squared dissimilarity matrix;
    $\mathbf{X}$ is an $n \times p$ point configuration matrix to be constructed.

    procedure mstConfigure($\mathbf{A}, \boldsymbol{\Delta}$, maxIn = 100, maxOut = 100)
        nTries $\leftarrow 0$;   Converged? $\leftarrow$ FALSE; $T_V \leftarrow (V, E) \leftarrow tree(\mathbf{A})$;
        repeat
            nTries $\leftarrow$ nTries $+ 1$
            $\mathbf{X} \leftarrow \mathbf{0}$; $\boldsymbol{A} \leftarrow \mathbf{0}$;                                          ▷ Initialization
            $i \leftarrow \arg\max_{j \in V} degree(node(j))$                   ▷ Start at any maximal degree node $i$ in $T_V$
            $P \leftarrow \{i\}$                                                      ▷ Initial vertex set
            $T_P \leftarrow (P, \varnothing)$                                                 ▷ Root the tree
            Grow? $\leftarrow$ TRUE                                               ▷ Keep growing flag

            while Grow? do
                $(g, B) \leftarrow$ getBuds$(T_P, T_V)$                    ▷ grow $B \subset V$, $B \cap P = \varnothing$ from $g \in T_P$
                for $b \in B$ do
                    $(T_P, \boldsymbol{\Delta}, \mathbf{A}, \mathbf{X}$, Converged?, Grow?$) \leftarrow$ growTree$(g, b, T_P, T_V, \boldsymbol{\Delta}, \mathbf{A}, \mathbf{X}$, maxIn$)$
                end for
            end while
        until nTries $>$ maxOut or Converged?  == TRUE
        return $(\mathbf{X}$, Converged?$)$                                ▷ Return the point configuration
    end procedure

    procedure growTree($i, j, T_P, T_V, \boldsymbol{\Delta}, \mathbf{A}, \mathbf{X}$, maxTries $= 100$)
        Placed? $\leftarrow$ Converged? $\leftarrow$ FALSE;  nTries $\leftarrow 0$; $\mathbf{A}^* \leftarrow \mathbf{A}$;   $\boldsymbol{\Delta}^* \leftarrow \boldsymbol{\Delta}$;          ▷ Initialization
        repeat
            nTries $\leftarrow$ nTries $+ 1$
            $\mathbf{z} \sim$ Uniform$(S^{p-1})$                            ▷ Generate a random direction vector
            $\mathbf{x}_j \leftarrow \mathbf{x}_i + \mathbf{z} \times \sqrt{\delta_{ij}}$                                    ▷ Propose the point
            for $k \in P$ do                                             ▷ Try values for all placed nodes
                $\delta^*_{jk} \leftarrow \delta^*_{kj} \leftarrow ||\mathbf{x}_k - \mathbf{x}_j||^2$
                $a^*_{jk} \leftarrow a^*_{kj} \leftarrow 1$
            end for
            if mst$(\mathbf{A}^* \circ \boldsymbol{\Delta}^*) \subset T_V$ then Placed? $\leftarrow$ TRUE                 ▷ Preserves the MST?
            end if
        until Placed? or nTries $>$ maxTries
        if Placed? then                                                         ▷ Accept the point
            $\mathbf{X}[j,] \leftarrow \mathbf{x}_j^{\mathsf{T}}$;   $\mathbf{A} \leftarrow \mathbf{A}^*$;   $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}^*$;
            $nodes(T_P) \leftarrow P \cup \{j\}$; $edges(T_P) \leftarrow edges(T_P) \cup \{(i,j),(j,i)\}$
            if $T_P = T_V$ then Converged? $\leftarrow$ TRUE
            end if
        end if
        return $(T_P, \boldsymbol{\Delta}, \mathbf{A}, \mathbf{X}$, Converged?, Placed?$)$
    end procedure

    procedure getBuds$(T_P, T_V)$
        $B \leftarrow V - P$
        $E_* = \{(i,j) : i \in P, j \in B, (i,j) \in edges(T_V)\}$                ▷ edges in $T_V$ connecting $P$ and $B$
        $g \leftarrow \arg\max_{i \in P} \#\{e : e \in E_*$  and  $i \in e\}$            ▷ $g \in P$ having most connections to $B$
        $B \leftarrow \{b : (g,b) \in E_*\}$                                   ▷ reduce $B$ to nodes connected to $g$
        return $(g, B)$
    end procedure
```

(controlling both the *outlying* and *sparse* scagnostics), and the RUNT dendrogram (controlling the *clumpy* scagnostic).

In addition to the ability to control the edge length distribution of the minimal spanning tree, the angles between the proposed points can also be controlled. Since each new point is placed by first proposing an angle from the unit circle, the *striated* scagnostic can be controlled by proposing angles from a restricted subset of the unit circle or from a desired angle distribution.

The question remaining then, is what do these configurations look like, and are they of any interest? This question will be thoroughly discussed in Chapter 10.

Chapter 5

edmcr - An R Package for Euclidean Distance Matrix Completion Problems

As both the newly proposed dissimilarity parameterization formulation and guided random search algorithms are designed to create completions from the minimal spanning tree, examining the relationship between the sparsity of the partial distance matrix and the accuracy of the completed Euclidean distance matrix for all of the algorithms considered is certainly of interest. Interest lies in the relationship between sparsity and the accuracy of the resulting point configuration for the various methods. These experiments (and others) will be conducted in Chapter 6.

To execute these types of experiments, each of the algorithms discussed in Chapter 4 need to be programmed in a common language so they can be compared directly. For this purpose, `edmcr` is introduced, an R package in which each of the algorithms of interest are implemented. The purpose of this chapter is to introduce this package and validate each of the implementations via example.

In addition to the algorithms needed for the experiments in Chapter 6, two additional algorithms have also been implemented in `edmcr`. First, to perform sensor network localization, the algorithm of [66] is implemented. To execute molecular conformation, the semidefinite programming-based protein structure determination (SPROS) algorithm of [3] is also implemented. The details of these two algorithms will also be discussed in this chapter (although they will not be used further).

A large portion of the implementation of the SPROS algorithm was the underlying optimization solver. The solver implemented was the semidefinite quadratic linear optimization solver of [105], and was implemented in the R package `sdpt3r`. While implementation of the solver does not represent new, novel research (and hence has been included as an appendix), it does allow for a large number of new problems to be solved in R that were previously unavailable. The details of `sdpt3r` can be found in Appendix C.

5.1 Euclidean Distance Matrix Completion in R

The `edmcr` package is a novel addition to the `R` library in the form of Euclidean distance matrix completion. There are currently five methods available to complete a partial distance matrix in the `edmcr` package.

1. Semidefinite programming algorithm [2]
2. Non-convex position formulation [34]
3. Dissimilarity parameterization formulation [107]
4. Dissimilarity parameterization formulation with minimal spanning tree preserving lower bounds [89]
5. Guided random search [89]

In addition, the sensor network localization algorithm of [66] and the SPROS algorithm [3] for molecular conformation are also available in `edmcr`.

5.1.1 edmc

To streamline the implementation of the Euclidean distance matrix completion algorithms, the main function of the `edmcr` package, `edmc` allows any of the Euclidean distance matrix completion algorithm above to be called. The `edmc` function takes the following input variables

`D`	An $n \times n$ Euclidean distance matrix to be completed, with unknown entries set to NA.
`method`	The completion algorithm to be used. One of `sdp`, `npf`, `dpf`, `snl`, `grs`.
`...`	Additional input variables specific to the `method` used.

Of practical importance in the implementation of `edmc` is to note that only two inputs are specified by name in the definition of the function - a partial Euclidean distance matrix `D`, and the desired completion algorithm `method`. All other input variables are specific to the algorithm specified in `method`, and are therefore not specified in the `edmc` input list. The user must specify the required input variables (outlined for each `method` below) by naming them directly during the call to `edmc`.

The output of `edmc` is also specific to the method used, and will be discussed individually with each method.

method = "sdp"

The semidefinite programming algorithm [2], denoted *sdp*, and coded in the internal function `sdp`, requires the following input variables to be specified in `edmc`

`D`	An $n \times n$ Euclidean distance matrix to be completed, with unknown entries set to NA.
`A`	A weight matrix, with $a_{ij} = 0$ implying d_{ij} is unknown. Generally, if d_{ij} is known, $a_{ij} = 1$ although any non-negative weight is allowed.
`toler`	The convergence tolerance for the algorithm. The default is set to $1e-8$.

Successful completion of the routine results in the following list of output variables

`D`	The completed $n \times n$ Euclidean distance matrix.
`optval`	The minimum value of the objective function in Equation (4.7) achieved during minimization.

The `sdp` algorithm is appropriate for smaller completion problems, as it is computationally quite expensive ($\mathcal{O}(n^2)$).

method = "npf"

The non-convex position formulation algorithm [34], denoted *npf*, and coded in the internal function `npf`, requires the following input variables to be specified in `edmc`

`D`	An $n \times n$ Euclidean distance matrix to be completed, with unknown entries set to NA.
`A`	A weight matrix, with $a_{ij} = 0$ implying d_{ij} is unknown. Generally, if d_{ij} is known, $a_{ij} = 1$, although any non-negative weight is allowed.
`d`	The dimension of the resulting completion.
`dmax`	The maximum dimension to consider during dimension relaxation. The default is set to be $n-1$, where n is the number of rows in `D`.
`decreaseDim`	During dimension reduction, the number of dimensions to decrease by at each step. The default is set to 1.
`stretch`	The stretching factor applied to the distance matrix (see Section 4.2.4). The default is set to 1 (i.e. no stretching is applied).
`dimMethod`	One of `Linear` (principal components) or `NLP` (nonlinear dimension reduction) corresponding to the desired method of dimension reduction (see Section 4.2.4). The default is set to `Linear`.
`toler`	The convergence tolerance for the algorithm. The default is set to $1e-8$.

In most cases, the default values of the input variables `dmax`, `decreaseDim`, `stretch`, `dimMethod`, and `toler` are sufficient to find a solution. As such, these input values need not be specified during the call to `edmc` - only `D`, `A`, and `d` are required. An advanced user may be interested in changing these defaults in the event that the algorithm fails to converge on a global solution. Section 4.2.4 outlines the technical background required to understand how these optional input variables can be used.

In the case of utilizing the `npf` internal function, `edmc` produces the following list of outputs

`D`	The completed $n \times n$ Euclidean distance matrix.
`optval`	The minimum value of the objective function in Equation (4.8) achieved during minimization.

The `npf` algorithm is appropriate for any completion problem, as it is substantially faster than `sdp`. This increase in speed unfortunately comes at the cost of convexity, so care must be taken to ensure that algorithm arrives on a global solution, and not a local solution. A global solution is achieved if the value returned by `optval` is sufficiently close to 0.

method = "dpf"

The dissimilarity parameterization formulation algorithm [107], denoted *dpf*, and coded in the internal function `dpf`, requires the following input variables to be specified in `edmc`

`D`	An $n \times n$ Euclidean distance matrix to be completed, with unknown entries set to NA.
`d`	The dimension for the resulting completion.
`lower`	An $n \times n$ matrix containing the lower bounds for the unknown entries in `D`. If NULL (default), `lower` is set to be a matrix of zeros.
`upper`	An $n \times n$ matrix containing the upper bounds of the unknown entries in `D`. If NULL, `upper`[i,j] is set to be the shortest path between node i and node j.
`retainMST`	A logical input indicating if the current minimal spanning tree structure in `D` should be retained. If `TRUE`, an mst-preserving lower bound is calculated.

For `dpf`, both `D` and `d` must be specified by the user. The remaining inputs are optional and have set defaults. Their use will be described via example in Section 5.2. The `dpf` subroutine produces the following list of outputs

`D`	An $n \times n$ Euclidean distance matrix with embedding dimension d.
`optval`	The minimum value of the objective function in Equation (4.11) achieved during minimization.

By setting the `retainMST` input variable to true, the dissimilarity parameterization formulation with minimal spanning tree lower bounds algorithm [89] is executed. In this case, if `lower` has been specified by the user, it is overwritten using the minimal spanning tree preserving lower bounds computed using Algorithm 2.

method = “grs”

When the `method` variable is set to `"grs"`, `edmc` executes the guided random search algorithm [89] by calling the internal function `grs`. The `grs` algorithm is used to solve Euclidean distance matrix completion problems where only the minimal spanning tree is known. When `grs` is the specified method, two input variables are required

`D`	An $n \times n$ partial Euclidean distance matrix to be completed containing only the distances in the desired minimal spanning tree.
`d`	The dimension of the resulting completion.

Completion of the matrix $\mathbf{D}$ results in two output variables

`P`	An $n \times d$ matrix containing the positions of the n nodes in d dimensions.
`D`	An $n \times n$ Euclidean distance matrix with embedding dimension `d`.

In Section 5.4, the ability of this algorithm to create point configurations in a lower embedding dimension than the original partial Euclidean distance matrix is explored.

method = “snl”

Up to this point, only algorithms explicitly designed for Euclidean distance matrix completion have been explored. While the sensor network localization algorithm of [66] is capable of performing vanilla Euclidean distance matrix completion, it is designed for the purpose of sensor network localization. Given a set of anchors with known positions and a number of sensors for which some distances to the anchors are known, the problem of sensor network localization is to find the positions of as many of the unknown sensors as possible.

The sensor network localization algorithm of [66], denoted *snl*, and coded in the internal function `snl`, requires the following input variables to be specified in `edmc`

`D`	The partial distance matrix specifying the known distances between nodes. If `anchors` is specified (and is a $p \times d$ matrix), the p final columns and p final rows of `D` specify the distances between the anchors.
`d`	The dimension for the resulting completion.
`anchors`	A $p \times d$ matrix specifying the d dimensional locations of the p anchors. If the anchorless problem (i.e. a general edmcp) is to be solved, `anchors` = NULL.

The `snl` algorithm requires that both `D` and `d` be specified as input variables, and an additional argument `anchors`, which specifies `d`-dimensional positions of the anchors is optional. When `anchors` is specified, the algorithm seeks to locate (in `d`-dimensional space) as many of the sensors as possible. Unlike the other algorithms considered thus far, the main purpose of the `snl` algorithm is to solve the sensor network localization problem, however, it can also solve the Euclidean distance matrix completion problem by specifying the `anchors` input as `NULL`. Executing the `snl` algorithm results in the following output

`X`	the d-dimensional positions of the localized sensors.

The $m \times d$ matrix `X` contains the positions of the localized sensors. It is not necessarily the case that all sensors can be localized, in which case they are not included in `X`, and therefore we have $m \leq n$, where n is the total number of sensors.

sprosr

Determining the structure of a molecule requires a specialized algorithm that can handle the additional angle constraints often found in molecular conformation problems. In *edmcr*, the protein problem can be solved using the `sprosr` function, an `R` implementation of the *semidefinite programming-based protein structure determination algorithm* [3]. `sprosr` has three required and one optional input variable

`seq`	A table containing the amino acid sequence of the protein in CYANA .seq format.
`aco`	A table containing the angle constraint information in CYANA .aco format
`upl`	A table containing the distance constraint information in CYANA .upl format.
`hydrogen_omission`	Should side-chain hydrogen atoms be omitted? TRUE/FALSE. Default is FALSE.

Each of `seq`, `aco`, and `upl` require a table following the form set out by CYANA [49]. For `seq`, the sequence file (following the format of the .seq CYANA file) is a table of two columns - column one is the abbreviated name of the amino acid, and column two is the corresponding residue number. The `aco` input table contains angular constraints on the phi and psi angles for the amino acid residues, and follows the format of the .aco CYANA file. It contains five columns - the residue number of the amino acid corresponding to the `seq` input variable, the name of the amino acid residue, the name of the angle being constrained (either PHI or PSI), the lower limit of the angle, and the upper limit of the angle, in degrees. Finally, the `upl` input table contains the distance constraints (in Ångströms) between amino acid residues. It contains seven columns - the residue number, amino acid name, and atom name of the first (columns 1-3) and second (columns 4-6) residues, and the constrained distance between them. Table 5.1 provides an example of each input type.

MET	1
GLN	2
ILE	3
PHE	4
VAL	5

(a)

2	GLN	PHI	-95.0	-85.0
3	ILE	PHI	-149.0	-139.0
4	PHE	PHI	-129.0	-119.0
6	LYS	PHI	-117.0	-107.0
7	THR	PHI	-110.0	-100.0

(b)

1	MET	HA	2	GLN	H	2.93
1	MET	HB2	2	GLN	H	3.86
1	MET	HB2	63	LYS	HA	5.50
1	MET	HB3	2	GLN	H	3.86
1	MET	HB3	63	LYS	HA	5.50

(c)

Table 5.1: *(a) The first 5 rows of a sequence (seq) file. (b) The first five rows of an angle constraint (aco) file. (c) The first 5 rows of a distance constraint (upl) file.*

`sprosr` provides two output variables upon completion

`X`	The three dimensional point configuration of the completed protein molecule
`report`	A list detailing the total violations in the protein molecule

The `X` output is straight forward, it contains the three dimensional positions of the amino acid residues in the completed protein molecule. The `report` output is a list detailing the total violations in the protein, including the number (and total size) of the violations in the equality constraints (`$eq_err`), the number of violations in both the lower (`$lo_err`) and upper bounds (`$up_err`), the number of chiral violations (`$chiral`), the number of angle violations in both phi (`$phi`) and psi (`$psi`), and the number of dihedral (`$dihed`) and hydrogen bonds (`$hbond`) that violate their upper bounds.

5.2 Unconstrained EDMCP

This section serves as a quick validation of the implementation of the five Euclidean distance matrix algorithms that will be used to perform the experiments of Chapter 6. Consider the following partial Euclidean distance matrix (from [107]), and its known completion in three dimensional space.

$$\begin{bmatrix} 0 & 3 & 4 & 3 & 4 & 3 \\ 3 & 0 & 1 & ? & 5 & ? \\ 4 & 1 & 0 & 5 & ? & 5 \\ 3 & ? & 5 & 0 & 1 & ? \\ 4 & 5 & ? & 1 & 0 & 5 \\ 3 & ? & 5 & ? & 5 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 & 4 & 3 & 4 & 3 \\ 3 & 0 & 1 & \sqrt{18} & 5 & \sqrt{18} \\ 4 & 1 & 0 & 5 & \sqrt{32} & 5 \\ 3 & \sqrt{18} & 5 & 0 & 1 & \sqrt{18} \\ 4 & 5 & \sqrt{32} & 1 & 0 & 5 \\ 3 & \sqrt{18} & 5 & \sqrt{18} & 5 & 0 \end{bmatrix}$$

For comparative purposes, note that $\sqrt{18} \approx 4.243$ and $\sqrt{32} \approx 5.657$. To solve the problem using *edmcr*, first define the Euclidean distance matrix in `R` as follows

```
R> #Define the partial distance matrix
R> D <- matrix(c(0,3,4,3,4,3,
                 3,0,1,NA,5,NA,
                 4,1,0,5,NA,5,
                 3,NA,5,0,1,NA,
                 4,5,NA,1,0,5,
                 3,NA,5,NA,5,0), byrow=TRUE, nrow=6)
```

For solving standard Euclidean distance matrix completion problems such as this, the `sdp`, `npf`, and `dpf` algorithms should be considered. Beginning with the `sdp` algorithm

```
R> #Define the adjacency matrix A
R> A <- matrix(c(1,1,1,1,1,1,
                 1,1,1,0,1,0,
                 1,1,1,1,0,1,
                 1,0,1,1,1,0,
                 1,1,0,1,1,1,
                 1,0,1,0,1,1), byrow=TRUE, nrow=6)

R> edmc(D=D, method = "sdp", A=A, toler=1e-8)

$D
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 0.000000 3.000020 3.999993 3.000020 3.999993 3.000011
[2,] 3.000020 0.000000 1.000051 4.321144 5.000003 4.453067
[3,] 3.999993 1.000051 0.000000 5.000003 5.656642 5.000002
[4,] 3.000020 4.321144 5.000003 0.000000 1.000051 4.453067
[5,] 3.999993 5.000003 5.656642 1.000051 0.000000 5.000002
[6,] 3.000011 4.453067 5.000002 4.453067 5.000002 0.000000

$optval
[1] 5.406998e-10
```

Note that while a solution to the problem was found (since the optimal function value is near zero), this matrix differs slightly from the known completion. The main reason for this difference is the rank of the completed distance matrix. The corresponding Gram matrix has five non-zero eigenvalues, meaning the completed squared distance matrix has rank five, where the original solution had an embedding dimension of three.

Next, consider the solution given by the `npf` algorithm, with default settings used where appropriate

```
R> set.seed(48)

R> #Define the adjacency matrix A
R> A <- matrix(c(1,1,1,1,1,1,
              1,1,1,0,1,0,
              1,1,1,1,0,1,
              1,0,1,1,1,0,
              1,1,0,1,1,1,
              1,0,1,0,1,1), byrow=TRUE, nrow=6)

R> #Default settings for some inputs (not required as input to edmc())
R> dmax = n - 1
R> decreaseDim = 1
R> stretch = 1
R> method = "Linear"
R> toler = 1e-8

R> edmc(D=D, method="npf", A=A, d=3)

$D
     [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,]    0 3.000000 4.000000 3.000000 4.000000 3.000000
[2,]    3 0.000000 1.000001 4.242615 5.000000 4.241516
[3,]    4 1.000001 0.000000 5.000000 5.656888 5.000000
[4,]    3 4.242615 5.000000 0.000000 1.000001 4.241561
[5,]    4 5.000000 5.656888 1.000001 0.000000 5.000000
[6,]    3 4.241516 5.000000 4.241561 5.000000 0.000000

$optval
[1] 3.894878e-11
```

The resulting solution, unlike in the `sdp` subroutine, converges very nearly to the known solution. This is due to the ability to specify the embedding dimension. This, however, comes with its own set of difficulties. Suppose the embedding dimension of the example matrix is unknown, and is instead specified as `d = 2`, resulting in the following completion

```
R> set.seed(48)
R> edmc(D=D, method="npf", A=A, d=2)
```

```
$D
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 0.000000 2.957321 3.711975 2.957320 3.711976 2.104172
[2,] 2.957321 0.000000 1.423176 3.660281 5.036896 4.790540
[3,] 3.711975 1.423176 0.000000 5.036896 6.377897 5.118735
[4,] 2.957320 3.660281 5.036896 0.000000 1.423176 4.790539
[5,] 3.711976 5.036896 6.377897 1.423176 0.000000 5.118735
[6,] 2.104172 4.790540 5.118735 4.790539 5.118735 0.000000

$optval
[1] 72.33753
```

No solution achieving a global minimum of zero is found in two dimensions, so the algorithm is forced to settle on a local solution, reaching an optimal value of 72.34. If it is believed a solution actually exists in two dimensions, it may be worthwhile to attempt stretching, or possibly increasing the number of dimensions considered during relaxation. However, a more likely solution is to change the completion dimension.

Finally, consider the solution provided using the `dpf` algorithm

```
R> set.seed(98)

R> edmc(D=D, method="dpf", d=3)

$D
     [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,]    0 3.000000 4.000000 3.000000 4.000000 3.000000
[2,]    3 0.000000 1.000000 4.240979 5.000000 4.245210
[3,]    4 1.000000 0.000000 5.000000 5.659058 5.000000
[4,]    3 4.240979 5.000000 0.000000 1.000000 4.248187
[5,]    4 5.000000 5.659058 1.000000 0.000000 5.000000
[6,]    3 4.245210 5.000000 4.248187 5.000000 0.000000

$optval
[1] 1.878904e-09
```

Again, with the ability to specify a an embedding dimension, the `dpf` algorithm is able to converge very nearly to the global minimum. However, similar to the `npf` algorithm, consider what happens if the embedding dimension is unknown, and an attempt is made to complete the matrix with an embedding dimension of $d = 2$

```
R> set.seed(98)

R> edmc(D=D, method="dpf", d=2)

$D
     [,1]     [,2] [,3]     [,4] [,5]     [,6]
[1,]    0 3.000000    4 3.000000    4 3.000000
[2,]    3 0.000000    1 3.967597    5 4.349922
[3,]    4 1.000000    0 5.000000    6 5.000000
[4,]    3 3.967597    5 0.000000    1 4.349923
[5,]    4 5.000000    6 1.000000    0 5.000000
[6,]    3 4.349922    5 4.349923    5 0.000000

$optval
[1] 8.456663
```

With an achieved minimum value of 8.46, the algorithm has again failed to converge near the required global minimum of 0, indicating that the proposed solution is not a distance matrix.

5.3 A Completion from the Minimal Spanning Tree

To demonstrate the guided random search algorithm and the dissimilarity parameterization formulation with minimal spanning tree lower bounds algorithm, consider the same matrix as above with only the minimal spanning tree known

$$\begin{bmatrix} 0 & 3 & 4 & 3 & 4 & 3 \\ 3 & 0 & 1 & \sqrt{18} & 5 & \sqrt{18} \\ 4 & 1 & 0 & 5 & \sqrt{32} & 5 \\ 3 & \sqrt{18} & 5 & 0 & 1 & \sqrt{18} \\ 4 & 5 & \sqrt{32} & 1 & 0 & 5 \\ 3 & \sqrt{18} & 5 & \sqrt{18} & 5 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 & ? & 3 & ? & 3 \\ 3 & 0 & 1 & ? & ? & ? \\ ? & 1 & 0 & ? & ? & ? \\ 3 & ? & ? & 0 & 1 & ? \\ ? & ? & ? & 1 & 0 & ? \\ 3 & ? & ? & ? & ? & 0 \end{bmatrix}$$

While this can be completed by any of the Euclidean distance matrix completion algorithms, only the guided random search and dissimilarity parameterization with minimal spanning tree lower bounds algorithms are guaranteed to preserve the minimal spanning tree in the completion. Both of these algorithms are used to complete the matrix below.

```
R> set.seed(690)

R> D <- matrix(c(0,3,NA,3,NA,3,
                 3,0,1,NA,NA,NA,
                 NA,1,0,NA,NA,NA,
                 3,NA,NA,0,1,NA,
                 NA,NA,NA,1,0,NA,
                 3,NA,NA,NA,NA,0), byrow=6, byrow=TRUE)

R> dpflb <- edmc(D = D, d = 3, method = "dpf", retain.MST = TRUE)

$D

         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 0.000000 3.000000 3.062128 3.000000 3.759797 3.000000
[2,] 3.000000 0.000000 1.000000 4.290205 4.932488 5.786480
[3,] 3.062128 1.000000 0.000000 4.607175 5.136999 5.548418
[4,] 3.000000 4.290205 4.607175 0.000000 1.000000 4.024081
[5,] 3.759797 4.932488 5.136999 1.000000 0.000000 4.261201
[6,] 3.000000 5.786480 5.548418 4.024081 4.261201 0.000000

R> guided.random <- edmc(D = D, d = 3, method = "grs")

$D

         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 0.000000 3.000000 3.793104 3.000000 3.200879 3.000000
[2,] 3.000000 0.000000 1.000000 3.444424 3.913602 5.482522
[3,] 3.793104 1.000000 0.000000 4.279306 4.640279 6.007068
[4,] 3.000000 3.444424 4.279306 0.000000 1.000000 4.721678
[5,] 3.200879 3.913602 4.640279 1.000000 0.000000 4.243435
[6,] 3.000000 5.482522 6.007068 4.721678 4.243435 0.000000
```

The minimal spanning tree can be shown to have been retained using the `spantree` function in the *vegan* package [82]. The `spantree` function characterizes the minimal spanning tree by identifying (in order) the child nodes of the minimal spanning tree, and the distance between the child node and its respective parent node. Note that the parent node is simply the index (+1) of the child node.

```
R> #minimal spanning tree for the original matrix
R> spantree(D)$kid
[1] 1 2 1 4 1

R> spantree(D)$dist
[1] 3 1 3 1 3

R> #minimal spanning tree for the completion using dpf w/ lower bounds
R> spantree(dpflb$D)$kid
[1] 1 2 1 4 1

R> spantree(dpflb$D)$dist
[1] 3 1 3 1 3

R> #minimal spanning tree for the completion using the guided random search
R> spantree(guided.random$D)$kid
[1] 1 2 1 4 1

R> spantree(guided.random$D)$dist
[1] 3 1 3 1 3
```

Since each minimal spanning tree is identical, both the guided random search algorithm and the dissimilarity parameterization formulation with minimal spanning tree preserving lower bounds algorithm were successful in both completing the distance matrix and preserving the underlying minimal spanning tree.

5.4 Dimension Reduction using the Guided Random Search

The statue data set is a well known data set in dimension reduction due to the belief that it represents a three dimensional manifold, gaining popularity due to its use in the original publication of isomap [103]. It consists of 698 images in 64x64 greyscale of a bust at various angles and lighting conditions. Viewing each pixel as a point in a given dimension, this data set can be viewed as 698 points in 4096 dimensional space.

Consider using the `grs` algorithm to reduce the dimension of the statue data set from 4096 dimensions to three using only the minimal spanning tree. The statue data is available in *edmcr*. Figure 5.1, created using Loon [115], illustrates the results.

```
R> library(vegan)
R> data(statue)
R>
R> #Create partial distance matrix
R> out <- spantree(statue.dist)
R>
R> statue.mst <- matrix(NA,nrow(statue),nrow(statue))
R> diag(statue.mst) <- 0
R.
R> for(i in 2:nrow(statue)){
R>   statue.mst[i,out$kid[i-1]] <- out$dist[i-1]
R>   statue.mst[out$kid[i-1],i] <- out$dist[i-1]
R> }
R>
R> #Dimension Reduction by grs
R> out.grs <- grs(statue.mst,3)
```

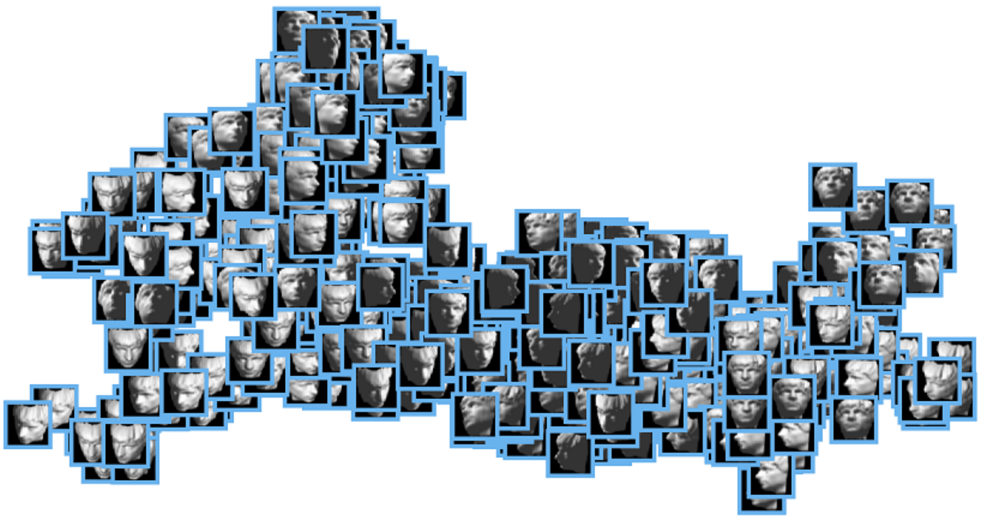


Figure 5.1: *A visualization of the first two dimensions in the three dimensional configuration produced by the Guided Random Search algorithm using the minimal spanning tree of the 4096 dimensional statue manifold data set.*

Despite using only the information in the minimal spanning tree in 4096 dimension space, there are some clear groupings that appear in the first two principal components of the dimension reduced data. First, note that the darker faces tend to be grouped together in the centre of the image. There is also a fade from these dark faces to lighter faces, from the centre of the image in both the right and left directions. There is also clear grouping in the orientation of the bust.

This example shows that, with minimal information, the guided random search algorithm is able to create sensible groupings of a high dimensional configuration in low dimensional space.

5.5 Concluding Remarks

The `edmcr` package implements each of the semidefinite programming algorithm, the non-convex position formulation, the dissimilarity parameterization formulation (plus the minimal spanning tree preserving lower bounds modification), and the guided random search algorithm. This package represents a novel addition to the R library, as previously, Euclidean distance matrix completions problems could not be solved in R.

With the exception of the semidefinite programming algorithm (where the original Matlab implementation will be used), each of these algorithms will be used in Chapter 6 to analyze the effect of sparsity in the partial Euclidean distance matrix on the resulting completion and point configuration. In particular, the limiting case of knowing (and wanting to preserve) the minimal spanning tree will be of interest.

Chapter 6

EDMCP: Experimental Analysis

In Chapter 4 three existing methods of completing Euclidean distance matrices were described. These were the semidefinite programming algorithm, nonconvex position formulation, and dissimilarity parameterization formulation. To these two new methods were added, the dissimilarity parameterization formulation with minimal spanning tree preserving lower bounds, and the guided random search algorithm. In this chapter, the results of these methods will be assessed experimentally in a variety of ways.

Each experiment begins with a known point configuration $\mathbf{X}$ and its Euclidean distance matrix $\mathbf{D}$. Elements of $\mathbf{D}$ will be removed and each method will be expected to find a completion $\widehat{\mathbf{D}}$ and a corresponding point configuration $\widehat{\mathbf{X}}$. The quality of each method is then based on the nearness of $\widehat{\mathbf{X}}$ and $\widehat{\mathbf{D}}$ to the original $\mathbf{X}$ and $\mathbf{D}$, respectively. For each, the time taken to arrive at a solution is measured or, equivalently, the number of solutions produced in the same time.

In the experiments that follow, two different point configurations are considered. The first is the well known Anderson Iris data as collected by [5] and recorded in [37]. The data consists of four measurements (sepal width and length, petal width and length) on each of 150 flowers, 50 from each of three different Iris species (Versicolor, Virginica, and Setosa). The second will be a family of synthetic configurations drawn randomly from uniform distributions on unit hypercubes of varying dimensionality. That is each row location of $\mathbf{X}$ is randomly generated as $\mathbf{x}_i \sim U[0,1]^p$ for $i = 1, \ldots, n$.

6.1 Reconstructing the Iris data

For the three existing methods from Chapter 4, Euclidean distance matrix completions can be generated for any pattern of missing dissimilarities provided the corresponding graph spans all points. To investigate the relative performance of these methods we take $\mathbf{D}$ to be constructed from the distances between flowers in the Iris data. Distances are removed at random and the percentage of distances removed varied. Each completion method produces $\widehat{\mathbf{D}}$ for several randomly selected patterns of missing distances for each percentage missing.

6.1.1 Completions as a Function of Percentage Missing

For each completion of the iris data, two metrics will initially be of interest. First, the time taken (in seconds) for each method to complete the distance matrix is recorded. Second, to measure the accuracy of each completion the relative difference in dissimilarities

$$RDD = \frac{||\mathbf{D} - \widehat{\mathbf{D}}||_F^2}{||\mathbf{D}||_F^2} \tag{6.1}$$

is calculated as well.

The top plot of Figure 6.1 shows the effect percentage missing has on the computational time. For each percentage missing, every method but `SDP` was applied to the same five different incomplete missing at random matrices; the `SDP` method took so long that it was applied to only the first of the five matrices. Not surprisingly computational time increases with the percentage missing. Comparing methods, we see that the `DPF` method is consistently faster than the `NPF` method, which in turn is consistently faster than the `SDP` method. Computational times are given on a logarithmic scale so these differences are substantial. Some variation in the times taken can also be seen, especially for the larger percentages.

The minimal spanning tree case has the greatest percentage missing possible and appears at the far right of each plot. All times to completion here appear to have dropped, with `NPF` and `DPF` switching positions to make it the fastest of the three methods. Since this is the minimal spanning tree case, the two new methods proposed in Chapter 4 can be added. Not surprisingly, `DPFLB` which differs from `DPF` only in having precomputed non-zero lower bounds for every missing distance takes essentially the same time to complete as does `DPF`. More interestingly, the guided random search method `C` is orders of magnitude faster than all other methods.

The lower plot of Figure 6.1 shows the average accuracy with which the various methods reconstruct **D**. On this logarithmic scale lower values indicate greater accuracy so the accuracy of every method decreases as the percentage missing increases – largely because the numerator in Equation (6.1) has more non-zero entries while the denominator remains unchanged. For every percentage up to and including 85% missing, each method was applied to 50 different missing at random matrices, the exception being `SDP` which, because of the time required, was only applied to a single matrix each time. Beyond 85% only 10 different random matrices were used for each of `NPF` and `DPF`. Again the methods can be ordered: `NPF` is consistently most accurate and `SDP` consistently least accurate across all percentages missing.

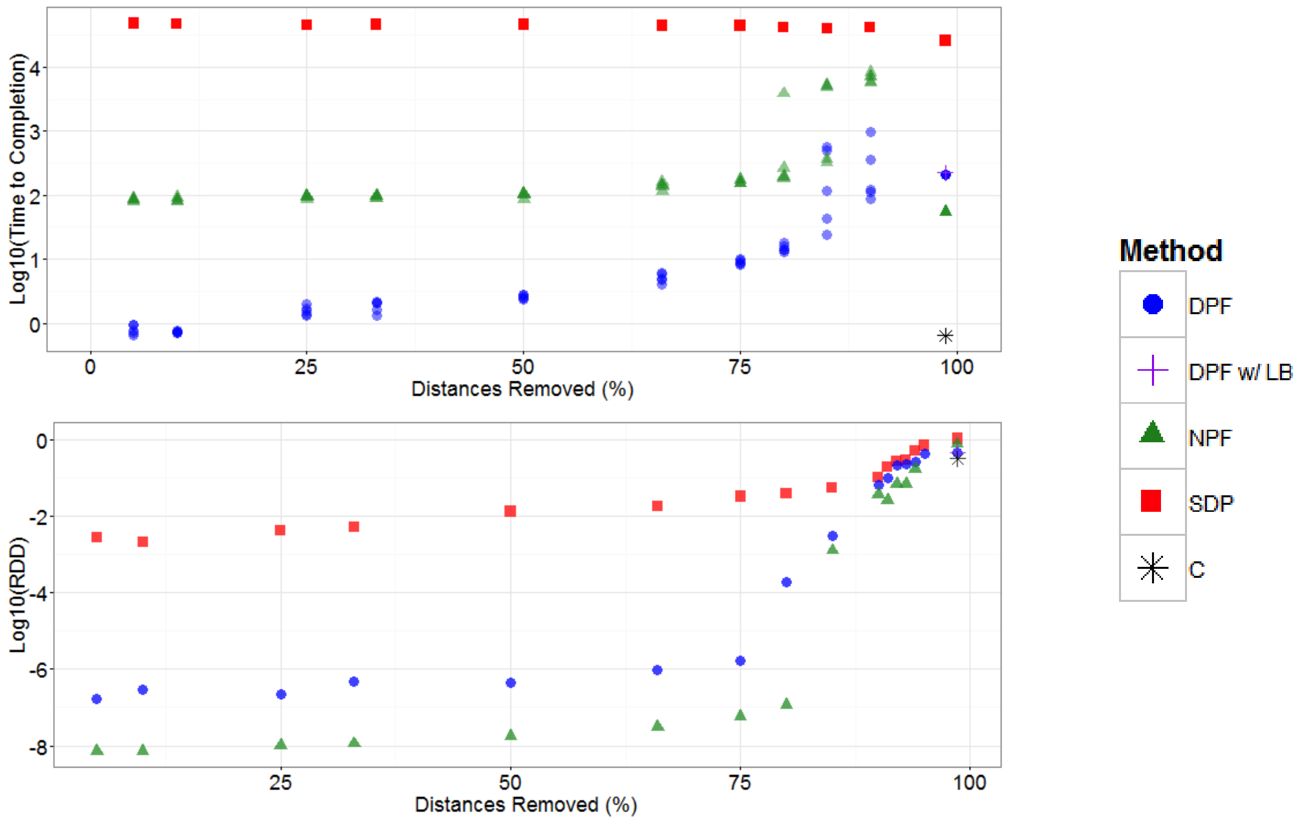


Figure 6.1: *Effect of increasing the percentage of missing distances on each method. Methods are coded by colour and symbol shape. For each percentage, several different matrices with different patterns of randomly selected missing data were used. In the top plot, each point symbol represents the result of one such matrix with one method; alpha blending of colour is used so that places where the values are essentially the same will appear more saturated due to over-plotting and the blending of the colours. In the bottom plot, only the average values are shown.*

For the minimal spanning tree case, there is only one matrix to complete but all methods (with the exception of `SDP`) have a random step (only if necessary for `NPF`) and so could produce many potentially different solutions. Each method was allowed to complete as many completions as possible in the same time taken for `SDP` to construct one completion – `NPF` was able to make 466 completions (all identical since the random step never needed to be invoked), `DPF` made 124 completions, `DPFLB` 114, and the guided random search produced a remarkable 39,935 completions!

Figure 6.2 displays boxplots of the $\log_{10} RDD$ for the five methods. Clearly, the guided random search `C` nearly always outperforms all of the others. From most to least accurate there is the guided random search `C` as most accurate, then `DPF` closely followed by `DPFLB`, then `NPF`, and finally `SDP`.

As was the case with the time to completion, note that again the `DPFLB` and `NPF` switched order in the minimal spanning tree case.

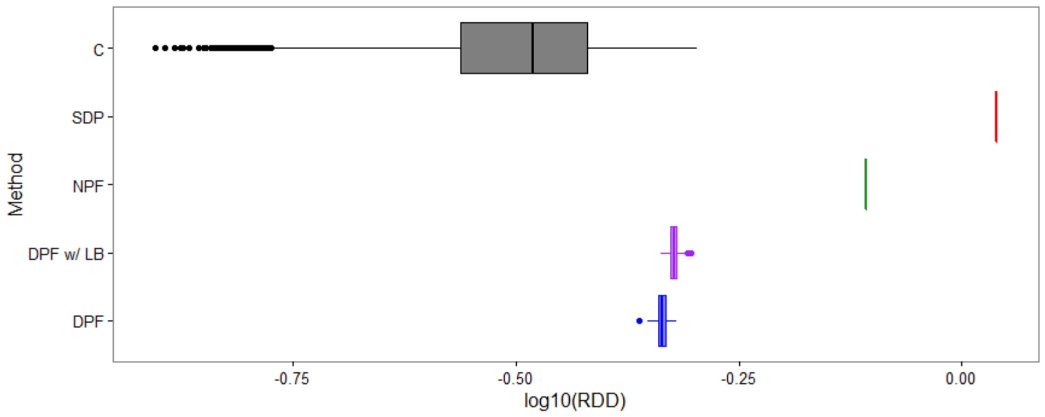


Figure 6.2: *Relative dissimilarity difference when completing the Iris data from its minimal spanning tree distances.* `C` *has 39,935 completions,* `SDP` *one,* `NPF` *466 identical completions,* `DPF` *124 completions, and* `DPFLB` *114 – each set was completed in the time taken for a single* `SDP` *completion ($>$ 7 hours)*

It would seem that `SDP` performed poorest on both measures, whatever the percentage missing. For `DPF` and `NPF` one performed better than the other on each measure and neither dominated the other on both. When we consider beginning only with the minimal spanning tree distances, the guided random search `C` performed best on both measures with the improvement in time being considerable.

6.1.2 Distances as a Function of Percentage Missing

Further insight into the three completion algorithms can be had by comparing the reconstructed distances $\widehat{\mathbf{D}} = [\widehat{d_{ij}}]$ with the actual distances $\mathbf{D} = [d_{ij}]$.

Figure 6.3 plots the pairs $(d_{ij}, \widehat{d_{ij}})$ for all $i < j$ for a few of the missing percentages. Perfect reconstruction would be all points on the $y = x$ line; the box in each plot is the range of the original distances and is identical in size across all plots; scales are identical for the same percentage of missing distances.

For each case, `NPF` outperforms the other two – all distances appear within the box, appear on either side of the $y = x$ line and is nearer to this line in all cases. As the percentage missing increases, the reconstruction of all three methods degrades. Both `DPF` and `SDP` tend to produce ever larger distances in their reconstructions as the percentage increases. `DPF` does produce some distances that are smaller than the original too. In contrast, `SDP` has a tendency to consistently produce distances that are too large for every percentage, and produces much larger distances than does `DPF`. Overall `NPF` provides the best reconstructed distances and `SDP` the worst.

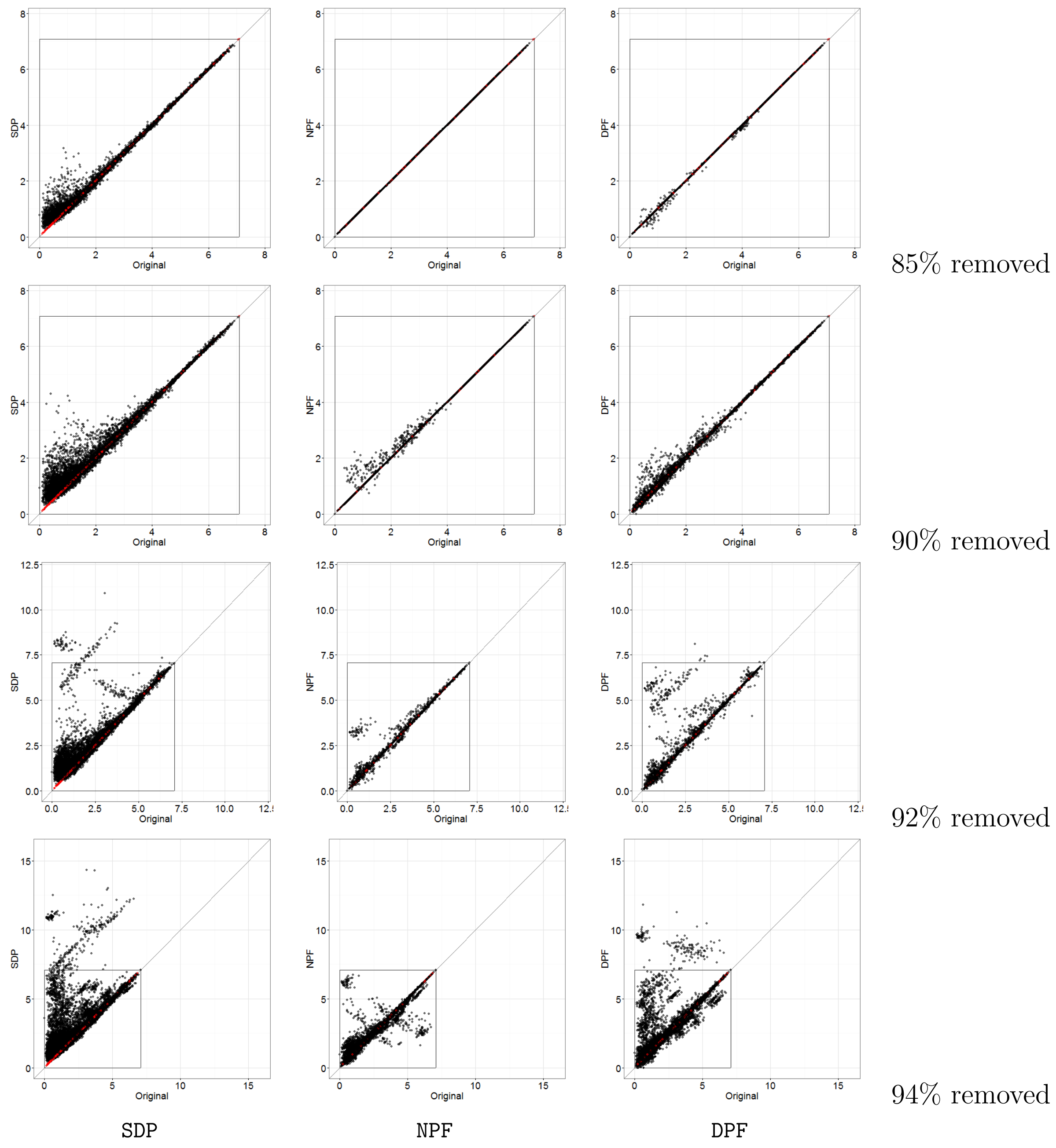
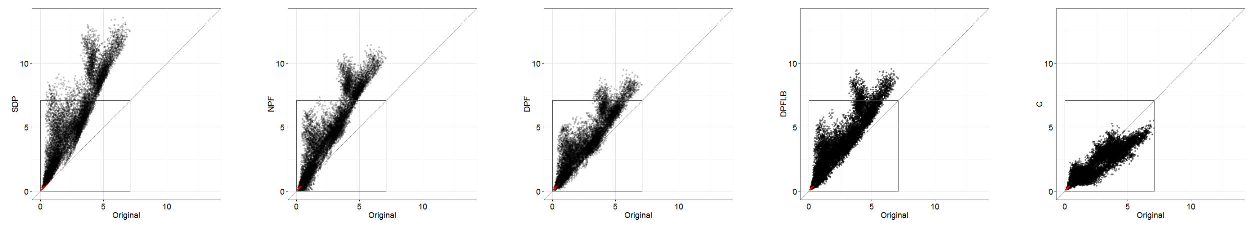


Figure 6.3: *Plots of the pairs of* $(d_{ij}, \widehat{d_{ij}})$ *for all* $i < j$ *for a single reconstruction of the Iris distance matrix; red values are the original minimal spanning tree distances. Perfect reconstruction would be all points on the* $y = x$ *line; the box in each plot is the range of the original distances and is identical in size across all plots; scales are identical for the same percentage of missing distances.*

Turning to minimal spanning tree completions, Figure 6.4 shows the pairs $(d_{ij}, \widehat{d_{ij}})$ for all $i < j$ for a single reconstruction for all five methods. The shapes of the first four (`SDP`, `NPF`, `DPF`, and `DPFLB`) are surprisingly similar, each showing three different branches. All four have almost all distances $\widehat{d_{ij}} > d_{ij}$ and many larger than $\max_{ij} d_{ij}$, with `SDP` producing the largest distances, followed by `NPF`, then `DPFLB`. The last of these has larger distances than those of `DPF` likely because of the lower bounds which ensure that `DPFLB` is mst-preserving.



SDP NPF DPF DPFLB C

Figure 6.4: *Plots of* $(d_{ij}, \widehat{d_{ij}})$ *for all* $i < j$ *for a single reconstruction when the matrix to be completed contained only minimal spanning tree distances (shown in red); the* $y = x$ *line indicates perfect matching; the box in each plot shows the extent of the original distances.*

In marked contrast, the guided random search `C` produces a completion whose distances $\widehat{d_{ij}}$ are all within the range of the true distances d_{ij} and are much more nearly concentrated around the $y = x$ line. If anything, `C` seems more inclined to produce smaller distances than are necessary. This might be corrected by having the vectors generated in Algorithm 3 not be generated on a sphere uniformly but to favour directions that would increase distances to other points already in the tree.

6.1.3 Reproducing the Minimal Spanning Tree

To see how well the various methods reproduced the minimal spanning tree, the spanning tree distances $\widehat{d_i}$ for $i = 1, \ldots, n-1$, and adjacency matrix $\widehat{\mathbf{A}}$ were determined from each completion $\widehat{\mathbf{D}}$. Figure 6.5 shows all completions by all methods where each horizontal location is the proportion of edges in $\mathbf{A}$ which also appear in $\widehat{\mathbf{A}}$ and each vertical location is

$$\frac{(\sum_{i=1}^{n-1} d_i - \sum_{i=1}^{n-1} \widehat{d_i})^2}{\sum_{i=1}^{n-1} d_i^2}$$

where d_i for $i = 1, \ldots, n-1$ are the original minimal spanning tree distances of $\mathbf{A} \circ \mathbf{D}$ and $\widehat{d_i}$ for $i = 1, \ldots, n-1$ are those from $\widehat{\mathbf{A}} \circ \widehat{\mathbf{D}}$ for that completion.

The mst-preserving completions are those in the bottom right of Figure 6.5 – `DPFLB`, `C` , and `SDP` (all values on both measures but have been separated vertically to better distinguish their shapes from the others methods). Of these, two were designed to be mst-preserving and so should appear here; each of these two points actually represent hundreds or tens of thousands of completions which must return the same minimal spanning tree. In contrast, the point for

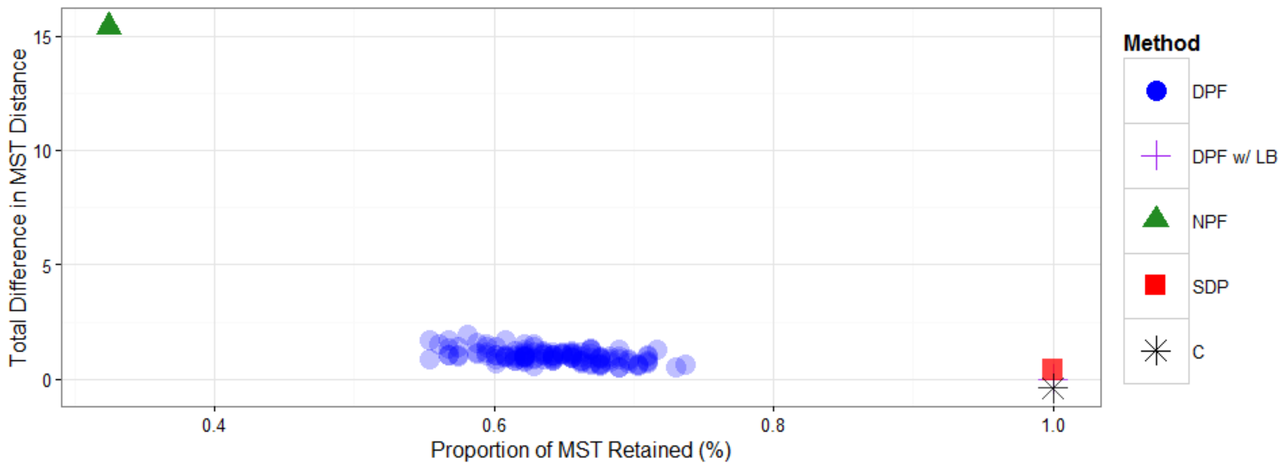


Figure 6.5: *Square of the total difference in minimal spanning tree distances as a multiple of the total squared minimal spanning tree distances versus the proportion of minimal spanning tree edges that were retained. All completions of all five methods are shown. The three points at the right have identical values but have been given different vertical positions to better distinguish the points in the plot.*

`SDP` is a singleton point representing the one completion actually constructed for `SDP`. This `SDP` completion has turned out to be mst-preserving, though not by design. As seen in Figures 6.3 and 6.4, `SDP` tends to produce very large distances in its completions, and these distances increase as the percentage missing increases. This would explain why `SDP` is mst-preserving here.

The 466 identical completions of `NPF` all appear in the top left corner of Figure 6.5 and show `NPF` to be the poorest performer in terms of preservation of the minimal spanning tree. The 124 completions by `DPF` are spread across the bottom, retaining about 55-75% of the edges in the minimal spanning tree and matching the distances fairly closely (at least compared to `NPF`).

6.1.4 Reproducing the Point Configurations

We now examine the point configurations produced by all five methods for a single reconstruction from the minimal spanning trees.

Procrustes analysis [46] finds the best rotation (and possibly translation; no rescaling here) of the constructed point configuration to best match that of the target original point configuration. The quality of the reproduction is measured by the total sum of squares of the differences between the original and the optimally rotated and translated constructed point configuration (i.e. the squared Frobenius norm of the matrix difference).

The results for each method are shown in Figure 6.6; smaller values indicate closer reproductions of the original point configuration. Each of NPF and SDP produced a single solution and their respective Procrustes squared errors are given by the dashed and dot-dashed vertical lines. Clearly SDP produces a point configuration least like the original and NPF the next worst at a little more than half the squared error. Each of the remaining three methods produced multiple solutions whose Procrustes squared error values are shown as density estimates. The bottom two panels of Figure 6.6 show the densities of the DPF and the DPFLB methods. These two densities overlap considerably with DPF producing slightly better point configurations but at the cost of not necessarily preserving the MST. The guided random search C appears in the top panel. It preserves the MST and with high probability produces a closer point configuration than either of DPF or DPFLB. Still, there is small but nonzero probability that the method C will produce a point configuration farther from the original than does its nearest competitor DPFLB.

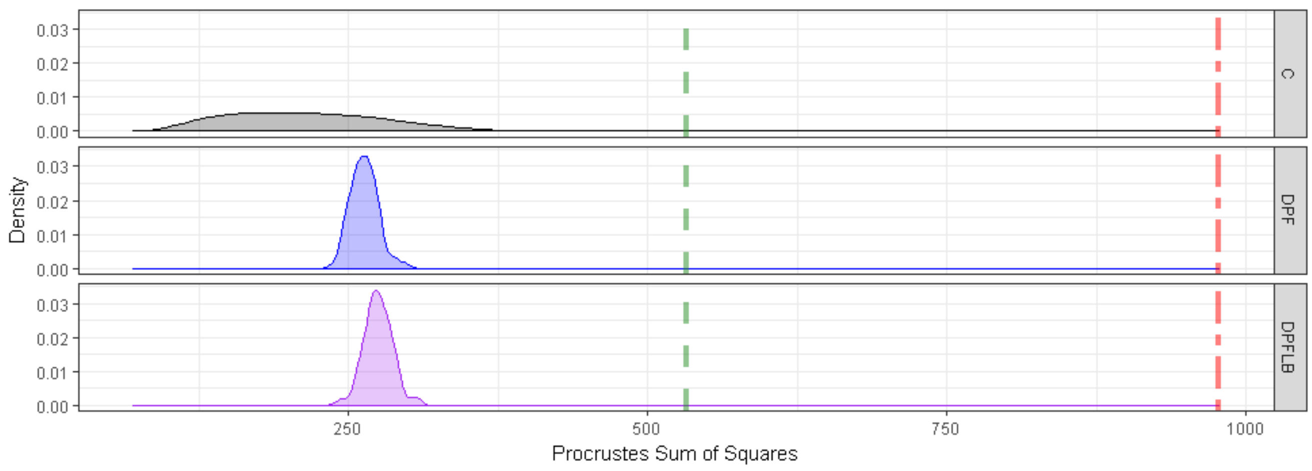
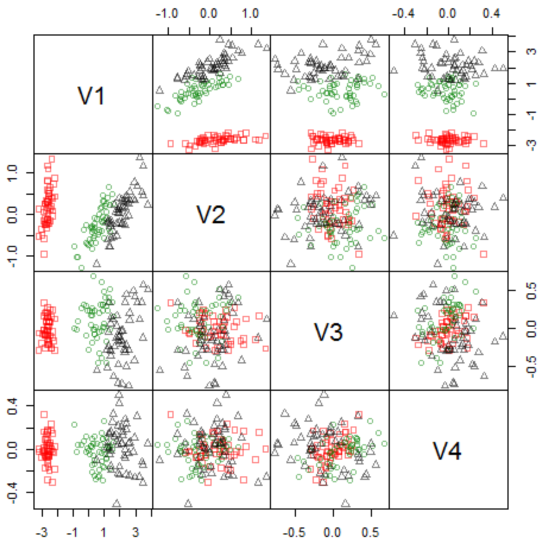


Figure 6.6: *Procrustes analysis (via* **procrustes()** *from the* **R** *package* **vegan** *[82]) of the Iris data point configurations reconstructed in the original four dimensions given by each method. The error is measured by "Procrustes Sum of Squares" which is the squared Frobenius norm of the difference between the original point configuration and each one constructed by the various methods (and optimally rotated and translated to best match the original). Each of C, DPF, and DPFLB are shown as density estimates based on their many reproduced configurations; each of NPF (dashed) and SDP (dot-dashed) produce a single reconstructed point configuration and so are shown as vertical lines.*

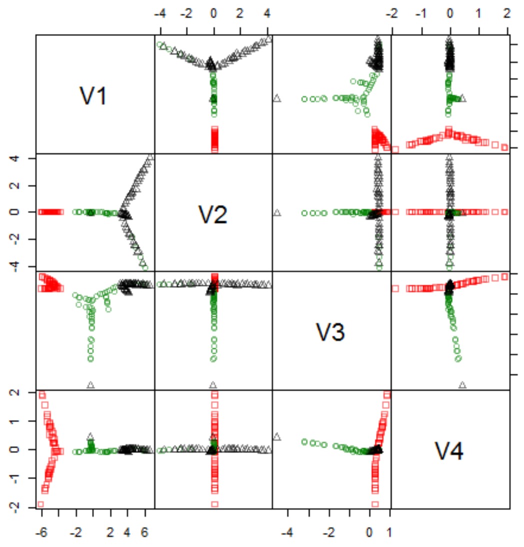
To standardize the comparisons, the Iris data is transformed to its principal directions from a singular value decomposition of $\mathbf{X} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^{\mathsf{T}}$ as $\mathbf{XV}$. This projects each point $\mathbf{x}_i$ onto a new coordinate system given by the new variates $V1$, $V2$, $V3$, and $V4$.

Figure 6.7(a) shows the original Iris data in a scatterplot matrix for this new coordinate system. Each scatterplot shows the transformed data for the pair of variates given by the diagonal entries from the same row and column as the scatterplot. The three different point colours and shapes identify the three different species of Iris. There are 150 points.

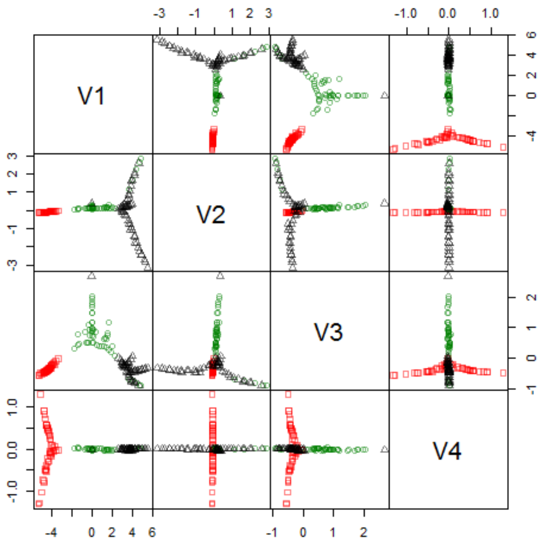
For each completion method, a single completion $\widehat{\mathbf{D}}$ is taken with embedding dimension $p = 4$,



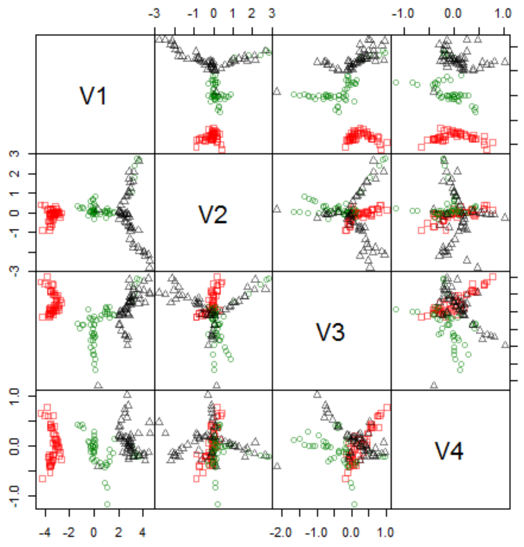
(a) Original



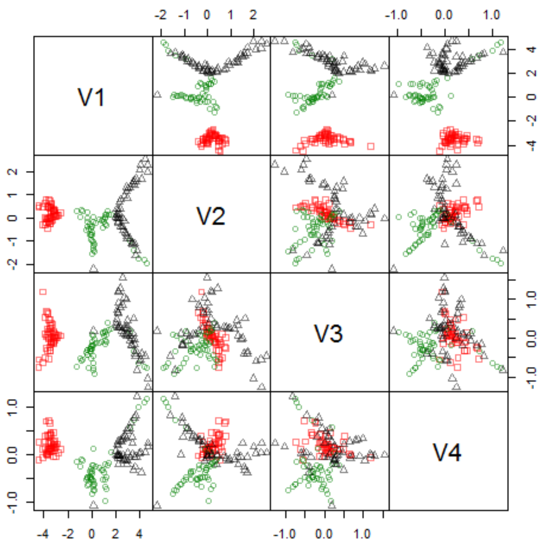
(b) SDP



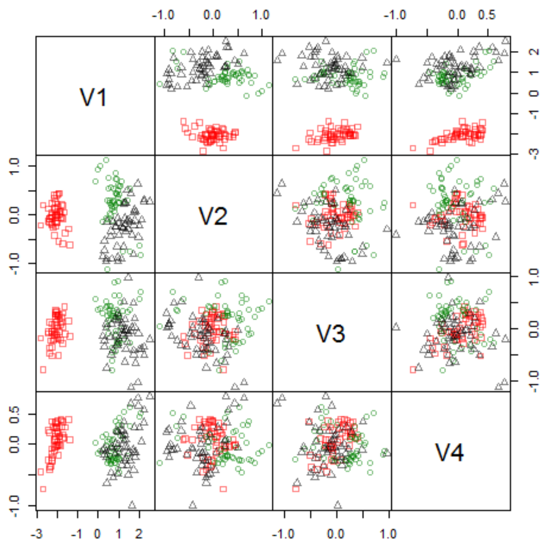
(c) NPF



(d) DPF



(e) DPFLB



(f) C

Figure 6.7: *Iris data point configurations reconstructed in the four dimensions given by each configuration's principal coordinates. The three species of flower are distinguished both by colour and by shape of the point symbols.*

the singular value decomposition of its estimated point configuration $\widehat{\mathbf{X}} = \widehat{\mathbf{U}}\widehat{\mathbf{\Lambda}}\widehat{\mathbf{V}}^{\mathsf{T}}$ determined, and the transformed point configuration $\widehat{\mathbf{X}}\widehat{\mathbf{V}}$ plotted in a scatterplot matrix with the same point symbols. The transformed point configurations for the five completion methods are shown as Figures 6.7(b–f).

As can be seen, SDP and NPF produce star-shaped configurations of straight lines. So too does DPF although the shapes are slightly noisier. DPFLB is very much like DPF but may be slightly noisier again. Most striking is the configuration produced by the guided random search C; of the five methods considered, C produces a configuration most like that of the original data. We note also that the axis ranges in these scatterplots reflect the size of the corresponding singular values from the configurations. These are consistent with the remarks made earlier about the comparative size of the distances produced by each method in Figure 6.4. Again, the configuration of C appears to be closest to the original in this respect as well. Finally, all completions appear to preserve much of the group separation seen in the original data between the three species of flower. This is not too surprising since the distances from the minimal spanning tree were given. The minimal spanning tree is the basis for many hierarchical clustering methods. The two methods designed to preserve the minimal spanning tree should fare best in maintaining separation of clusters.

6.2 Reconstructing Data $\sim U[0,1]^p$

Here n locations $\mathbf{x}_i$ are simulated independently from $U[0,1]^p$ for varying values of p to give an $n \times p$ point configuration $\mathbf{X}$ within the unit hypercube in $\mathbb{R}^p$. For each p, five different matrices of point configurations $\mathbf{X}_1, \ldots, \mathbf{X}_5$ are generated allowing for variability in results to be observed.

As with the Iris data, the performance of the five methods will be compared but only completions from the distances of the minimal spanning tree of each configuration are considered. Of interest then is how this performance might depend on varying dimensionality p rather than the percentage of distances missing.

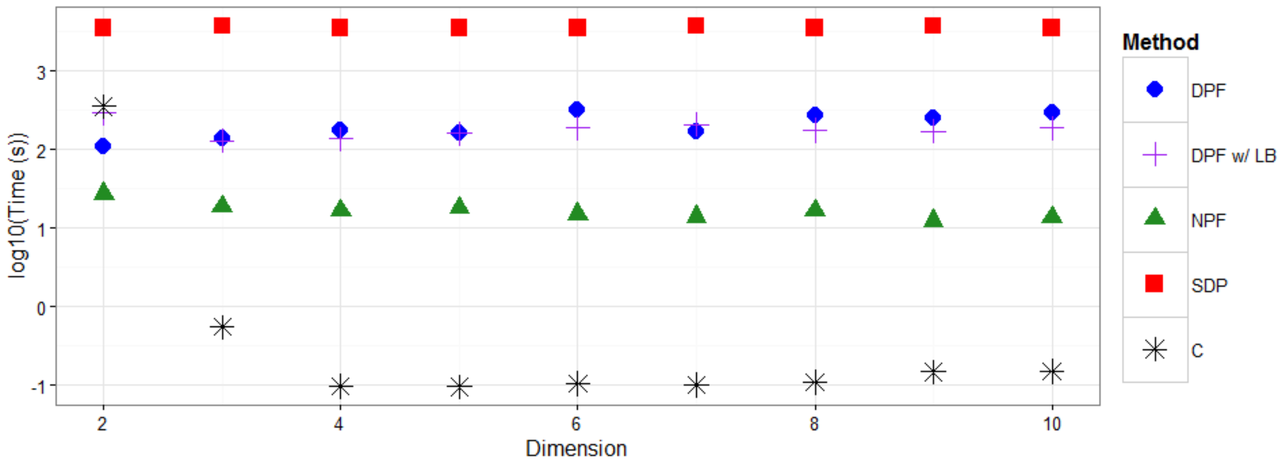
In what follows, $p \in \{2, 3, 4, \ldots, 10\}$ will be used. The ith configuration matrix of dimension $q \leq p$ will be constructed as the first q columns of $\mathbf{X}_i$ where $\mathbf{X}_i$ is the $n \times p$ matrix whose rows were independently generated from a $U[0,1]^{10}$ distribution. Throughout, we take $n = 100$.

6.2.1 Completions as a Function of Dimension

Figure 6.8 shows the effect of varying dimension p on the time to completion (top plot) and on the accuracy, as measured by RDD of Equation (6.1) using all distances, for the three algorithms. The completion times are from a single matrix ("matrix 1") for each dimension; the accuracies are shown for each of the five different simulated matrices (marked 1 through 5) with methods distinguished by line type and colour. Recall that a lower dimensional matrix shares its columns with all higher dimensional ones.

For time to completion, there is a clear ordering of methods from the least efficient `SDP` to the several orders of magnitude more efficient `C`. Both `DPF` and `DPFLB` take about the same time and `NPF` is second fasted though still two orders of magnitude slower than `C`. The only exception to this ordering occurs for $p = 2$. There the two mst-preserving methods are slower than both `DPF` and `NPF`; this is likely due to the increased difficulty in finding completions which preserve the minimal spanning tree for uniformly generated data in only two dimensions. As p increases, `SDP` stays relatively constant in computation time, `NPF` decreases slightly, both `DPF` and `DPFLB` tend to increase, and `C` drops quickly as it becomes easier in larger dimensional spaces to find random directions that work.

In terms of accuracy, all methods degrade as dimensionality increases with the notable exception of `NPF` whose accuracy improves. Unfortunately, `NPF` does not preserve the minimal spanning tree. The others in order from least to most accurate over all dimensions are `SDP`, `DPF` and `DPFLB` (about the same), and `C`. Note again that the logarithm has been taken of the average RDDs so these differences can be substantial.



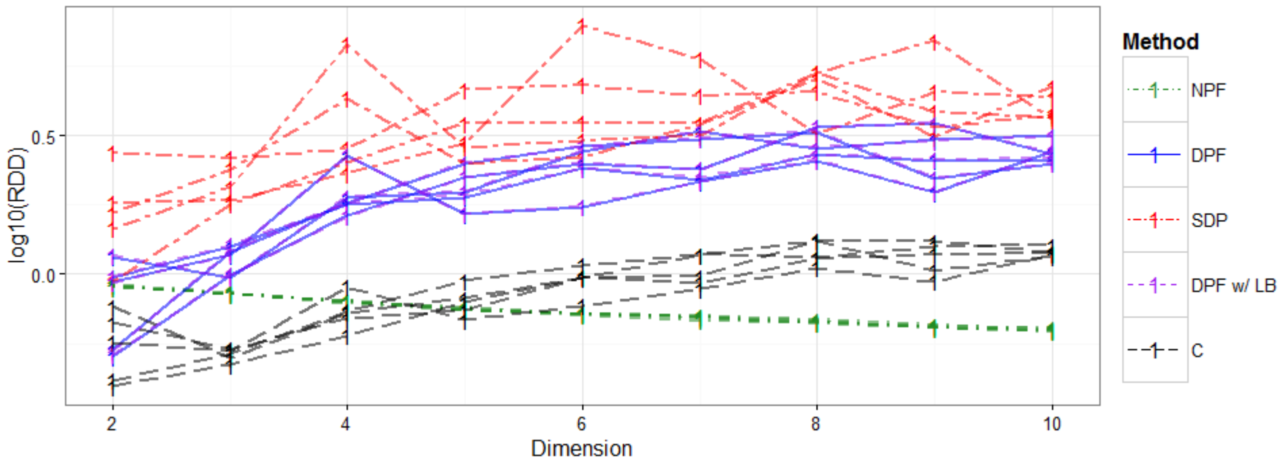
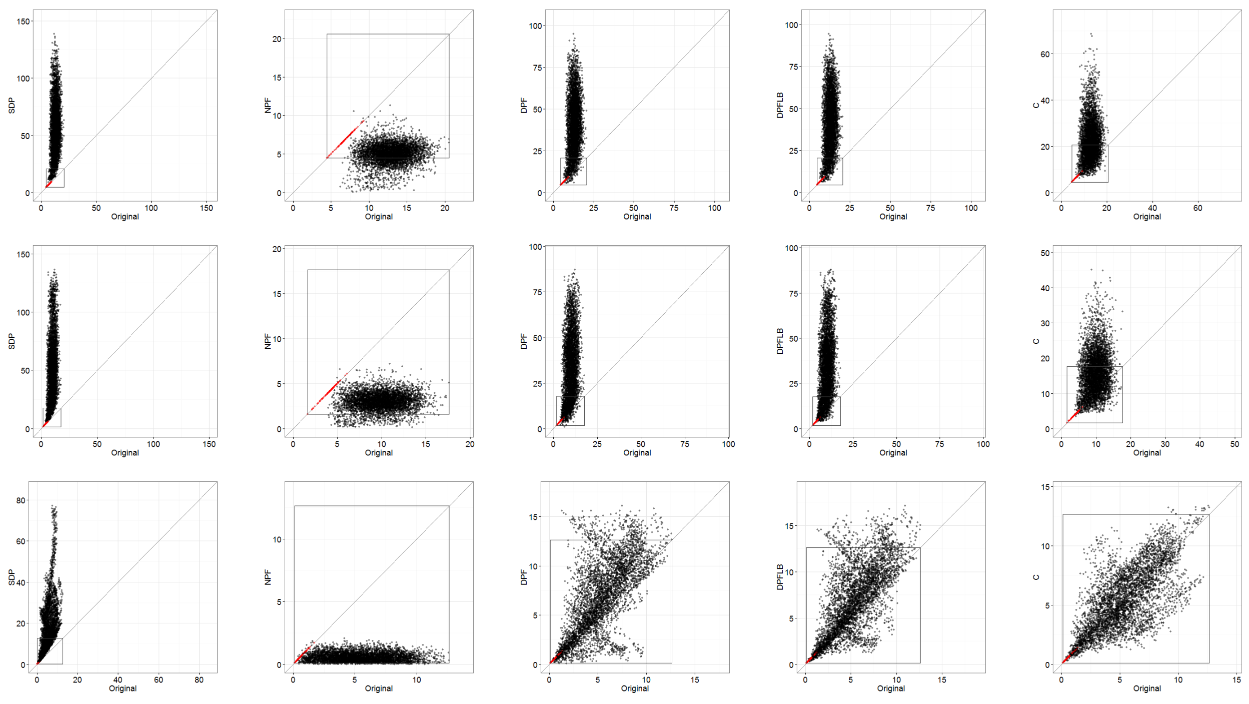


Figure 6.8: *Performance of the five completion matrices. Top plot shows base 10 logarithms of the average times to complete ($n = 1$ for* `SDP`*, $n = 466$ for* `NPF`*, $n = 124$ for* `DPF`*, $n = 114$ for* `DPFLB`*, $n = 39,935$ for* `C`*) from a single matrix ("matrix 1"). The bottom plot shows $\log_{10}$ of the average RDDs for these completions from all five matrices (marked 1 to 5).*

6.2.2 Distances as a Function of Dimension

Figure 6.9 shows the completed distances $\widehat{d_{ij}}$ of each of the five methods paired with the actual distances d_{ij} for one completion of matrix 1 having dimension $p = 2, 6, 10$. The box within each plot shows the extent of the distances d_{ij} and is identical in absolute magnitude across all plots. The five methods appear as columns; the three rows show increasing dimensionality $p = 2, 6, 10$ from bottom to top.



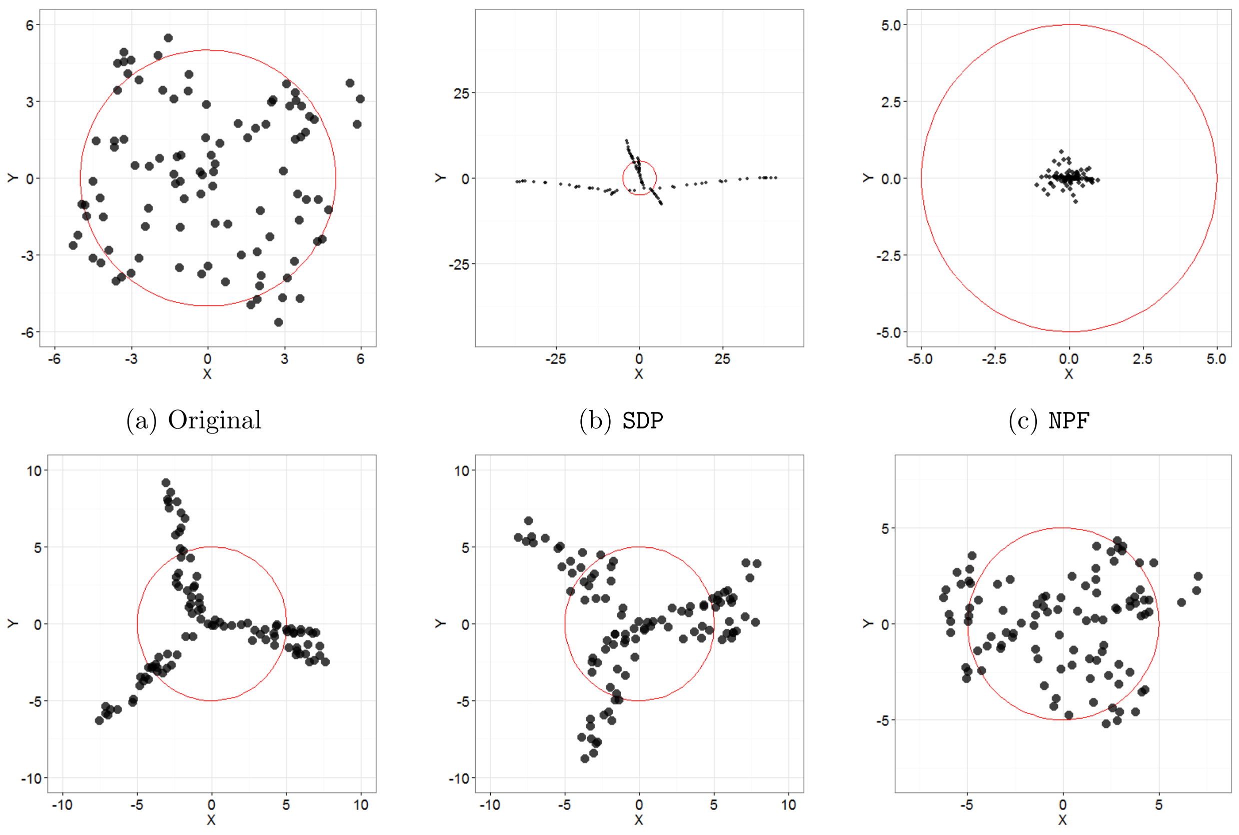
(a) SDP (b) NPF (c) DPF (d) DPFLB (e) C

Figure 6.9: *Plots of* $(d_{ij}, \widehat{d_{ij}})$ *for all* $i < j$ *for a single reconstruction when the matrix to be completed contained only minimal spanning tree distances (shown in red); the* $y = x$ *line indicates perfect matching; the box in each plot shows the extent of the original distances. Results for all five methods are shown in each column; rows, from bottom to top, show increasing dimensionality of* $p = 2, 6, 10$.

Consider the bottom row where $p = 2$. Here four of the five methods produce most, if not all, distances within the box given by the range of the original distances d_{ij}. Of the two mst-preserving completions, `C` gives distances that are roughly symmetric about the $y = x$ line and clustered near it; a relatively few large distance appear outside the box at the top right. In contrast, `DPFLB` produces distances that are more often above the $y = x$ line than below and are outside the box at top all along the range of the original distances. The completion `DPF` is similar to `DPFLB` but will not necessarily reproduce the minimal spanning tree.

Most unusual in the bottom row are `SDP` and `NPF`. Already for $p = 2$, `SDP` produces huge distances $\widehat{d}_{ij}$, far outside the box. Moving up column (a) as the dimensionality increases, `SDP` produces larger and larger distances, much larger than the original distances and larger than those produced by any other completion method. In contrast, `NPF` produces small distances, essentially all lying below the $y = x$ line. This continues to be the case as the dimension increases; distances produced by `NPF` ever smaller as p increases with many becoming much smaller than any of the original distances d_{ij}. This is the opposite of each of the other four methods whose distances become larger and larger as p increases. This contrast explains why in Figure 6.8 `NPF` showed little variation in RDD compared to the other methods. Because `NPF` produces small distances bounded below by 0, RDD is bounded for `NPF` but not for the others.

For all dimensions the guided random search method `C` produces distances that are closer to the original distances than any of the other four methods.



(a) Original (b) `SDP` (c) `NPF`

(d) `DPF` (e) `DPFLB` (f) `C`

Figure 6.10: *Uniform two dimensional data: the original and those reconstructed by the five methods. Coordinates are the principal coordinates for each configuration. The red circle has the same centre and diameter in all plots.*

6.2.3 Reproducing the Minimal Spanning Tree

Both `C` and `DPFLB` reproduce the minimal spanning tree by design, as does `SDP`, except in this case largely by accident. As Figure 6.9 shows, the distances produced by `SDP` are typically so large that they do not change the minimal spanning tree. The same effect can be seen with `DPF` where, because ever larger distances are produced as p increases, the proportion of the minimal spanning tree retained by `DPF` completions increases with p. Completions by `NPF` on the other hand do not preserve the minimal spanning tree. The small distances produced by `NPF` interfere with the minimal spanning tree.

6.2.4 Reproducing the Point Configurations

To compare point configurations, consider a completion from each of the five methods for matrix 1 when $p = 2$ and $p = 6$. The data were first rotated to their principal coordinates as described in Section 6.1.4.

Figure 6.10 shows the original data in (a) and the reconstructed configurations in (b)–(f); each red circle has the same centre and radius in each plot. Figure 6.10 reproduces the findings from the bottom row of Figure 6.9 in that `SDP` produces unusually large distances, `NPF` unusually small distances, and the mst-preserving methods `DPFLB` and `C` produce distances closer to those of the original data, with `C` being the closest.

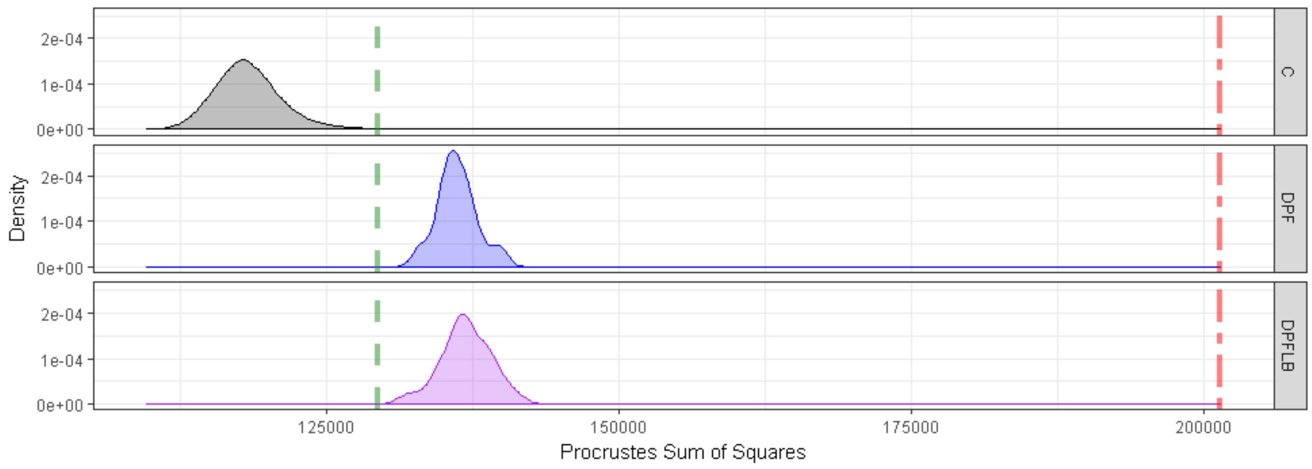


Figure 6.11: *Each of **C**, **DPF**, and **DPFLB** are shown as density estimates based on their many reproduced configurations; each of **NPF** (dashed) and **SDP** (dot-dashed) produce a single reconstructed point configuration and so are shown as vertical lines.*

As was the case with the Iris data, Figure 6.10 shows again that the completion methods tend to concentrate points near lines. The mst-preserving `DPFLB` spreads the configuration out more than does DFP but not nearly as much as does C. For $p = 2$ C produces a point configuration that is more like the original data than any of the others.

As with the Iris data, consider a Procrustes analysis on the constructed point configurations in two dimensions for each method. The results are given in Figure 6.11.

Again, SDP produces the worst configurations following its tendency to have large distances in its completions; conversely, NPF performs very well in this example possibly because of its tendency to have small distances which works in its favour when the points are confined to a unit hypercube. The methods DPF and DPFLB perform equally well though not as good as NPF. The constructive method C clearly outperforms all other methods.

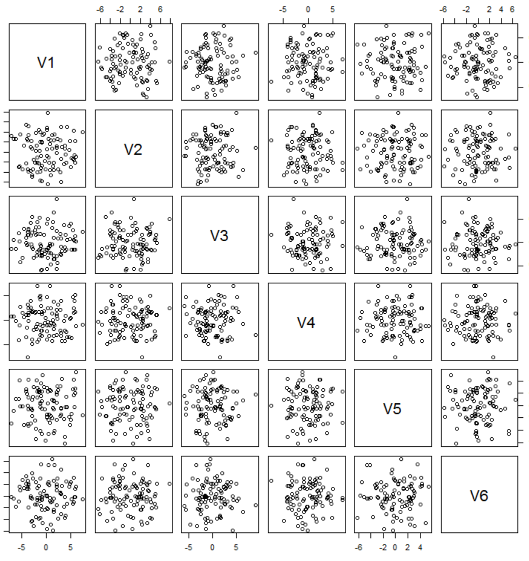
Figure 6.12 shows the point configurations produced when $p = 6$. Much the same patterns prevail as were seen earlier with $p = 2$ and also when $p = 4$ for the Iris data. SDP has large distances and strongly linear configurations; NPF has small distances but also exhibits star-shaped linear structure; DPF and DPFLB also show linear structure and some star shape; C shows a dispersed configuration most like the original data but also has outlying points in the $V1$ direction.

6.3 Concluding Remarks

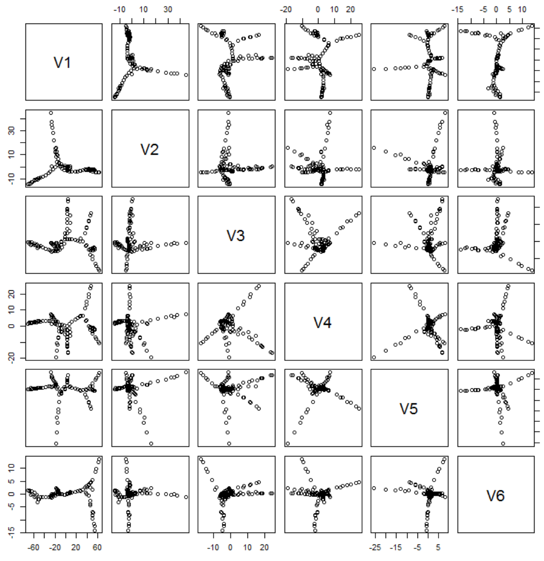
Each of the experiments performed in this chapter were designed to measure the relationship between the sparsity of the Euclidean distance matrix to be completed and the accuracy of both the completed Euclidean distance matrix and the corresponding point configuration. The limiting case of knowing only the distances in the minimal spanning tree was also considered, where additional interest lay in preserving the minimal spanning tree in the completed distance matrix.

Using the Anderson Iris data and a simulated uniform configuration, it was shown that as the sparsity of the partial Euclidean distance matrix increased, the time taken by the various algorithms to complete the distance matrix increased for all algorithms, and the accuracy of the resulting distances (as compared to the original matrix) decreased. In terms of relative performance, the semidefinite programming algorithm performed the worst, the dissimilarity parameterization formulation second, and non-convex position formulation was the best performing.

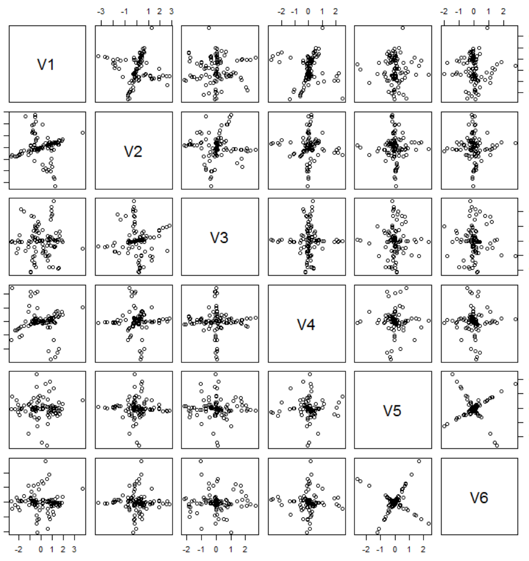
In analyzing the preservation of the minimal spanning tree in the completion of a Euclidean distance matrix, two new algorithms were introduced, the dissimilarity parameterization formulation with minimal spanning tree preserving lowers bounds, and the guided random search algorithm. In terms of minimal spanning tree retention, and overall accuracy of the resulting completed distance matrix and point configuration when knowing only the minimal spanning tree, these two algorithms were the best performing. This was particularly evident when viewing the configurations from the resulting completions, where the guided random search in particular most closely resembled the original configurations. This was confirmed using a procrustes analysis, which showed that the guided random search algorithm most often produced a configuration with the



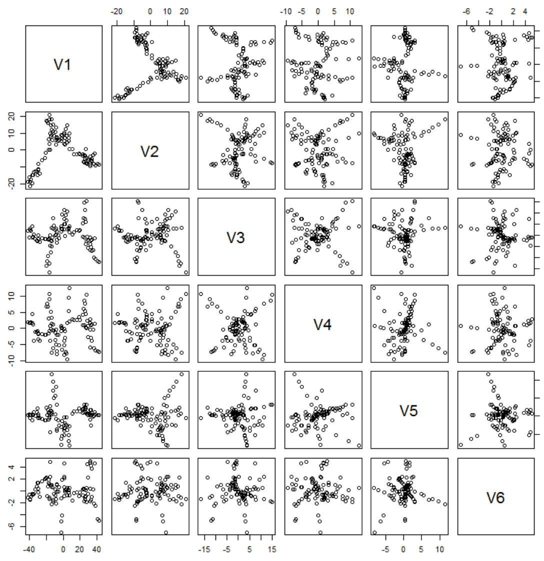
(a) Original



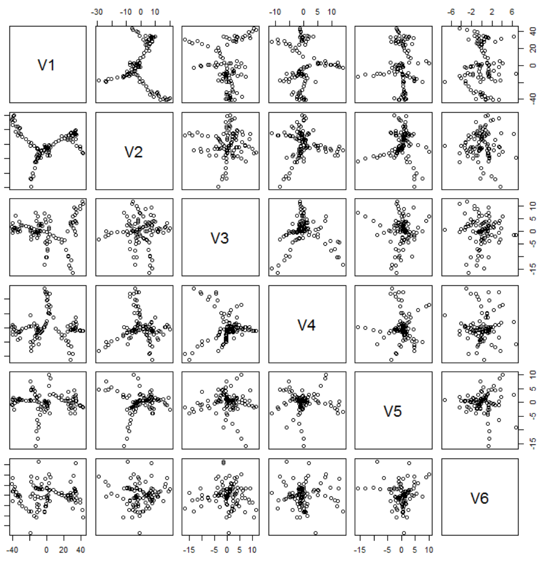
(b) `SDP`



(c) `NPF`



(d) `DPF`


(e) DPFLB

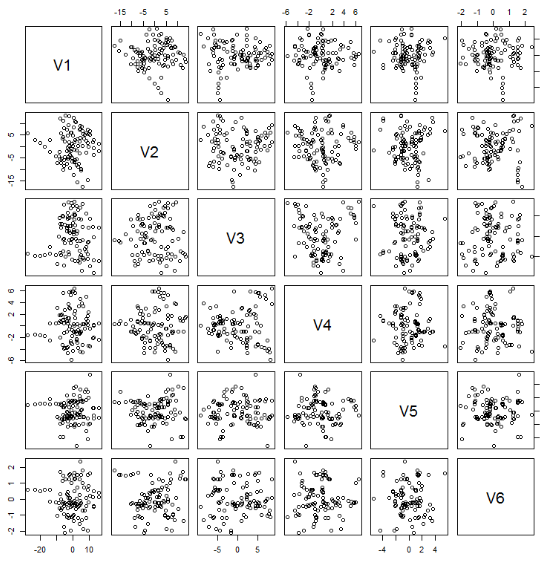

(f) C

Figure 6.12: *Uniform within a six-dimensional hypercube. Data point configurations reconstructed in six dimensions using each reconstruction's (including the original data) principal coordinates.*

lowest procrustes sum of squares, indicating it most closely matched the original configuration.

Unexplored in this chapter is the generality of the guided random search algorithm, and its ability to create configurations with a known minimal spanning tree structure. Also to be examined is the effect of restricting the proposed angle when a new point is added to the configuration. Both of these will be explored in Chapter 10.

Chapter 7

Binning: For More Than the Birds

Big Data, the often used buzz word referring to data with both a large number of observations (n) and a large number of dimensions (p), has given rise to a large volume of research in reducing the inherent size of the data before analysis is undertaken. Much of this work has been in reducing the *dimension* of the data - that is, for a data set of dimension p, reduce the dimension to some $d < p$, while preserving as much of the underlying *structure* of the data as possible. There are several well known algorithms for dimension reduction, including isomap [103], local linear embedding [93], and Laplacian eigenmaps [9], among others.

Of equal importance is the idea of reducing the number of observations in a data set - decreasing n to some $m < n$, while again preserving as much of the underlying structure as possible. Decreasing the size of n in a meaningful way is attractive due to the complexity of many statistical analyses, which are often dependent on the size of n non-linearly. By decreasing n, while preserving the underlying structure of the data, these analyses could be executed in less time, and with similar accuracy as compared to the original. Typically, this is done through *binning*, which generally entails grouping nearby points, and representing each group as a single representative point.

In Chapter 1, it was noted that recent advancements in binning were mostly centred around the fields of classification and kernel density estimation, and that the algorithms developed in these fields were not appropriate for binning a general point configuration. This left only a small subset of the available binning algorithms for consideration, including equal width [121], equal frequency [121], fixed frequency [123], hexagonal binning [18], and random sampling. Each of these methods are discussed in Section 7.1.

In addition to these methods, a new method, *tree-based binning*, is proposed in Section 7.2. The concept of tree-based binning was inspired by classification and regression trees [40], which seek to partition the domain (or *feature space* in the machine learning literature) of a point configuration into a set of rectangular subspaces. The goal is to then fit a model in each rectangle to accurately predict some response variable $\mathbf{y}$ from a set of explanatory variables $\mathbf{x}_1, ..., \mathbf{x}_n$. While not immediately applicable to general data as the optimal splitting criteria depends on a response or class label $\mathbf{y}$, the concept of tiling the domain of the data using rectangles based on data-drive measures is attractive. This non-supervised data driven partitioning process forms the basis of tree-based binning.

What remains to be explored then is which (if any) of these algorithms are capable of reducing the size of a configuration with minimal damage to both the underlying probabilistic and geometric structure of the data. Also of importance is to explore the effect of increasing dimension on each of the binning algorithms, to see if their performance degrades as dimension increases. These questions will be addressed via experimentation in Chapter 9.

7.1 Existing Binning Techniques

One application of binning has been in machine learning, where it is specifically used to discretize continuous attributes for use in classification. A sample of the algorithms used in the experiments of [42] are given in Table 7.1. For a more complete taxonomy of current binning techniques, see [42].

Method	**Reference**
Multivariate Discretization	[8]
MODL	[10]
Distance-based Discretizer	[19]
Class Attribute Interdependent Maximization	[20]
Zeta	[55]
Minimum Description Length Principle	[35]
ChiMerge	[64]
Unsupervised Correlation Preserving Discretizer	[77]
Modified Chi2	[102]
Equal Frequency	[121]
Equal Width	[121]
Fixed Frequency Discretizer	[123]
Proportional Discretizer	[123]
FUSINTER	[131]

Table 7.1: *A list of some of the discretization algorithms used in the experiments of [42].*

In general, binning techniques can be subdivided into two major categories, supervised and unsupervised, and subdivided further into top-down and bottom-up discretizers. Algorithms from each of these subdivisions will be discussed in this section. Also considered is a special case of a two-dimensional binning algorithm, hexagonal binning [18].

7.1.1 Supervised

Supervised binning is most often used during the discretization of continuous attributes that will be used as input into a machine learning or data mining algorithm [64]. As the term suggests, these supervised methods require data in the form $(\mathbf{y}, \mathbf{x})$, where $\mathbf{y}$ is a class tag or label, and $\mathbf{x}$ is the variable to be discretized. Two of the best performing of these supervised algorithms (as measured by the analysis of [42]) are the ChiMerge algorithm [64], and the Zeta algorithm [55]. These discretizers are also examples of bottom-up and top-down discretization methodologies respectively.

ChiMerge

The ChiMerge algorithm [64] is a simple two-step discretization algorithm, consisting of an initialization step, and a bottom-up merging step. A bottom-up merging algorithm is a process that arrives at a final discretization by merging bins, as opposed to splitting bins.

Consider data of the form $(\mathbf{y}, \mathbf{x})$, where $\mathbf{x}$ is the (one dimensional) attribute to be discretized and $\mathbf{y}$ is the corresponding class label. During the initialization step, the data are sorted on their attribute level, so that $x_{(1)} \leq x_{(2)} \leq ... \leq x_{(n)}$. The attribute is then initially discretized such that each $x_{(i)}$ is placed in its own interval. This is achieved by placing a cutpoint at $\frac{x_{(i)}+x_{(i+1)}}{2}$, for $i = 1, ..., n-1$.

Once the initial discretization is completed, the merging process begins, which is a two step process, repeated continuously

1. Compute the value of χ^2 for each pair of adjacent cells
2. Merge the pair of adjacent cells with the lowest value of χ^2

The value of χ^2 for each adjacent cell is computed as follows:

$$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{k} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

where:

$$\begin{aligned}
i &= \text{ Intervals being compared} \\
k &= \text{ Number of classes} \\
O_{ij} &= \text{ Number of points in the } i^{th} \text{ interval and } j^{th} \text{ class} \\
R_i = \sum_{j=1}^{k} O_{ij} &= \text{ Number of points in the } i^{th} \text{ interval} \\
C_j = \sum_{i} O_{ij} &= \text{ Number of points in the } j^{th} \text{ class} \\
N = \sum_{j=1}^{k} C_j = \sum_{i} R_i = \sum_{i=1}^{2} \sum_{j=1}^{n} A_{ij} &= \text{ Total number of points} \\
E_{ij} = \frac{R_i C_j}{N} &= \text{ Expected Number of points in } i^{th} \text{ interval, } j^{th} \text{ class}
\end{aligned}$$

The steps above are repeated until all the calculated values of χ^2 are above a pre-determined threshold, or until the maximum number of intervals have been created. In determining the threshold, a desired level of significance is chosen based on a chi-square distribution with degrees of freedom of $k-1$. Typically, the recommended significance level is set to .9, .95, or .99, and the maximum number of intervals is set to 15 [64].

Zeta

Unlike *ChiMerge*, the *Zeta* algorithm [55] is a top-down discretization algorithm, meaning it begins with the data in one large bin and then splits it into k smaller bins according to some criteria.

The *Zeta* discretization process is based on a measure ζ designed to capture the strength of association between nominal variables. As pointed out in [55], the easiest way to understand what *Zeta* is designed to capture is to consider a simple case of two dichotomous variables A & B, whose sample distribution is given as follows:

	A_1	A_2
B_1	n_{11}	n_{12}
B_2	n_{21}	n_{22}

Now, consider using the variable A to predict the variable B. There are two possible cases to consider if each value of A is used to predict a different value of B. Either we associate B_1 with A_1 and B_2 with A_2, or we associate B_1 with A_2 and B_2 with A_1. If the first association is used, then the number of correct predictions is $n_{11} + n_{22}$. If the second association is used, then the number of correct predictions is $n_{12} + n_{21}$. The association that results in the largest number of correct predictions is chosen, so define ζ as

$$\zeta = \frac{max(n_{11} + n_{22}, n_{12} + n_{21})}{N}$$

where $N = n_{11} + n_{12} + n_{21} + n_{22}$. In general, ζ is defined as

$$\zeta = \frac{\sum_{i=1}^{k} n_{f(i)i}}{N}$$

where $f(i)$ is defined to be the pairing assignment, associating A_i with $B_{f(i)}$, that results in the highest number of correct predictions.

In practice, given a classification variable of size k, Zeta seeks the $k-1$ cutpoints on a continuous random variable $\mathbf{x}$ that result in the largest value of ζ. It is impractical to determine the optimal $k-1$ cutpoints, as the number of possibilities is very large, so a simple heuristic is employed, described in detail in [55], where the $\mathbf{x}$ variable is continuously divided assuming each partition is dichotomous.

7.1.2 Unsupervised

Unlike the supervised algorithms considered above, unsupervised discretization methods do not require a class tag, allowing for application to a wide range of data. Classic unsupervised discretization techniques include equal width and equal frequency binning [121], and fixed frequency binning [123]. Also considered is a special case of a related two-dimensional binning technique known as hexagonal binning [18]. Finally, while not a binning technique, random sampling is a very simple way to reduce the number of observations in a configuration.

Equal Width

Under equal width binning, the (one-dimensional) attribute which is being discretized, $\mathbf{x}$, is split into k intervals (using $k-1$ cutpoints) of equal width. Consider the minimum and maximum value of the attribute $\mathbf{x}$, $(x_{(1)}, x_{(n)})$. The width of each interval is calculated as

$$W = \frac{x_{(n)} - x_{(1)}}{k}$$

The cutpoints are then calculated as

$$cut_i = x_{(1)} + i * W, \quad i = 1, ..., k-1$$

This results in the discretization intervals $[x_{(1)},\ cut_1)$, $[cut_1,\ cut_2)$,..., $[cut_{k-1},\ x_{(n)}]$. In each bin, the *representative point* (i.e. the observation that replaces all the observations in the bin) is generally taken to be the by-dimension mean of the observations in each bin. This process can be repeated in each dimension for multivariate data. Figure 7.1 illustrates the equal width binning algorithm applied to a bivariate normal point configuration.

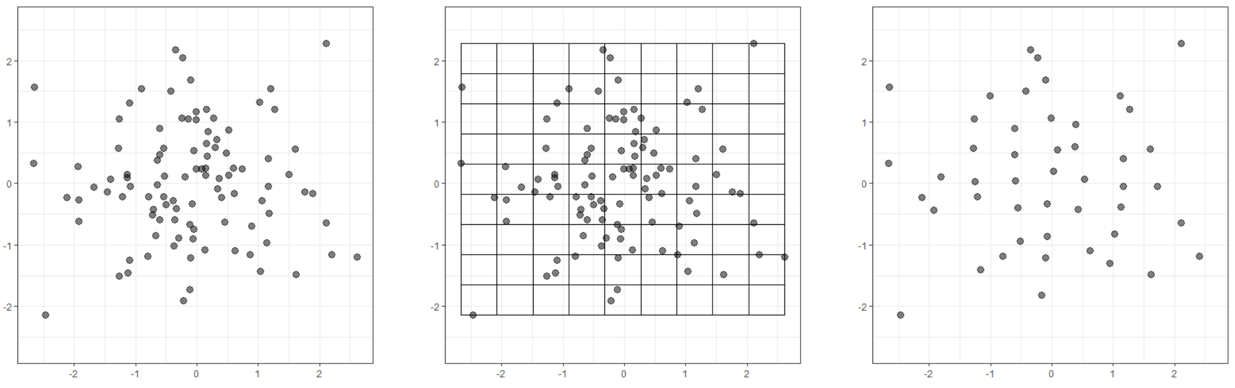


Figure 7.1: *An example of the equal width binning method reducing a bivariate normal configuration from 100 points to 47.*

Equal Frequency

Under equal frequency binning, the (one-dimensional) attribute being discretized, $\mathbf{x}$, is split into k intervals (using $k-1$ cutpoints), such that each bin contains the same number of points. Assume $\mathbf{x}$ has n points, sorted in ascending order such that $\mathbf{x} = \{x_{(1)}, ..., x_{(n)}\}$, and for simplicity, assume $\frac{n}{k}$ is an integer. Then, to place $k-1$ cutpoints such that each bin contains the same number of points, define $p_i = i \cdot \frac{n}{k}$. The cutpoints are then computed as

$$cut_i = \frac{x_{(p_i)} + x_{(p_i+1)}}{2}, \quad i = 1, ..., k-1$$

This results in the discretization $[x_{(1)},\ cut_1)$, $[cut_1,\ cut_2)$,..., $[cut_{k-1},\ x_{(n)}]$, where again the representative point is taken to be the by-dimension mean of each bin. Like the equal width algorithm, this process can be repeated in each dimension for multivariate data. Figure 7.2 illustrates the equal frequency binning algorithm applied to a bivariate normal point configuration.

Note that if $\frac{n}{k}$ is a non-integer (which is a frequent occurrence), $p_i = i \cdot \lfloor \frac{n}{k} \rfloor$, and the remaining points can be assigned to a bin in any number of ways. For instance, if there are 102 points which are to be divided into 10 bins (meaning 10 points per bin), two potential solutions for assigning the remaining points are

1. Assign the two extra points to each of the first two bins. This would yield bins of size $[11, 11, 10, 10, 10, 10, 10, 10, 10, 10]$.
2. Assign the two extra points to bins at random. For instance, they could be assigned to bins 4 and 7, yielding bins of size $[10, 10, 10, 11, 10, 10, 11, 10, 10, 10]$

In either case, the cutpoints for these bins would need to be adjusted accordingly.

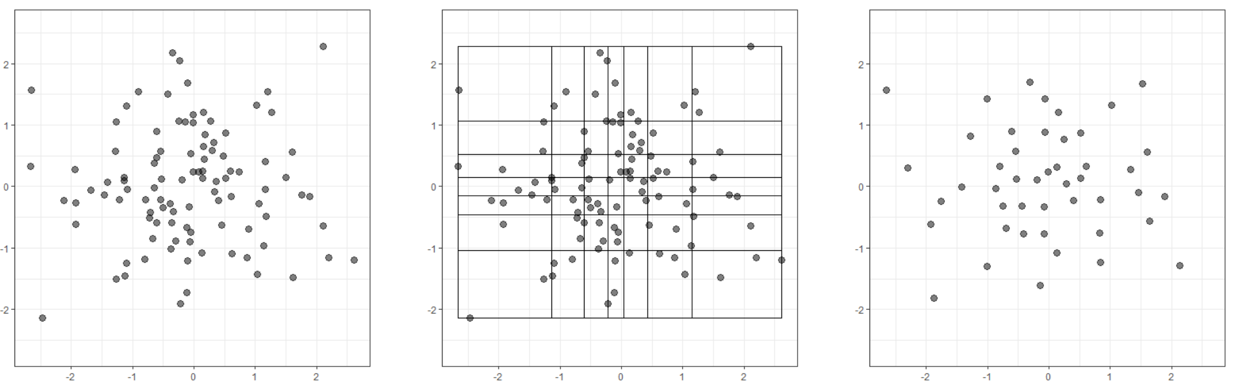


Figure 7.2: *An example of the equal frequency binning method reducing a bivariate normal configuration from 100 points to 46.*

Fixed Frequency

Under equal frequency discretization, the number of bins desired in the final discretization is the quantity chosen during the discretization process. Using the desired number of bins k, the number of points in each bin is computed as $\frac{n}{k}$. Instead of choosing k, consider instead choosing the number of points in each bin and then deriving k from it [123].

For instance, if it was decided that b observations per bin was desired, then that would result in there being $\frac{n}{b}$ bins, each containing b observations, again assuming $\frac{n}{b}$ is an integer. In the case where $\frac{n}{b}$ is not an integer, the number of bins is taken to be $\lfloor \frac{n}{b} \rfloor$, and the remaining points are again placed in bins according to any number of techniques as in equal frequency binning.

With the number of bins derived, cut points are computed as in equal frequency. Again the representative point is taken to be the by-dimension mean of each bin. Figure 7.3 illustrates the equal frequency binning algorithm applied to a bivariate normal point configuration.

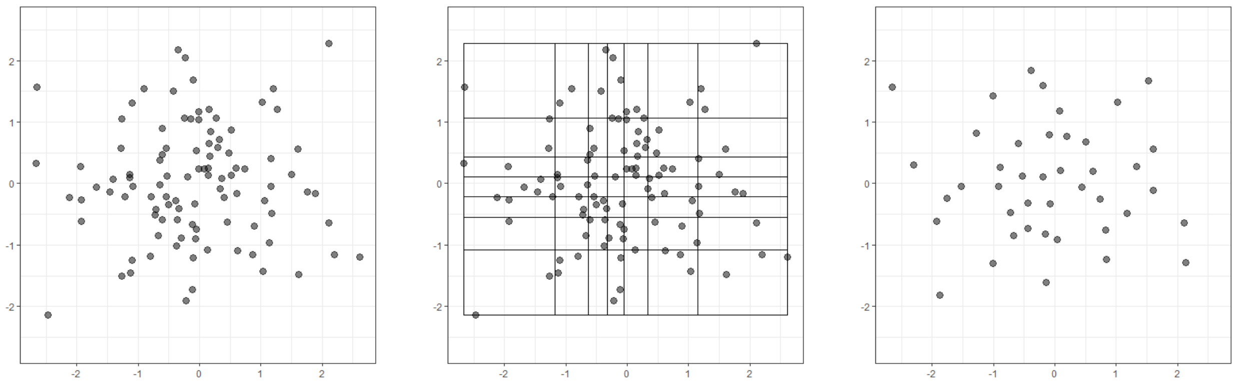


Figure 7.3: *An example of the fixed frequency binning method reducing a bivariate normal configuration from 100 points to 42.*

Hexagonal Binning

In two dimensions, each of equal width, equal frequency, and fixed frequency can be viewed as a rectangular tessellation of the plane. Instead of rectangles, hexagonal binning uses regular hexagons (where each side of the hexagon is the same size) to tessellate the plane [18].

The advantages of hexagonal binning over those methods that use rectangular bins is well documented. Using hexagonal bins instead of rectangular bins leads to a reduction in the bias of the density estimate performed on the binned configuration [96]. In addition, due to the staggering of the vertical and horizontal bin centers, [18] assert that the human visual system prefers hexagonal binning to other methods (such as equal width) as it de-emphasizes vertical and horizontal lines in the binned configuration. It should be noted that while hexagonal binning may be preferred to equal width binning in two dimensions, it is (marginally) more expensive to compute [18].

Figure 7.4 illustrates hexagonal binning on a bivariate normal point configuration. Of particular importance when considering the hexagonal binning algorithm is that, unlike the equal width, equal frequency, and fixed frequency algorithms, hexagonal binning does not generalize easily to dimension higher than two (e.g. different polytopes must to be used in higher dimensions and substantial algorithmic changes must be made to accommodate, for instance, three dimensional data).

There are of course higher dimensional analogues of hexagonal binning. The accuracy of the resulting quantization using various polytopes in terms of the mean squared error is discussed in [23]. The error measured results from replacing the x_i in the polytope with its centroid measured over a uniformly distributed set of points.

While discussion in [23] included a wide range of polytopes, for the purposes of binning, only space-filling polytopes (i.e. polytopes that tessellate the space) are of interest. Of all space-filling polytopes in two dimensions, the hexagonal lattice is shown to be optimal in terms of minimizing mean-squared error [23]. In three dimensions, the set of space-filling polytopes includes the cube, hexagonal prism, rhombic dodecahedron, and truncated octahedron. The analysis of [23] shows that the truncated octahedron has the lowest mean-squared error of these, and so could be used as the three dimensional analogue of the hexagon. Similarly in four dimensions, the 24-cell polytope is shown to be the optimal space-filling polytope.

Algorithms using lattice points (for which fast quantization algorithms already exist [22]) could be developed for higher dimensional analogues of hexagonal binning. In the experiments that will follow, however, hexagonal binning is limited to two dimensional space.

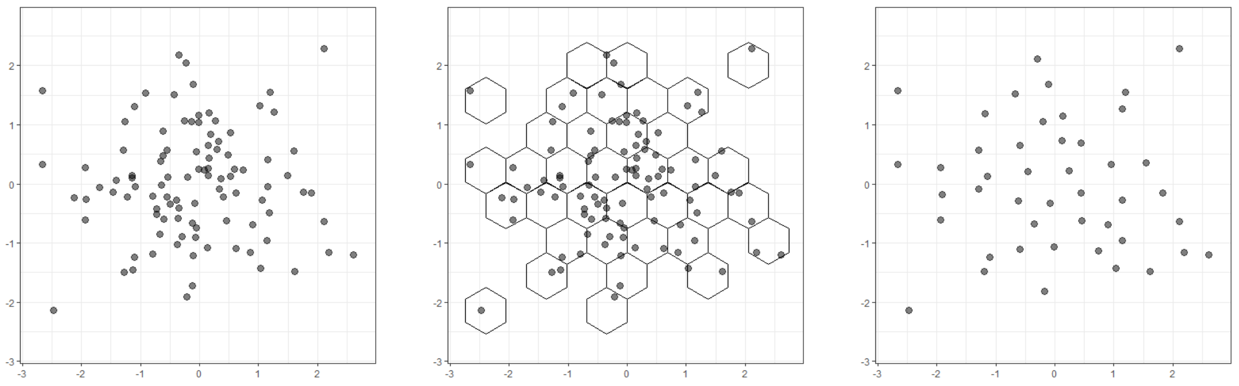


Figure 7.4: *An example of the hexagonal binning method reducing a bivariate normal configuration from 100 points to 45.*

Random Sampling

A final way to decrease the number of observations in a configuration is to simply perform simple random sampling on the configuration. Given a configuration with n observations, a random sample of size m is taken from the set $[1, .., n]$ without replacement, and with each element given an equal probability of being chosen. Denoting the (ordered) set of chosen observations $\mathbf{j}$, the reduced configuration is

$$\mathbf{x}_{binned} = [x_{(\mathbf{j}_1)}, ..., x_{(\mathbf{j}_m)}]$$

Unlike the previous algorithms, random sampling does not partition the domain of the data. Instead, it chooses sampled points that are themselves representative of the configuration (on average). Figure 7.5 illustrates the random sampling algorithm applied to a bivariate normal point configuration.

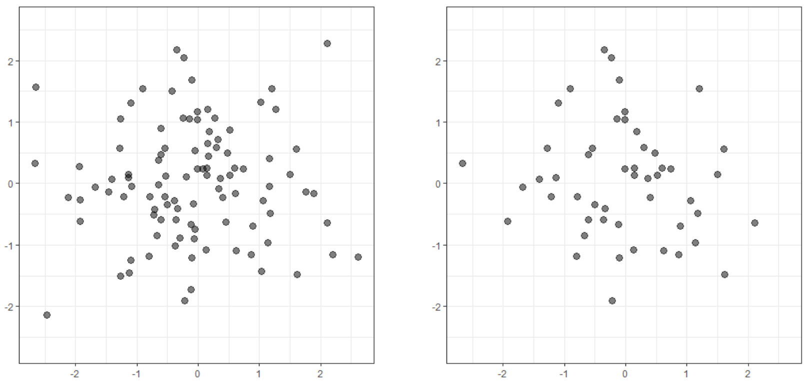


Figure 7.5: *An example of the random sampling method reducing a bivariate normal configuration from 100 points to 50.*

7.1.3 Issues with Existing Methods

There are two issues worth considering with the current unsupervised binning algorithms. First, consider Figures 7.1 - 7.5, where issues with the binned configurations are visible. Beginning with the equal width and hexagonal binning algorithms, the binned configurations lack areas of high density that were evident in the original. This is due to the identical tiles used in both areas of high and low density. In high density areas, many points will be placed in the same bin, eliminating relative density in the configuration. Also evident in the equal width algorithm is the presence of straight lines in the binned configuration. While this is lessened in the hexagonal binning configuration (as noted in [18]), the artificial structure imposed through binning is problematic in ensuring the binned configuration is representative of the original.

Issues also arise with the equal frequency, fixed frequency, and random sampling algorithms. Instead of eliminating density however, these algorithms seem to negatively impact the shape of the configuration, by placing points at the extremes of the configuration in the same bin. This leads to binned configurations that tend to retain the relative density of the configuration, at the expense of the overall shape.

The second issue with each of these algorithms (with the exception of random sampling) is their ability to generalize to higher dimensions. As was discussed, hexagonal binning is a specialized algorithm for two dimensions, and generalizing to higher dimensional space requires new algorithms to be developed. Algorithms like equal width and equal frequency do generalize in a straight-forward fashion (i.e. the algorithms are capable of binning in any dimensional space with minimal change), but the issue with these algorithms becomes one of computational feasibility. Consider placing $k-1$ cutpoints (i.e. k bins) in each dimension. Under this scheme, the number of bins that must be considered in a d dimensional point configuration is k^d. For a configuration of dimension ten, for example, placing only five bins in each dimension means there are $5^{10} = 9,765,625$ bins that must be considered. The number of bins in each dimension can of course be decreased, but this negatively effects the accuracy of the binned configuration. Overall, even in a relatively small number of dimensions with a small number of bins, methods such as equal width, equal frequency, and fixed frequency quickly become computationally infeasible.

These issues necessitate the need for a new algorithm that can not only be easily generalized to high dimensions, but also preserves more of the density and shape of the underlying configuration. This task will be undertaken in Section 7.2.

7.2 Tree-based Binning Framework

To motivate a new binning framework, first consider the results of classification and regression using tree-based models (commonly referred to as CART). For data of the form $\{\mathbf{y}; \mathbf{x}_1, ..., \mathbf{x}_p\}$, where $\mathbf{y}$ is a class (or response) variable, classification and regression trees are a supervised set of algorithms that seek to first partition the data into a set of rectangles before fitting a separate model in each. Figure 7.6 illustrates the general result of the CART partitioning step.

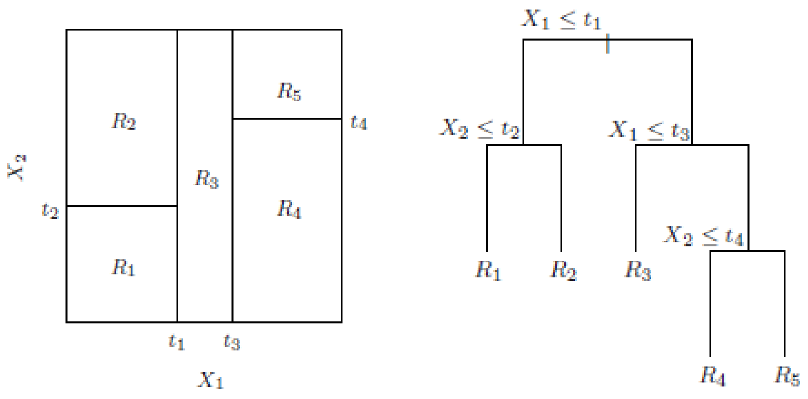


Figure 7.6: *The general idea of a classification and regression tree [40] which serves as the motivation for tree-based binning.*

The resulting partitioned feature space from classification and regression trees looks very similar to the binned partitions that were presented in Figures 7.1-7.3. The question that needs to be addressed then, is if CART methods can be adapted to fit into a non-supervised binning framework.

Consider how the partitions are created in classification and regression trees. As a supervised learning method, the optimal splitting point at each step is found by solving the optimization

$$\underset{j,s}{\arg\min}\left\{\underset{c_1}{\arg\min}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2\ +\ \underset{c_2}{\arg\min}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right\}$$

where $R_1(j,s) = \{\mathbf{x} \mid \mathbf{x}_j \leq s\}$, $R_2(j,s) = \{\mathbf{x} \mid \mathbf{x}_j > s\}$, j is the variable (or dimension) on which the split will occur, and s is the point (in dimension j) where the split will occur [40].

The main idea of CART is to find a dimension and two points between which a cut point will be placed. This same idea can perhaps be used in a non-supervised algorithm, where the optimal split is found using the vector of explanatory variables $\mathbf{x}$ instead of the vector of responses $\mathbf{y}$.

Beginning with all of the data in one bin, consider an algorithm that seeks to find partitions of the data subject to some criteria. In the absence of a class label (or response variate), an alternative way to determine where the split will occur is required. Consider computing a data-driven *bin score*, which will be used to choose the bin in which a cut point will be placed, for each bin using the $\mathbf{x}_i$ variables. Examples of possible bin scores include (but are not limited to)

- The observation count in the bin
- The largest difference between adjacent observations (i.e. $\max(x_{(i)} - x_{(i-1)}),\ i = 2, ..., n$)
- The range of the observations in the bin, $x_{(n)} - x_{(1)}$

Using the bin score (i.e. finding the maximum/minimum/some other measure), a bin is chosen in which a cut point will be placed. Once a bin has been chosen, a splitting function is defined to place a cutpoint. Like the bin score, the splitting function can be defined to meet any number of criteria, including

- The cutpoint is placed at the midpoint of the bin
- The cutpoint is placed between adjacent points with the largest difference
- The cutpoint is placed such that the two resulting bins contain the same number of points

These two steps are repeated until a stopping criteria is met. The stopping criteria can be defined in many different ways, including

- The binned configuration reaches a desired size
- The maximum density reaches a certain threshold
- The largest bin reaches a certain threshold

Once all of the bins have been defined, a representative point must be chosen or computed. Typically, this will be the mean of the points in the bin, but again could be computed in a number of ways.

These steps encompass the general framework of *tree-based binning*. Algorithm 4 illustrates the entire algorithm.

Algorithm 4 Tree-based Binning

```
Structures
    $\mathbf{X}$ - an $n \times p$ configuration matrix;
    $\mathbf{B}$ - A list containing all of the current bins
    nbins - the number of bins desired;

procedure TREE-BASED BINNING($\mathbf{X}$, nbins)
    $\mathbf{B} \leftarrow \mathbf{X}$                                        ▷ Initialize the original bin
    $cbins = 1$                                                               ▷ Current number of bins
    while !$stopCriteria(\mathbf{B})$ do
        $\mathbf{s} \leftarrow binScores(\mathbf{B})$                          ▷ Compute a score for each bin
        $i \leftarrow which.max(\mathbf{s})$                          ▷ Find the bin with the largest bin score
        $\mathbf{B}_{new} \leftarrow split(\mathbf{B}_i)$                     ▷ split the bin with the largest score
        $\mathbf{B} \leftarrow \{\mathbf{B}_{-i}, \mathbf{B}_{new}\}$  ▷ Combine the list of previous bins with the new bins
        $cbins \leftarrow cbins +$ number of new bins
    end while
    $\mathbf{Z} \leftarrow getPoints(\mathbf{B})$               ▷ Compute the representative point of each bin
    return $\mathbf{Z}$                                                 ▷ Return the binned configuration
end procedure
```

The function *binScores* takes each current bin, and computes some data-driven measure on it. Here, the bin with the highest measure will be chosen for splitting, although this need not necessarily be the case. The function *split* takes the bin with maximum bin score, and splits it into two (or more) bins. The *getPoints* function computes the representative point in each bin. The *stopCriteria* function decides if the algorithm should stop by seeing if the stopping criteria has been met. As discussed, each function is completely general, and can be tailored to the specific needs of a given data analysis. An example of such an algorithm is the *GapBin* algorithm, presented in Section 7.2.1.

7.2.1 GapBin

The framework for tree-based binning is completely general, and so can be tailored to meet the needs of any given data analysis. *GapBin* is but one possible proposal that fits in the tree-based binning framework. The main idea behind *GapBin* is to separate points when a large gap, viewed as a discontinuity in the data set, exists between them. This concept is somewhat inspired by the gap statistic [104].

Formally, define a data set of arbitrary size n and arbitrary dimension p. For a given bin

$i,\ i = 1, ..., m$, where m is the current number of bins, let n_i denote the number of data points in the bin, such that $\sum_{i=1}^{m} n_i = n$.

For bin i, define the sorted data in dimension k by $x_{i,k,(1)}, ..., x_{i,k,(n_i)}$. That is, each bin is defined by a vector containing individual entries of the form $x_{Bin,Dim,Entry}$ such that $x_{i,k,(1)} \leq x_{i,k,(2)} \leq .... \leq x_{i,k,(n_i)}$ for any k. Define the centre of bin i by the by-dimension mean of the data, $b_i = (\bar{x}_{i,1}, \bar{x}_{i,2}, ..., \bar{x}_{i,p})$, where $\bar{x}_{i,l} = \sum_{k=1}^{n_i} x_{i,l,k}$.

GapBin seeks to find the largest *gap* in a bin, that is, the largest difference between two sorted values in any dimension, which may be representative of a discontinuity in the data at that point. Define the maximum gap for bin i, dimension l as

$$g_{i,l} = max_j(x_{i,l,(j+1)} - x_{i,l,(j)})$$

where $j = 1, ..., n_i - 1$ and $x_{i,l,(j)}$ is the j^{th} largest value in bin i dimension l. Then, define the bin score for bin i as

$$\begin{aligned} s_i &= max_l \left(\frac{(g_{i,l})(n_i)}{x_{\cdot,l,(n)} - x_{\cdot,l,(1)}} \right) \\ &= max_l (g_{i,l} f_l) \end{aligned} \tag{7.1}$$

where $l = 1, ..., p$, and $f_l = \frac{n}{x_{\cdot,l,(n)} - x_{\cdot,l,(1)}}$. By using a score such as this, *GapBin* tends to favour splitting bins with a large gap relative to the spread of the points (which helps to retain the shape of the configuration), or bins with large counts, ensuring that binning occurs in more dense areas of the configuration, preserving the density of the configuration. Binning continues until the configuration reaches the desired size. The *GapBin* algorithm is presented Algorithm 5.

Figure 7.7 illustrates the GapBin algorithm applied to a simple bivariate normal configuration. What is interesting to note about the GapBin algorithm is that, like the equal width and hexagonal binning algorithms, the general shape of the bivariate normal configuration seems to be preserved - the majority of the outlying points in the original configuration are in the binned configuration as well. This is in contrast to both the equal frequency and fixed frequency algorithms, which instead chooses to place some of the outlying points in the same bin, distorting the overall shape.

Also of interest is the seemingly retained relative density of the binned configuration - high density areas in the original configuration still appear to be high density areas in the binned configuration (relatively speaking). This retention of density is also shown in configurations produced by equal frequency, fixed frequency, and random sampling, while equal width and hexagonal binning tend to make the overall density of the configuration very uniform.

A significant advantage that *GapBin* offers over competing algorithms is in the number of points in the reduced configuration that can be achieved. For each of the other algorithms consid-

Algorithm 5 GapBin

```
Structures
    X - an $n \times p$ matrix containing the points identified by $x_{i,l,j}$;
    cbins = the current number of bins;

procedure gapScores(X, cbins)
    $g_{i,l} \leftarrow \max_j(x_{i,l,(j+1)} - x_{i,l,(j)}) \quad \forall \quad l = 1,...,p, \quad i = 1,...,cbins$
    $s_i \leftarrow \max_l(\frac{g_{i,l}\, n_i}{x_{\cdot,l,(n)} - x_{\cdot,l,(1)}}), \quad i = 1,...,cbins$
    $\mathbf{S} \leftarrow \{s_1,...,s_{cbins}\}$
    return S
end procedure

procedure gapSplit(X, S, cbins)
    for $i^* \in \arg\max\{i : s_i\}$ do
        $l^* = \arg\max\{l : \max_l(\frac{g_{i^*,l}\, n_{i^*}}{x_{i^*,\cdot,(n)} - x_{i^*,\cdot,(1)}})\}$
        $j^* = \arg\max\{j : \max_j(x_{i^*,l^*,(j+1)} - x_{i^*,l^*,(j)})\}$
        for $k > j^*$ do
            $x_{i^*+1,\cdot,(k-j^*)} \leftarrow x_{i^*,\cdot,(k)}$      ▷ Create a new bin by splitting bin $i^*$ for ALL dimensions
            $x_{i^*,\cdot,(k)} \leftarrow \varnothing$      ▷ Remove this element from bin $i^*$
        end for
        $cbins \leftarrow cbins + 1$
    end for
    return X      ▷ Return the newly formed bin list
end procedure
```

ered (other than random sampling), the exact number of points in the binned configuration cannot be directly controlled. Only the number of bins in each dimension can be directly controlled, so only an approximate number of points can be achieved. Conversely, in *GapBin* an additional split (creating one additional point in the binned configuration) is always possible to perform. As such, the number of points is controlled directly, allowing for a more accurate binned configuration.

A second advantage offered by *GapBin*, and tree-based binning in general, is the ease with which it generalizes to higher dimensional space. Since it seeks to place a partition in only the dimension(s) with the largest relative gaps, it is quite possible some dimensions will not be binned at all. This decreases the total number of bins that must be considered, significantly decreasing the run time of the algorithm. This is in contrast to algorithms such as equal width or equal frequency, where each dimension is binned, regardless of the spread of the data.

7.3 Additional Tree-based Binning Algorithms

In addition to the *GapBin* algorithm, the tree-based binning framework encompasses a wide variety of binning algorithms, including equal width binning. Also, any clustering algorithm that works in one dimension falls into the tree-based binning framework. For example, l1-fusion clustering algorithm proposed in [56] (and studied further in [88]) can be modified to work on one

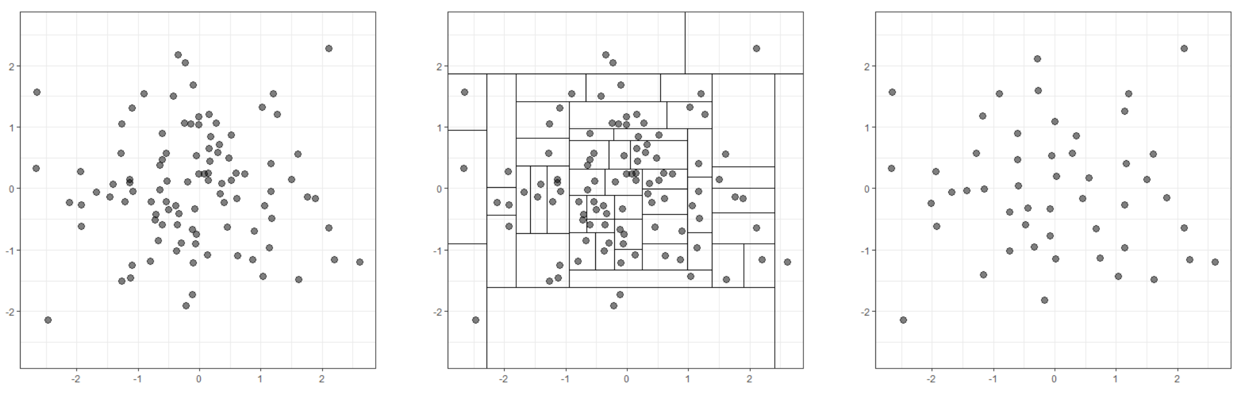


Figure 7.7: *An example of the GapBin method reducing a bivariate normal configuration from 100 points to 50.*

dimension at a time, and can thus be used as a tree-based binning technique. In a one-dimension, the algorithm seeks to minimize the criterion in Equation (7.2).

$$\underset{\alpha_1,\ldots,\alpha_n}{\arg\min} \sum_{i=1}^{n} ||\mathbf{x}_i - \alpha_i||_2^2 \; + \; \lambda \sum_{1\leq i<j\leq n} |\alpha_i - \alpha_j| \tag{7.2}$$

for some penalty parameter $0 \leq \lambda \leq \infty$. Instead of choosing a value of λ, an algorithm is proposed where each successive iteration is equivalent to increasing λ [88]. The pseudo-code for the algorithm is shown in Algorithm 6. Algorithm 7 illustrates how this can be cast in the tree-based binning framework.

Algorithm 6 Clustered Splitting Algorithm

```
Structures
    $\mathcal{C}$ - A cluster of points;
procedure CLUSTEREDSPLIT($\mathbf{x}_n$)
    Sort data in ascending order and store them as $\mathbf{x}_n = \{x_1, ..., x_n\}$
    Set the current partition of $\mathbf{x}_n$ to $\mathbf{x}_n$
    while Any cluster contains more than one point do
        Select one cluster $\mathcal{C}$, with $|\mathcal{C}| > 1$, from a current cluster partition of $\mathbf{x}_n$
        Find a split partition $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ that maximizes the distance $\bar{x}_{\mathcal{C}_2} - \bar{x}_{\mathcal{C}_1}$
        Store the split $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ and the corresponding value $\lambda = \frac{\bar{x}_{\mathcal{C}_2} - \bar{x}_{\mathcal{C}_1}}{|\mathcal{C}|}$
        Replace $\mathcal{C}$ with $\mathcal{C}_1 \cup \mathcal{C}_2$ in the current partition of $\mathbf{x}_n$
    end while
end procedure
```

While the l1-clustering algorithm does fit into the tree-based binning framework, it was not designed specifically for use in the context of binning. For this reason, only the *GapBin* algorithm,

Algorithm 7 Clustered Binning

```
Structures
    X - an $n \times p$ matrix containing the points identified by $x_{i,l,j}$;
    cbins - the current number of bins;

procedure clusterScores(X, cbins)
    $g_{i,l} \leftarrow \max_j(\frac{1}{n-j}\sum_{k=j+1}^{n} x_{i,l,(k)} - \frac{1}{j}\sum_{k=1}^{j} x_{i,l,(k)}) \quad \forall \quad l = 1, ..., p, \quad i = 1, ..., cbins$
    $s_i \leftarrow \max_l(g_{i,l}), \quad i = 1, ..., cbins$
    $\mathbf{S} \leftarrow \{s_1, ..., s_{cbins}\}$
    return S
end procedure

procedure clusterSplit(X, S, cbins)
    for $i^* \in \arg\max\{i : s_i\}$ do
        $l^* = \arg\max\{l : \max_l(g_{i^*,l})\}$
        $j^* = \arg\max\{j : \max_j(\frac{1}{n-j}\sum_{k=j+1}^{n} x_{i^*,l^*,(k)} - \frac{1}{j}\sum_{k=1}^{j} x_{i^*,l^*,(k)})\}$
        for $k > j^*$ do
            $x_{i^*+1,\cdot,(k-j^*)} \leftarrow x_{i^*,\cdot,(k)}$        ▷ Create a new bin by splitting bin $i^*$ for ALL dimensions
            $x_{i^*,\cdot,(k)} \leftarrow \varnothing$        ▷ Remove this element from bin $i^*$
        end for
        $cbins \leftarrow cbins + 1$
    end for
    return X        ▷ Return the newly formed bin list
end procedure
```

which has been explicitly designed for use in the context of tree-based binning, will be used in future experiments.

7.4 Concluding Remarks

The tree-based binning algorithm represents a completely general framework from which a large number of specific binning algorithms could be created, and tailored specifically to the type of analysis being undertaken. One such algorithm, *GapBin*, has been proposed with the aim of simultaneously retaining the shape and density of the original configuration during binning, while also generalizing easily to high dimensional space.

For any of these claims to be validated, the general tree-based binning algorithm must first be implemented as a software package. Chapter 8 details the implementation of tree-based binning in R. Using this implementation, the claims of *GapBin* are tested in a series of experiments in Chapter 9.

Chapter 8

treebinr - An R Package for Binning

The tree-based binning framework presented in Chapter 7 is implemented in R in the package *treebinr*. The implementation is completely general - a prospective user can create their own tree-based binning algorithm, or choose from pre-built options such as *GapBin* or the clustered binning algorithm discussed in Algorithm 7.

8.1 The General Structure of treebinr

A fully specified tree-based binning algorithm requires the following functions to be defined

1. A method by which a score is computed for each bin.
2. A method by which a bin is chosen.
3. A method by which a chosen bin is split into multiple bins.
4. A method by which a stopping criteria is computed.
5. A method by which a representative point is computed for each bin to create a binned configuration.

In addition, if a user wishes to add out-of-sample points to an already binned configuration, a method for determining which bin a point would fall in is also required.

8.1.1 The Main Function - treebin

The `treebinr` package was created to be completely customizable, a prospective user could customize any tree-based binning algorithm to fit the needs of their analysis. For instance, if a prospective user wanted the representative point of each bin to be the by-dimension median of the points instead of the by-dimension mean, that would be within their control. This is accomplished by passing custom (or built-in) options to the main function `treebin`. The input variables required for `treebin` are

`X`	The point configuration to be binned.
`stopCriteria`	A function to compute the stopping criteria for the algorithm.
`binMeasure`	A function to compute the measure associated with each bin.
`boundaryTest`	A function to test if a given point is contained in a given bin.
`makePoint`	A function to turn the contents of a bin into a single point.
`selectBin`	A function for choosing between bins to be split.
`splitBin`	A function for splitting a bin.
`binInfo`	Additional information to be supplied to the first bin.
`inputs`	A list containing additional input parameters required by user supplied functions.

The default algorithm (and therefore the default input for each of the variables above) is the *GapBin* algorithm. The specific requirements for each of the input variables above are discussed in Sections 8.1.3 - 8.1.8.

The function `treebin` returns an (S4) object of class *treebinr*, which contains the following slots

`points`	An $m \times d$ matrix containing the representative points of the binned configuration.
`counts`	A vector of length m containing the number of points from the original configuration in each bin.
`bins`	Information about each bin in the configuration, returned as an m-element list containing objects of class `bin`.
`tree`	A directed graph matrix representing the binning tree underlying the algorithm.

Each item in the `bins` list is an S4 object of class `bin`, which contains the slots

`boundary`	The defined boundary of the bin. The only requirement here is that this slot is compatible with the user defined function to test the inclusion of out-of-sample points.
`contents`	An $m \times p$ matrix containing the points in the bin.
`measure`	The bin score associated with this bin.
`index`	An internal tracker indicating where the bin is located in the binning tree.
`info`	Any additional information that should be associated with the bin.

The `tree` output variable is a directed graph object which maps the path of the binning tree. This allows the algorithm (or user) to lookup which bins share a common parent (i.e. were created as part of the same split). For example, a `tree` such as that below would indicate that bin 1 is the parent node of bins 2 and 3. Further, bin 2 and 3 are not the parent nodes to any other bin.

$$tree = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Figure 8.1: *An example of the structure of a simple tree from the tree-based binning framework. Here, node 1 is the parent to nodes 2 and 3. Nodes 2 and 3 are not parent nodes to any other bin.*

8.1.2 stopCriteria

The `stopCriteria` function checks if the current set of bins meets the stopping criteria for the algorithm. In the case of *GapBin*, the stopping criteria is simply the number of bins in the current iteration being larger than some desired amount. The function takes as input the following variables

`bins`	A list containing objects of class `bin`.
`inputs`	A named list-object containing additional input parameters.

and returns `TRUE` if the stopping criteria has been met, and `FALSE` otherwise. As an example, the `stopCriteria` function for *GapBin* is presented below.

```
gapStop <- function(bins, inputs){
  n <- length(bins)
  return((n >= inputs$numbins))
}
```

8.1.3 binMeasure

The `binMeasure` function computes the score associated with each bin, from which a bin will later be chosen for splitting. In the case of *GapBin*, the `binMeasure` function computes s_i from Equation (7.1). The function takes as input the following variables

`bin`	An object of class `bin`.
`inputs`	A named list containing additional input parameters.

and returns a `score` compatible with the `selectBin` function. This could be a single number, a vector, a matrix, or any other output required by `selectBin`. The only requirement is that the output be compatible with the `selectBin` function. As an example, the `binMeasure` function for *GapBin* is presented below.

```
gapMeasure <- function(bin, inputs){

  if(nrow(bin@contents) == 1){
    return(matrix(NA, 1, ncol(bin@contents)))
  }

  #For Each Dimension, compute the gap statistic
  gaps <- apply(bin@contents, 2, getGap)

  #Scale the gaps by the number of points in the bin and by tau
  n <- nrow(bin@contents)
  tau <- inputs$tau

  gaps <- gaps * n^tau

  #Scale the gaps by the dimension range
  dimRange <- matrix(inputs$dimRange,
                     nrow=(n-1),
                     ncol=ncol(bin@contents),
                     byrow=TRUE)

  gaps <- gaps/dimRange

  #Any dimension that has no variability should not be split.
  gaps[is.nan(gaps)] <- -Inf

  return(gaps)
}

getGap <- function(binDim){

  sortedBin <- sort(binDim)
  axis <- sortedBin[-length(sortedBin)]
  axis2 <- sortedBin[-1]

  gaps <- abs(axis - axis2)

  return(gaps)
}
```

8.1.4 boundaryTest

The `boundaryTest` function is designed to test if a point belongs to a given bin. This is to support out-of-sample additions to the configuration after binning is complete. `boundaryTest` takes the following input variables

`point`	A vector containing the point to be tested.
`bin`	An object of class `bin`.
`inputs`	A named list containing additional input parameters.

and returns a logical variable indicating if the point belongs to the particular bin or not. For example, if a rectangular bin is defined by the points {(-1,-1), (-1,1), (1,1), (1,-1)}, the tested point (0,0) would return `TRUE`, while the tested point (2,2) would return `FALSE`. As an example, the *GapBin* `boundaryTest` function is shown below.

```
gapBoundaryTest <- function(point, bin, inputs){

  point <- matrix(point,nrow=1)
  boundary <- bin@info$binRange
  indicator <- c()

  for(i in 1:ncol(point)){
    indicator <- c(indicator,
                  (point[i] < boundary[2,i] & point[i] > boundary[1,i]))
  }

  if(all(indicator)){
    return(TRUE)
  }else{
    return(FALSE)
  }
}
```

8.1.5 makePoint

The `makePoint` function computes the representative point of each bin. That is, given a bin with m points, `makePoint` computes a single point that represents (and replaces) these points in the binned configuration. It takes as input the following variables

`bin`	An object of class `bin`.
`inputs`	A named list-object containing additional input parameters.

The function must return a vector containing the representative point of the bin. As an example, the `makePoint` function for *GapBin* is shown below.

```
gapPoints <- function(bin, inputs){
  point <- apply(bin@contents, 2, mean)
  return(point)
}
```

8.1.6 selectBin

The `selectBin` function evaluates the scores computed by the `binMeasures` function, and chooses a bin to be passed to the `splitBin` function, where it will be split into two (or more) new bins. The `selectBin` function takes the following input variables

`bins`	A list containing `bin` objects.
`inputs`	A named list-object containing additional input parameters.

and must return the corresponding index of the bin to be split. For example, suppose four bins are passed to the `selectBin` function with scores of 2.7, 3.4, 0, and 9.8 respectively. If the `selectBin` function is designed to identify the largest such score, then the function would return the index of the largest score, which is bin 4 in this case. As an example, the `selectBin` function for *GapBin* is shown below.

```
gapSelect <- function(bins, inputs){

  #Find the maximum gap in each bin
  findMax <- function(bin){
    max <- max(bin@measure)
    return(max)
  }

  gapMax <- sapply(bins, findMax)

  #Find the bin with the largest of all gaps
  binIndex <- which.max(gapMax)

  return(binIndex)
}
```

8.1.7 splitBin

The `splitBin` function takes the bin chosen by `selectBin`, and splits it into two (or more) new bins. It takes as input the following

`bin`	An object of class `bin`.
`binMeasure`	A function to compute the measure associated with each bin.
`inputs`	A named list containing additional input parameters.

The function takes the bin chosen by `selectBin`, and creates new `bin` objects from it. In the case of *GapBin*, the chosen bin is split at the two points that resulted in the largest bin score. For example, suppose points (j) and $(j+1)$ in dimension k of bin i resulted in the largest bin score. Then, the `splitBin` function creates two new `bin` objects, one containing $x_{i,k,(1)},...,x_{i,k,(j)}$ and the other containing $x_{i,k,(j+1)},...,x_{i,k,(n)}$. As an example, the `splitBin` function for *GapBin* is shown below.

```
gapSplit <- function(bin, binMeasure, inputs){

  #Extract the bin information
  currentMeasure <- bin@measure
  currentBin <- bin@contents
  currentBoundary <- bin@info$binRange

  #Find the maximum measure in each dimension,
  #Take the overall maximum as the dimension to be split
  whichAxis <- which.max(apply(currentMeasure,2,max))

  #sort the bin along the chosen axis
  sortedBin <- currentBin[order(currentBin[,whichAxis]),,drop=FALSE]

  #Split the sorted bin at the maximum gap
  gapIndex <- which.max(currentMeasure[,whichAxis])

  leftContents <- sortedBin[seq(1, gapIndex, 1),,drop=FALSE]
  rightContents <- sortedBin[seq(gapIndex + 1, nrow(currentBin), 1),,drop=FALSE]

  #Define the new bin boundaries
  newBoundary <- (max(leftContents[,whichAxis]) + min(rightContents[,whichAxis]))/2

  leftBoundary <- currentBoundary
  leftBoundary[2,whichAxis] <- newBoundary

  rightBoundary <- currentBoundary
  rightBoundary[1,whichAxis] <- newBoundary

  #Put together the new bins in a binfo object
  leftBin <- bin(boundary = bin@boundary,
                 contents = leftContents,
                 measure = NULL,
                 info = list(binRange = leftBoundary))

  rightBin <- bin(boundary = bin@boundary,
                  contents = rightContents,
                  measure = NULL,
                  info = list(binRange = rightBoundary))

  #Compute the new bin measures
  leftBin@measure <- binMeasure(leftBin, inputs)
  rightBin@measure <- binMeasure(rightBin, inputs)

  #Return the new bins in a list
  return(list(leftBin, rightBin))
}
```

8.1.8 binInfo and inputs

Since `treebin` aims to be a completely general function, it may require additional input variables supplied by the user. This is facilitated using two input variables - `binInfo` and `inputs`.

The `binInfo` variable is used to store information specific to each bin, for use in any of the functions considered above. For instance, the *GapBin* implementation requires the bin boundaries for each bin. The bin boundaries are used to check for bin inclusion in `gapBoundaryTest`. `binInfo` is used to pass the initial bin dimension to the initial bin object created in `treebin`, and is updated as new bins are created.

Other input variables may be required for certain computations within `treebin` that do not depend on individual bins (i.e. the variable does not change from bin to bin). To pass these types of variables into `treebin`, the named-list `inputs` variable is used. In *GapBin* for instance, the stopping criterion is based on achieving a certain number of bins in the configuration. This information is passed to `treebin` through the `inputs` variable.

8.2 Other Built-in Algorithms

GapBin is not the only algorithm that fits into the tree-based binning framework. Both the 1-dimensional clustering algorithm discussed in Section 7.3 [56], as well as equal width binning [121] fall into the tree-based binning framework.

8.2.1 Clustered Binning

The first task to incorporating the 1-dimensional clustering algorithm into the tree-based binning framework is to define how individual bins will be scored. For that, consider the criteria by which a partition was created in Algorithm 6. A cluster point was placed at the point that maximizes $\bar{X}_{\mathcal{C}_2} - \bar{X}_{\mathcal{C}_1}$. This value was computed for every dimension in a given bin, and the overall maximum is taken as the score for that bin. This is encapsulated in the `clusterMeasure` algorithm.

```
clusterMeasure <- function(bin, inputs){
  #Find the cluster that results in the biggest difference in cluster mean

  if(nrow(bin@contents) == 1){
    measure <- 0
    return(measure)
  }

  maxDiff <- apply(bin@contents,2,getBiggestClusterDiff)

  return(maxDiff)
}

getBiggestClusterDiff <- function(bin){
  bin <- sort(bin)
  n <- length(bin)

  C1 <- bin[1]
  C2 <- bin[2:n]

  meanC1 <- mean(C1)
  meanC2 <- mean(C2)

  maxDiff <- meanC2 - meanC1

  for(i in 2:(n-1)){
    C1 <- bin[1:i]
    C2 <- bin[(i+1):n]

    meanC1 <- mean(C1)
    meanC2 <- mean(C2)

    maxDiff <- c(maxDiff, (meanC2-meanC1))
  }

  return(maxDiff)
}
```

Since the goal is to maximize this difference, choosing the next bin in the sequence to be split is done by choosing the bin with the largest score.

```
clusterSelect <- function(bins, inputs){
  #Find the maximum gap in each bin
  gapMax <- sapply(bins, findMax)

  #Find the bin with the largest of all gaps
  binIndex <- which.max(gapMax)

  return(binIndex)
}

 findMax <- function(bin){
    max <- max(bin@measure)
    return(max)
  }
```

Once a bin is selected, splitting it into two bins is straightforward. Computing the bin score required the points to be separated into two clusters, so all that remains in splitting the bin is to place a partition at the halfway point between the clusters, creating two new bins containing the respective points in each cluster.

```
clusterSplit <- function(bin, binMeasure, inputs){

  #Extract the bin information
  currentMeasure <- bin@measure
  currentBin <- bin@contents
  currentBoundary <- bin@info$binRange
  #Find the maximum measure in each dimension
  #take the overall maximum as the dimension to be split
  whichAxis <- which.max(apply(currentMeasure,2,max))

  #sort the bin along the chosen axis
  sortedBin <- currentBin[order(currentBin[,whichAxis]),,drop=FALSE]

  #Split the sorted bin at the maximum gap
  gapIndex <- which.max(currentMeasure[,whichAxis])

  leftContents <- sortedBin[seq(1, gapIndex, 1),,drop=FALSE]
  rightContents <- sortedBin[seq(gapIndex + 1, nrow(currentBin), 1),,drop=FALSE]

  #Define the new bin boundaries
  newBoundary <- (max(leftContents[,whichAxis]) + min(rightContents[,whichAxis]))/2

  leftBoundary <- currentBoundary
  leftBoundary[2,whichAxis] <- newBoundary

  rightBoundary <- currentBoundary
  rightBoundary[1,whichAxis] <- newBoundary

  #Put together the new bins in a binfo object
  leftBin <- bin(boundary = bin@boundary,
                 contents = leftContents,
                 measure = NULL,
                 info = list(binRange = leftBoundary))

  rightBin <- bin(boundary = bin@boundary,
                  contents = rightContents,
                  measure = NULL,
                  info = list(binRange = rightBoundary))

  #Compute the new bin measures
  leftBin@measure <- binMeasure(leftBin, inputs)
  rightBin@measure <- binMeasure(rightBin, inputs)

  #Return the new bins in a list
  return(list(leftBin, rightBin))
}
```

Once binning is complete, representative points are taken to be the by-dimension means of each bin

```
clusterPoints <- function(bin, inputs){
  point <- apply(bin@contents, 2, mean)
  return(point)
}
```

The algorithm is stopped by achieving a certain number of bins in the configuration, just as in *GapBin*.

```
clusterStop <- function(bins, inputs){
  n <- length(bins)
  return((n >= inputs$numbins))
}
```

Finally, like *GapBin*, the resulting bins are rectangular. As such, performing a test of inclusion for an out-of-sample point is as simple as comparing the position of the point relative to the boundaries of the bin under consideration.

```
clusterBoundaryTest <- function(point, bin, inputs){

  point <- matrix(point,nrow=1)
  boundary <- bin@info$binRange
  indicator <- c()

  for(i in 1:ncol(point)){
    indicator <- c(indicator,
                   (point[i] < boundary[2,i] & point[i] > boundary[1,i]))
  }

  if(all(indicator)){
    return(TRUE)
  }else{
    return(FALSE)
  }
}
```

Examples of the clustered binning algorithm will be considered in Section 8.3.

8.2.2 Equal Width Binning

The equal width binning algorithm can also be fit into the tree-based binning framework. First, an arbitrary dimension of the data is chosen, and is split into b bins of equal size, where b is a user supplied variable indicating how many bins should be created in each dimension. This is executed by the `ewSplit` function

```
ewSplit <- function(bin, binMeasure, inputs){

  #Extract the bin information
  currentMeasure <- bin@measure
  currentBin <- bin@contents
  currentBoundary <- bin@info$binRange

  #Find the maximum measure
  whichAxis <- which.max(currentMeasure)

  #sort the bin along the chosen axis
  sortedBin <- currentBin[order(currentBin[,whichAxis]),]

  #Split bin
  splits <- seq(inputs$oRange[1,whichAxis],
                inputs$oRange[2,whichAxis],
                length.out = inputs$binsperdim+1)

  #make new bins
  newBins <- list()

  for(i in 2:(inputs$binsperdim+1)){

    #Define new boundary
    newBoundary <- currentBoundary
    newBoundary[1,whichAxis] <- splits[i-1]
    newBoundary[2,whichAxis] <- splits[i]

    #Find the points below the split
    index <- which(currentBin[,whichAxis] <= splits[i])

    if(length(index) > 0){
      newBin <- bin(boundary = bin@boundary,
                    contents = currentBin[index,,drop=FALSE],
                    info=list(binRange = newBoundary))

      newBin@measure <- ewMeasure(newBin, inputs)
      currentBin <- currentBin[-index,,drop=FALSE]
    }else{
      newBin <- bin(boundary = bin@boundary,
                    contents = matrix(,0,ncol(currentBin)),
                    info=list(binRange = newBoundary))
      newBin@measure <- ewMeasure(newBin, inputs)
    }

    newBins <- unlist(list(newBins, newBin), recursive=FALSE)
  }
  return(newBins)
}
```

Each of these b bins are then chosen in sequence, and one of the remaining $d-1$ dimensions that were not previously binned are chosen arbitrarily (where d is the dimension of the original configuration), and is split using `ewSplit`. The way in which a dimension is chosen for binning is quite simple - the range of each dimension in the bin is compared to the range of the original configuration as a ratio. Any dimension that has not been binned will have a ratio of 1, while any dimension that has already been binned will have a ratio of $\frac{1}{b}$ - these ratios are computed as the bin score for each bin, using `ewMeasure`.

```
ewMeasure <- function(bin, inputs){

  #The measure is the bin with largest range (as a percentage of original range)
  unscaledMeasure <- bin@info$binRange[2,] - bin@info$binRange[1,]

  #Scale by original range
  oRange <- inputs$oRange[2,] - inputs$oRange[1,]
  scaledMeasure <- unscaledMeasure/oRange

  return(scaledMeasure)
}
```

The other functions which choose the bin to be split, compute the representative point, compute the stopping criterion, and test for the inclusion of out-of-sample points are all identical to *GapBin*. Binning continues until every bin has a bin score of $\frac{1}{b}$. Examples of equal width binning in the tree-based binning framework are shown in Section 8.3.

8.3 A Simple Example

As a simple example of each of the three built in algorithms available in `treebinr`, consider the bivariate normal point cloud in Figure 8.2. Using each of the three algorithms, consider decreasing the size of the configuration from one thousand points to five hundred (or as close as possible). The result of each algorithm is shown with the original configuration in Figure 8.2.

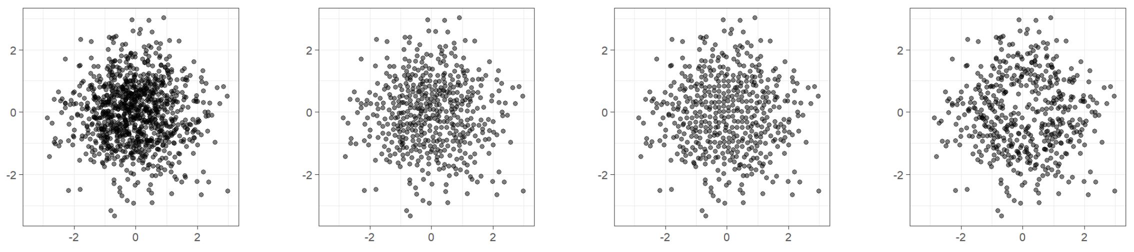
```
library(treebinr)

set.seed(23456)

norm.original <- matrix(rnorm(2000,0,1),ncol=2)
oRange <- sapply(1:2, FUN = function(j){range(norm.original[,j])})
dimRange <- sapply(1:ncol(X), FUN = function(j){diff(range(X[,j]))})
binRange <- binRange = sapply(1:2, FUN = function(j){range(norm.original[,j])}))

norm.gap <- treebin(norm.original,
                    inputs = list(tau = 1,
                                  dimRange = dimRange,
                                  numbins = 500)
norm.ew <- treebin(norm.original,
                   stopCriteria = ewStop,
                   binMeasure = ewMeasure,
                   selectBin = ewSelect,
                   splitBin = ewSplit,
                   boundaryTest = ewBoundaryTest,
                   makePoint = ewPoints,
                   inputs = list(oRange = oRange,
                                 binsperdim = 39,
                                 numbins = 1521),
                   binInfo = list(binRange = binRange))

norm.cluster <- treebin(norm.original,
                        stopCriteria = clusterStop,
                        binMeasure = clusterMeasure,
                        selectBin = clusterSelect,
                        splitBin = clusterSplit,
                        boundaryTest = clusterBoundaryTest,
                        makePoint = clusterPoints,
                        inputs = list(oRange = oRange,
                                      numbins = 500),
                        binInfo = list(binRange = binRange))
```



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

Figure 8.2: *Binning a (a) normal point configuration using each of (b) GapBin, (c) Equal Width Binning, (d) and Clustered Binning. All are done in the framework of tree-based binning.*

8.4 Additional Functionality

In addition to the ability to decrease the size of a configuration, there is other functionality available in *treebinr* that takes advantage of the tree structure of the resulting binned configuration. Once a configuration is binned, it can be modified in a number of ways by passing the `treebinr` object to a number of built in function. Each of the following modifications can be executed

1. Out of sample point(s) can be added to the configuration.
2. The next split in the sequence can be performed (as if the user asked for $b + 1$ bins instead of b bins).
3. The last split in the sequence can be undone (as if the user asked for $b - 1$ bins instead of b bins).
4. Any bin in the configuration can be chosen to be split.
5. Any bin in the configuration can be chosen to be pruned.

To explore the additional functionality available in *treebinr*, consider binning a point configuration from size 100 to 50.

```
library(treebinr)

set.seed(23456)
norm.original <- matrix(rnorm(200,0,1),ncol=2)

dimRange <- sapply(1:ncol(norm.original),
                   FUN = function(j){diff(range(norm.original[,j]))})
```

```
norm.gap <- treebin(norm.original, inputs = list(tau = 1,
                                            dimRange =dimRange,
                                            numbins = 50))
```

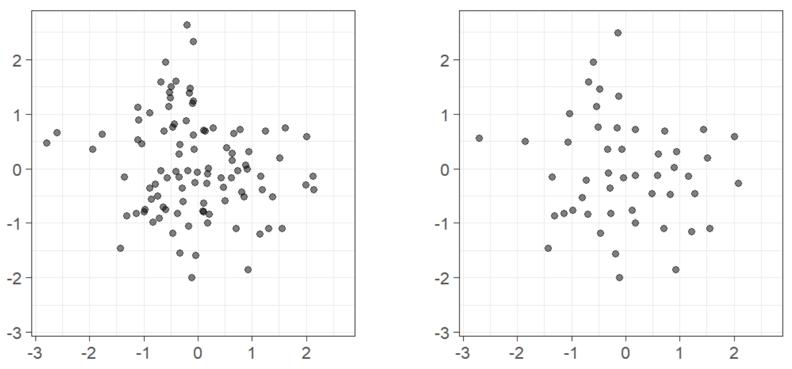


Figure 8.3: *Binning a bivariate normal point configuration from 100 points to 50 using GapBin.*

8.4.1 Add a Point

Once a configuration has been binned, a user may want to add out of sample points to the configuration. For instance, as new data is collected, a user may simply want to add it to the already binned configuration, without re-binning. For this purpose, the `addPoint` function is available. This function finds the bin to which the out-of-sample point belongs, and recomputes the representative point of that bin. `addPoint` takes as input

`point`	A vector containing the point to be added.
`treebinr.obj`	An object of class `treebinr`.
`binMeasure`	A function to compute the measure associated with each bin.
`makePoint`	A function to turn the contents of a bin into a single point.
`inputs`	A list containing additional input parameters required by user supplied functions.

and returns the modified `treebinr` object with the newly added point. As an example, consider adding a point at (-2,-2) to the bivariate normal configuration binned previously. The affected point in the binned configuration is highlighted in red.

```
dimRange <- sapply(1:2, FUN = function(j) {range(norm.original[,j])})
norm.gap2 <- addPoint(point = c(-2,-2),
                      treebinr.obj = norm.gap,
                      binMeasure = gapMeasure,
                      makePoint = gapPoints,
                      inputs = list(dimRange = dimRange, tau=1))
```

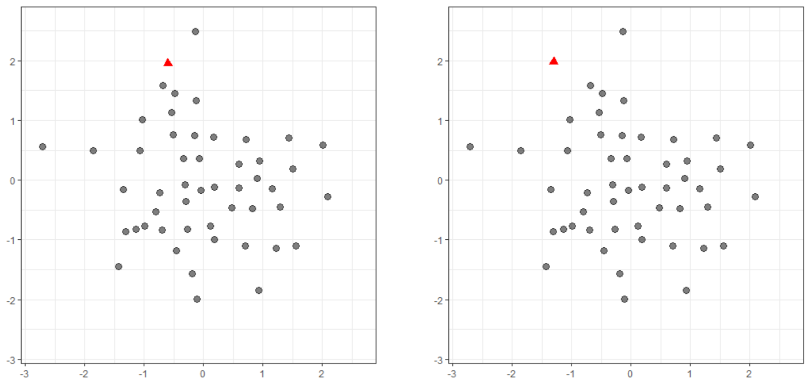


Figure 8.4: *Adding a point to the previously binned bivariate normal configuration. The affected point(s) are highlighted as red triangles.*

8.4.2 Perform the Next Split in Sequence

Since the tree-based binning algorithm creates bins in sequence, performing the next split in the sequence is a simple task. This is akin to wondering what the configuration would look like if the user asked for $b+1$ bins as opposed to b bins. This is done using the `doNextSplit` function, which takes as input

`treebinr.obj`	An object of class `treebinr`.
`selectBin`	A function for choosing between bins to be split.
`splitBin`	A function for splitting a bin.
`binMeasure`	A function to compute the measure associated with each bin.
`makePoint`	A function to turn the contents of a bin into a single point.
`inputs`	A list containing additional input parameters required by user supplied functions.

The function returns a `treebinr` object with the newly formed bin(s). Figure 8.5 illustrates the `doNextSplit` function by performing the next split in the sequence of the bivariate normal point configuration. The affected bin in the original configuration and the two newly formed bins in the resulting configuration are shown as red triangles.

```
norm.gap3 <- doNextSplit(treebinr.obj = norm.gap,
                         selectBin = gapSelect,
                         splitBin = gapSplit,
                         binMeasure = gapMeasure,
                         makePoint = gapPoints,
                         inputs = list(dimRange = dimRange, tau=1))
```

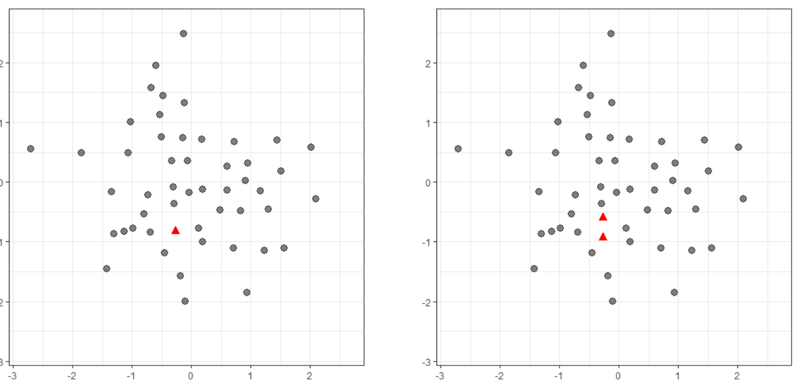


Figure 8.5: *Executing the next split in sequence for the previously binned bivariate normal configuration. The affected point(s) are highlighted as red triangles*

8.4.3 Undo the Last Split in Sequence

Similar to the `doNextSplit` function, the `undoLastSplit` function allows a user to go backwards along the binning tree, and undo the last split in the sequence. This is equivalent to the user asking for $b-1$ bins instead of b bins. The function takes the following input variables

`treebinr.obj`	An object of class `treebinr`.
`binMeasure`	A function to compute the measure associated with each bin.
`makePoint`	A function to turn the contents of a bin into a single point.
`updateBin`	A function to combine several `bin` objects into a single bin object.
`inputs`	A list containing additional input parameters required by user supplied functions.

The function returns an object of class `treebinr`, with the last split in the sequence undone.

The `undoLastSplit` function requires the input function `updateBin` which has yet to be discussed. This function takes two (or more) `bin` objects, and converts them into a single bin object. It must take as input the following

`bins`	A list of `bin` objects to be combined.
`binMeasure`	A function to compute the measure associated with each bin.
`inputs`	A list containing additional input parameters required by user supplied functions.

A valid `updateBin` function must return a single `bin` object. As an example, consider the one for *GapBin*

```
gapUpdate <- function(bins, binMeasure, inputs){

  #Using pruned nodes, update information
  newContents <- c()
  newLowerBounds <- c()
  newUpperBounds <- c()

  for(i in 1:length(bins)){
    newContents <- rbind(newContents, bins[[i]]@contents)
    newLowerBounds <- rbind(newLowerBounds, bins[[i]]@info$binRange[1,])
    newUpperBounds <- rbind(newUpperBounds, bins[[i]]@info$binRange[2,])
  }

  newBoundary <- rbind(apply(newLowerBounds, 2, min), apply(newUpperBounds, 2, max))

  newBin <- bin(boundary = bins[[1]]@boundary,
                measure=NULL,
                contents = newContents,
                index = inputs$node,
                info=list(binRange = newBoundary))

  newBin@measure <- binMeasure(newBin, inputs)

  return(newBin)
}
```

To demonstrate the `undoLastSplit` function, again consider the bivariate normal configuration. With the `doNextSplit` function, the number of bins was increased from 500 to 501. Using `undoLastSplit`, this split is undone, yielding the original configuration of size 500. Affected points are highlighted as red triangles.

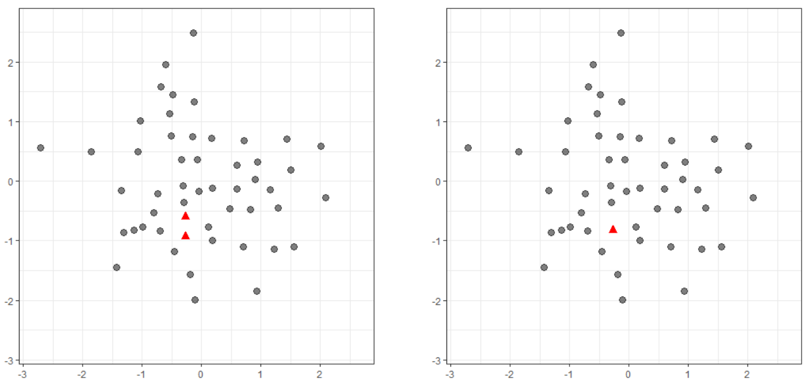


Figure 8.6: *Undoing the last split in the sequence for the previously binned bivariate normal configuration. The affected point(s) are highlighted as red triangles.*

8.4.4 Split a Chosen Bin

While `doNextSplit` will perform the next split in the sequence, it may be the case that the user wants to split a bin of their choosing. For that purpose, the `treeSplit` function is used, taking the following input variables

`bin`	An object of class `bin` to be split.
`treebinr.obj`	An object of class `treebinr`.
`binMeasure`	A function to compute the measure associated with each bin.
`splitBin`	A function for splitting a bin.
`makePoint`	A function to turn the contents of a bin into a single point.
`inputs`	A list containing additional input parameters required by user supplied functions.

A logical choice for a bin to be chosen for splitting would be the bin with the largest count. The following example splits the "max count" bin in the bivariate normal configuration. Affected bins are highlighted as red triangles.

```
max.count <- which.max(norm.gap@counts)
biggest.bin <- norm.gap@bins[[max.count]]

norm.gap5 <- treeSplit(bin = biggest.bin,
                   treebinr.obj = norm.gap,
                   binMeasure = gapMeasure,
                   splitBin = gapSplit,
                   makePoint = gapPoints,
                   inputs = list(dimRange = dimRange, tau=1))
```

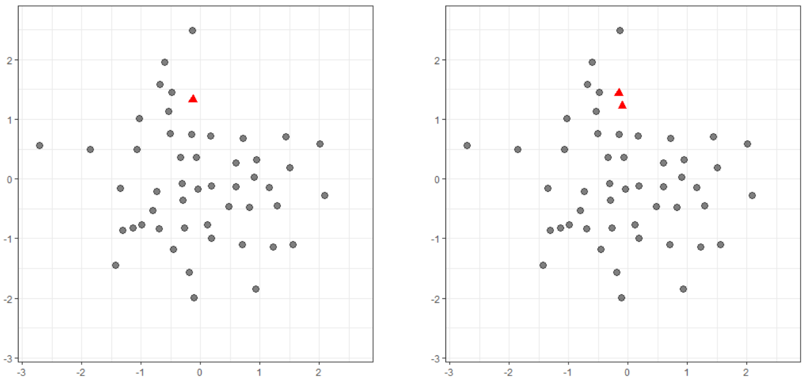


Figure 8.7: *Splitting the bin with the largest count in the previously binned bivariate normal configuration. The affected point(s) are highlighted as red triangles.*

8.4.5 Prune a Bin

The `treePrune` function performs the inverse operation of the `treeSplit` function. Given a `bin` object, the `treePrune` function merges it with the other bins created with it (i.e. the bins created when the parent bin was split). It takes as input the following

`bin`	An object of class `bin` to be pruned.
`treebinr.obj`	An object of class `treebinr`.
`binMeasure`	A function to compute the measure associated with each bin.
`makePoint`	A function to turn the contents of a bin into a single point.
`updateBin`	A function to combine several `bin` objects into a single bin object.
`inputs`	A list containing additional input parameters required by user supplied functions.

and returns a `treebinr` object with the newly pruned bins. A prudent example would be to prune the bins created using the `treeSplit` function. Again, affected bins are highlighted as red triangles.

```
target.bin <- norm.gap5@bins[[501]]

norm.gap6 <- treePrune(bin = target.bin,
                      treebinr.obj = norm.gap5,
                      binMeasure = gapMeasure,
                      makePoint = gapPoints,
                      updateBin = gapUpdate,
                      inputs = list(dimRange = dimRange, tau=1))
```

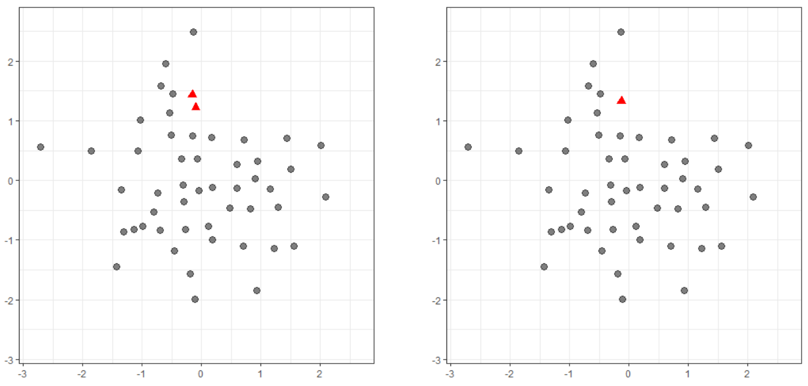


Figure 8.8: *Pruning the bin previously split in the binned bivariate normal configuration. The affected point(s) are highlighted as red triangles.*

8.5 Concluding Remarks

The `treebinr` package implements the general tree-based binning framework in R, allowing for a user to create a custom binning algorithm to meet the needs of their analysis. Three built-in algorithms were also implemented - the novel *GapBin* algorithm, the classic equal width binning algorithm, and the l1-clustering method of [56].

Additional functionality that takes advantage of the tree structure of the binning algorithm was also implemented. The `doNextSplit` function computed the next split in the tree-based binning sequence, which is equivalent to asking for $b + 1$ bins in the original call as opposed to b bins. The inverse operation, `undoLastSplit`, merged the last bins created in the tree-binning sequence, which is equivalent to asking for $b - 1$ bins as opposed to b bins. Similar to these functions were the `treeSplit` and `treePrune` functions, which split and pruned a chosen bin, as opposed to the next one in the binning sequence. These functions are again inverse operators. Other additional functionality included the `addPoint` function, which adds an out-of-sample point to a the binned configuration.

With a software implementation of tree-based binning in place, the claims made in Chapter 7 can now be explored through experimentation. This will be the focus of Chapter 9.

Chapter 9

Binning: Experimental Results

To measure the effectiveness of the unsupervised binning algorithms considered in Chapter 7 in their ability to retain both geometric and probabilistic structure, six configurations are chosen that provide varying levels of structure to capture, including varying levels of density, varying levels of extremeness in the tails of the parent distribution, and varying shapes in the configuration. For each configuration, samples of size 1000 in dimension varying from two to six are considered. Figure 9.1 provides a two dimensional example of each of the configurations that will be considered.

Once simulated, each configuration is scaled to lie in a unit hypercube and randomly rotated to avoid any bias. Each of the binning configurations - equal width, equal frequency, fixed frequency, hexagonal binning (in two dimensions), random sampling, and GapBin - are used to reduce each configuration from 1000 points to at most 500 points.

To assess the retention of geometric structure in the binned configurations, the convex and alpha hulls are used. For each, the area and perimeter are computed. The absolute difference between the original and the binned configurations are taken as measures of difference in geometric structure.

To assess the retention of probabilistic structure in the binned configurations, 250 points are first sampled uniformly from the unit hypercube in the dimension of the original configuration. For each binned configuration (and the original configuration) the density is computed at each of the sampled points using the standard kernel density estimate (with a Gaussian kernel) as discussed in Chapter 2. Using these estimates, three measures are computed - the sum of the absolute difference, Kullback-Leibler divergence, and Jensen-Shannon divergence, all of which were also discussed in Chapter 2. This is repeated fifty times, and the average over all repetitions is taken to be the final measure.

This chapter will be structured as follows. Section 9.1 will introduce the simulation experiment in two dimensions. In addition, the *diamonds* data set will be used as a real data example to assess the ability of the various binning algorithms under consideration. Section 9.2 will continue

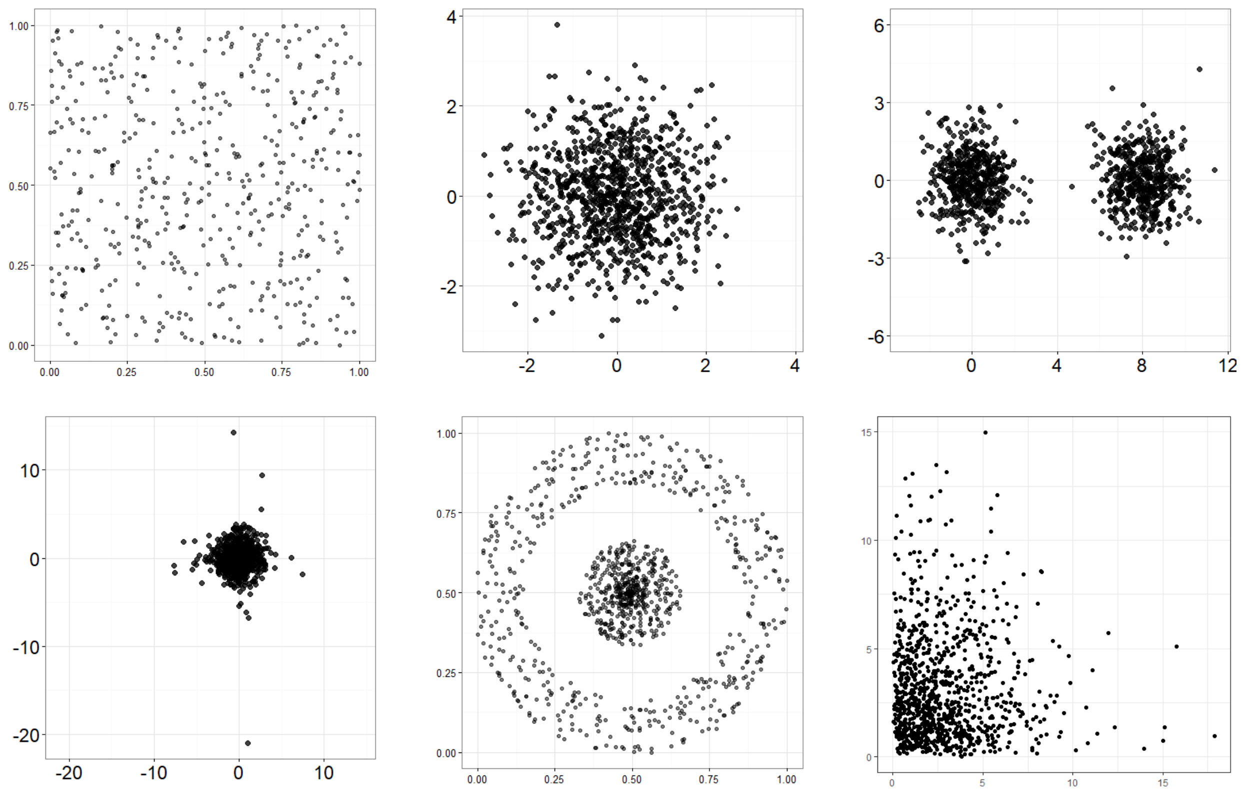


Figure 9.1: *Two dimensional examples of the six point configurations to be used to evaluate the effectiveness of the binning algorithms. In order from top left to bottom right by row: uniform, normal, clustered normals, student-t, donut, and chi-square.*

the simulation experiment in three dimensions, and also introduce the TOSCA high-resolution image data set [14], which will be used to visually assess the ability of each algorithm to retain three dimensional structure and detail. Finally, Section 9.3 will finish the simulation experiment in four dimensions and above. The ability of *GapBin* to bin in very large dimensions will be also be explored using image data sets.

9.1 Two Dimensions

In two dimensions, each of the six binning algorithms can be used. Each configuration from Figure 9.1 is simulated 25 times, and the measures discussed above computed on each. The average over these 25 repetitions for each measure is then computed. Each measure is scaled by its respective maximum (over the algorithms), so that each measure is scaled to be in [0,1]. For each measure, lower values represent better performance. The results of this experiment are summarized in Figure 9.2.

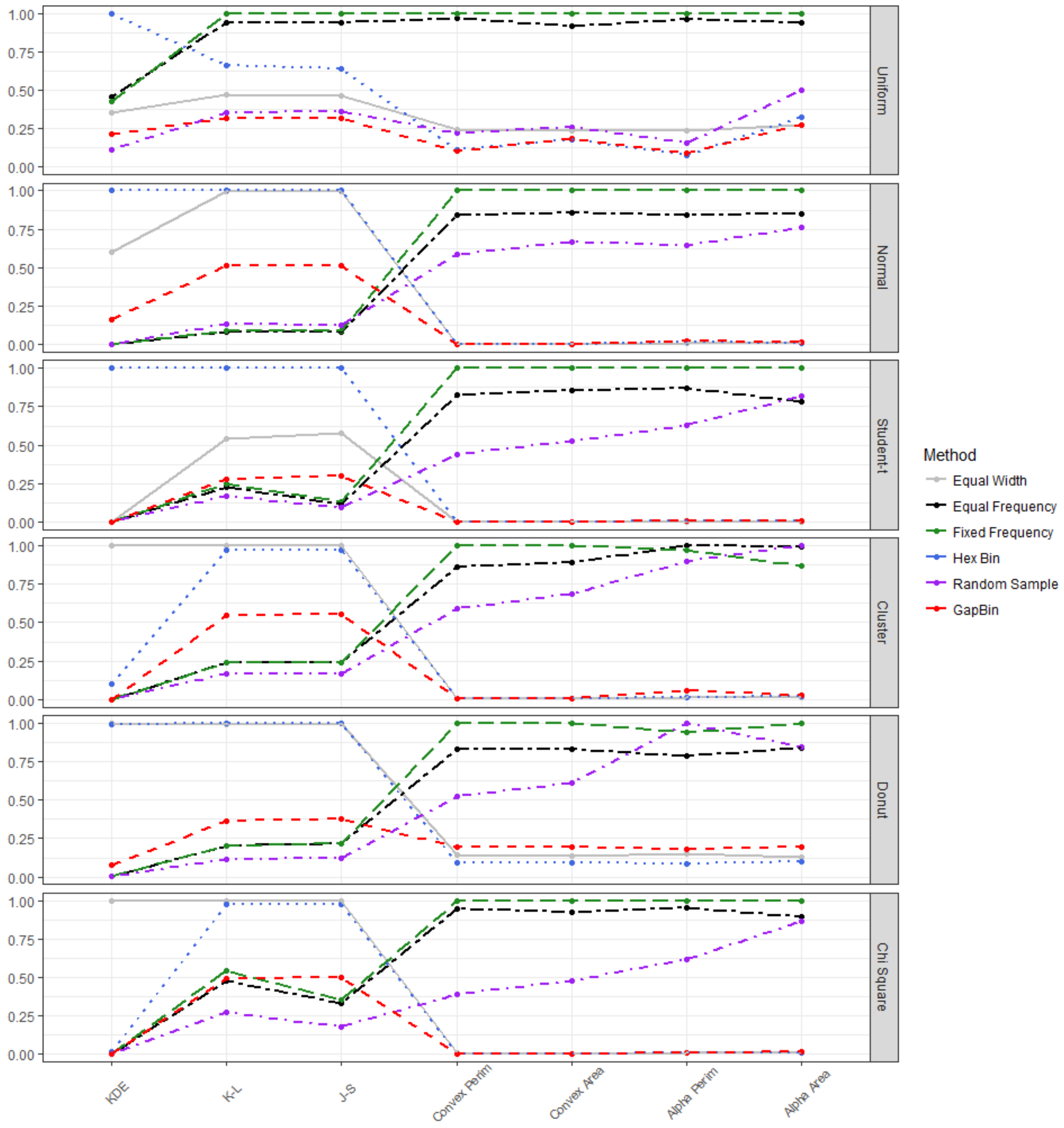


Figure 9.2: *The measures for each of the six binning algorithms on data in two dimensions.*

The two dimensional experiment leads to some very interesting conclusions. The measures have been chosen to capture two distinct types of structure, probabilistic and geometric. Specifically, the absolute difference, Kullback-Leibler, and Jensen Shannon measures on the kernel density estimate capture probabilistic structure, and the area and perimeter of the convex and alpha hulls capture geometric structure. With this in mind, there seems to be a very clear separation of the algorithms - the equal frequency, fixed frequency, and random sampling algorithms perform very well on the three measures designed to capture probabilistic structure, and perform poorly on the those designed to capture geometric structure. Vice versa for both equal width and hexagonal binning. Interestingly, *GapBin* seems to fall somewhere in between. It is highly competitive with equal width and hexagonal binning in the retention of geometric structure, while out-performing both in terms of retention of probabilistic structure. While not as competitive with the equal frequency, fixed frequency, and random sampling algorithms in terms of retention of probabilistic structure, *GapBin* also vastly out performs these algorithms in terms of retention of geometric structure.

What emerges from this experiment is essentially two classes of specialist algorithms: those that retain geometric structure (equal width, hexagonal binning), and those that retain probabilistic structure (equal frequency, fixed frequency, and random sampling). *GapBin* seems to fall in between these two sets of “specialist algorithms”.

9.1.1 Measures on a Real Data Set - Diamonds

Having a real data set to corroborate the results of the simulation experiment is helpful to ensure the observed results of the simulation study are not the result of using artificial data. For that purpose, consider the *diamonds* data (provided in the R package *ggplot2* [117]), a seven dimensional data set containing 53940 entries. For this two dimensional experiment, only the dimensions *depth* and *price* are considered. The data is also scaled to the unit cube so that it is consistent with the simulated data. The original configuration (scaled to the unit cube) is shown in Figure 9.3.

Each of tree-based binning, equal frequency, fixed frequency, equal width, hexagonal binning, and random sampling are used to reduce the size of the two dimensional diamonds data set from 53940 observations to as close to 5000 as can be achieved. The effectiveness of each of the methods is displayed in Figure 9.4.

As was observed in the simulation experiment, a number of interesting issues arise in attempting to bin this data. First, we can see that, due to the distribution of points ranging from very dense to very sparse along the y-axis, the equal frequency and fixed frequency binning methods fail in retaining the shape of the configuration - they instead rely too heavily on retaining the relative density of the distribution, resulting in heavy striation in the more sparse areas. The random sampling approach does not have the same obvious striation, but retains density by eliminating points in the sparse areas, drastically changing the shape of the configuration. In contrast, the

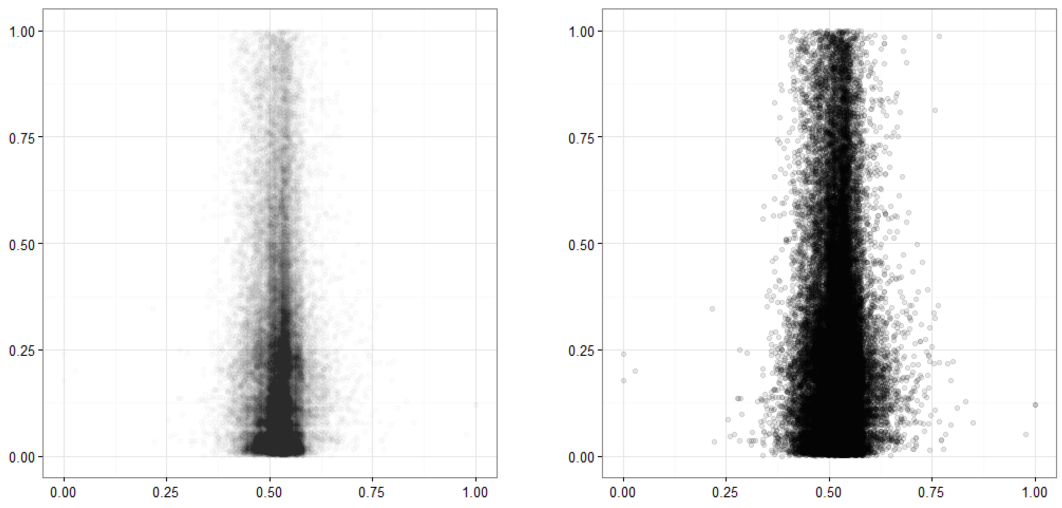


Figure 9.3: *The variables depth (x-axis) and price (y-axis) of the diamonds data set scaled to be in* $[0, 1] \times [0, 1]$. *Two levels of alpha blending are used to emphasize the density of the configuration (on left), and the shape of the configuration (on right).*

equal width and hexagonal binning algorithms retain the shape of the configuration almost perfectly, but completely wash away any semblance of varying density - the reduced configuration is almost uniformly dense across the entire configuration. Again, *GapBin* appears to fall somewhere in the middle of these algorithms. It retains many of the outlying points, as well as the (relative) density of the configuration.

Consider computing the measures used in the simulation experiment on the diamonds data. To do so, a sample of 20000 observations from the diamonds data set were chosen at random, and then binned using each algorithm to as close to 5000 points (without exceeding) as possible. Note that 20000 points are chosen since computing kernel densities on nearly 60000 points requires a very large amount of computational power. Sampling a subset of the data also allows the experiment to be repeated. Each of the measures are then computed, and the results summarized in Figure 9.5.

Again, *GapBin* seems to fall in between two distinct sets of algorithms. In evaluating the kernel density estimation, *GapBin* outperforms equal width and hexagonal binning, while being slightly worse than equal frequency, fixed frequency, and random sampling. In terms of general shape retention, it is outperformed by equal width and hexagonal binning, but outperforms (significantly) each of equal frequency, fixed frequency, and random sampling. This result illustrates that *GapBin* seems to represent a trade-off between density retention and overall shape retention. Each of the other algorithms can be seen as specialists in a sense, focusing on one of shape retention or density retention. This corroborates the results of the simulation story.

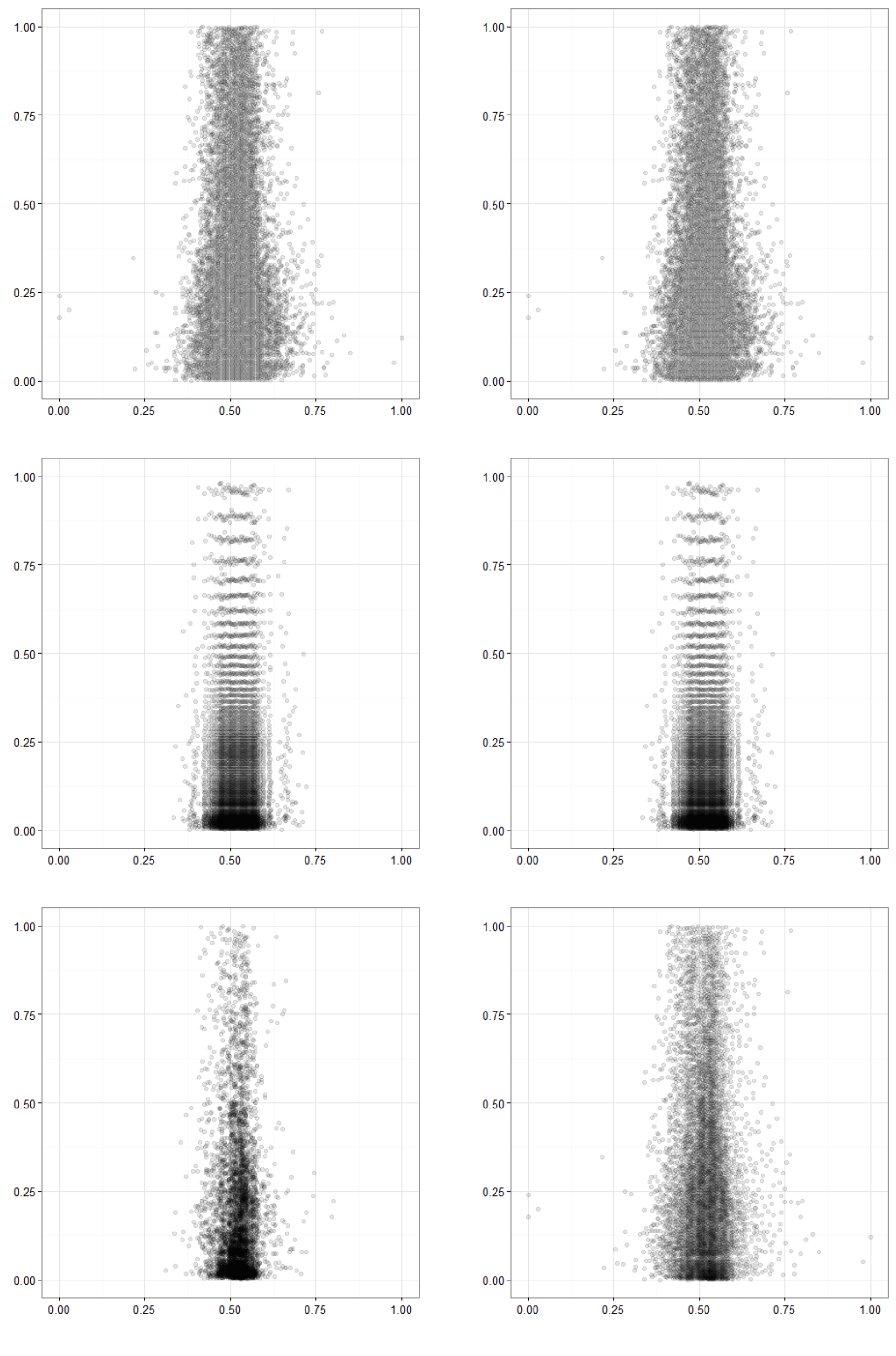


Figure 9.4: *The variables depth (x-axis) and price (y-axis) of the diamonds data set binned to 5000 points using six binning algorithms. From top left (by-row): equal width, hexagonal binning, equal frequency, fixed frequency, random sampling, and tree-based binning.*

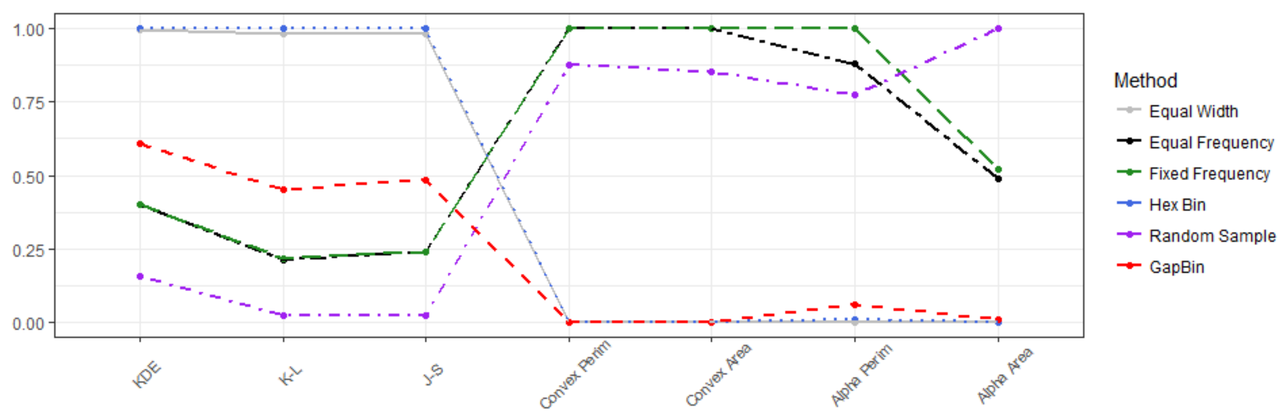


Figure 9.5: *Measure scores for each of the six binning algorithms on the diamonds data set.*

9.2 Three Dimensions

The simulation experiment continues with equal width, equal frequency, fixed frequency, random sampling, and *GapBin*. Each of the point configurations are easily generalized to three (and higher) dimensions. The results of computing the measures on the three dimensional configurations are shown in Figure 9.6.

In three dimensions the same dichotomous behaviour among the four classic algorithms that was observed in two dimensions continues. Of interest, however, is *GapBin*'s improved performance. It continues to be incredibly competitive against equal width binning in terms of geometric structure retention (with very little distinguishable difference in most cases), while also improving its performance against the algorithms who specialize in the retention of probabilistic structure. In fact, we can see that *GapBin* actually outperforms equal frequency and fixed frequency in three of the six configurations. Only random sampling is consistently better in terms of probability retention. This observation signals a potentially important trend in the algorithms under consideration - *GapBin* scales to higher dimensions better than the most popular binning algorithms.

To demonstrate the effect of binning on real data sets, the three dimensional TOSCA high resolution images [14] are used. For each image, each of equal width, equal frequency, random sampling, and *GapBin* are used to reduce the size of the structure from its original size (anywhere from 20000 to 60000 points), to size 5000.

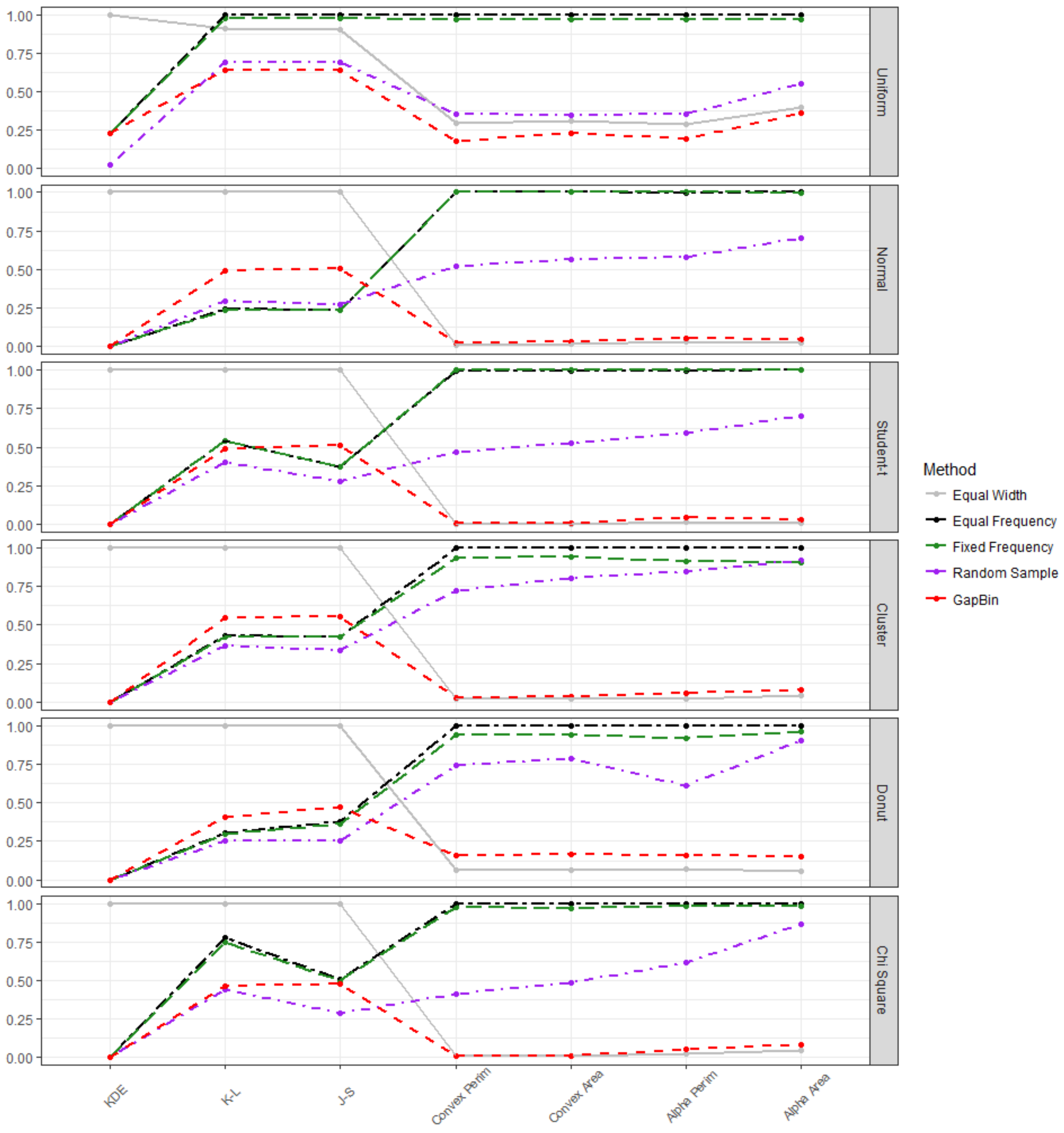


Figure 9.6: *The measures for each of the five binning algorithms on data in three dimensions.*

9.2.1 Cats

In total, there are eleven cat structures, each in a unique pose. Each cat structure contains 27894 individual points. Figure 9.7 illustrates the structure that will be used. The results of binning this configuration using the various algorithms under consideration are shown in Figure 9.8.

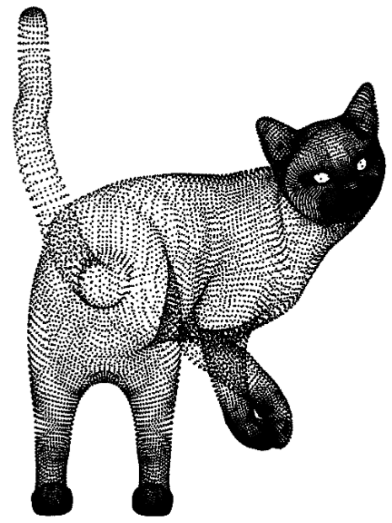


Figure 9.7: *A three dimensional rendering of a cat.*

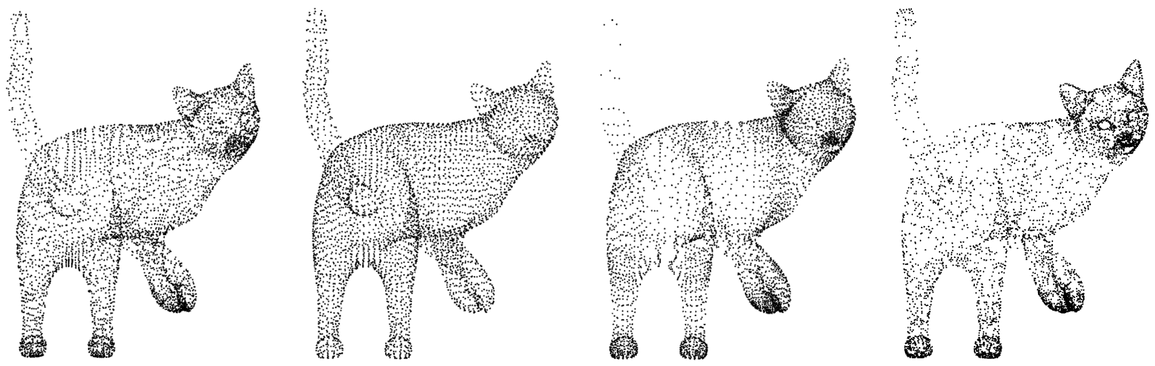


Figure 9.8: *The result of binning Figure 9.7 with (from left to right) GapBin, equal width, equal frequency, and random sampling.*

As expected, the equal width and *GapBin* algorithms best retain the overall shape of the cat. The equal frequency and random sampling algorithms tend to over-represent the high density areas of the cat - its face and paws - at the expense of the overall shape of the cat - its tail and body. An example of shape distortion is in the tail of the cat, which is essentially erased in equal frequency.

The face of the cat is very detailed, and each of the algorithms vary in the way in which that detail is preserved. Equal frequency and random sampling both do a good job in preserving the structure of the face. *GapBin* preserves a lot of the structure, such as the nose and eyes, but it is overall less visible than either of equal frequency or random sampling. *GapBin* seems to be effective in preserving the paws of the cat, however. In the case of equal width, the face of the cat has been essentially erased. This is consistent with what has been seen in previous experiments.

9.2.2 Centaur

The Centaur structure is much smaller than the Cat structure, containing only 15768 points. The original image is shown in Figure 9.9. Similar to the cat structure, interest lies in how well each of the binning algorithms retain the structure of the centaur. The results of binning the Centaur data set to 5000 points using each of equal width, equal frequency, random sampling, and *GapBin* are shown Figure 9.10.

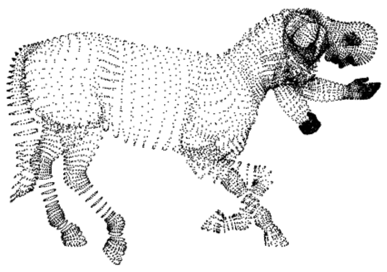


Figure 9.9: *A three dimensional rendering of a centaur.*

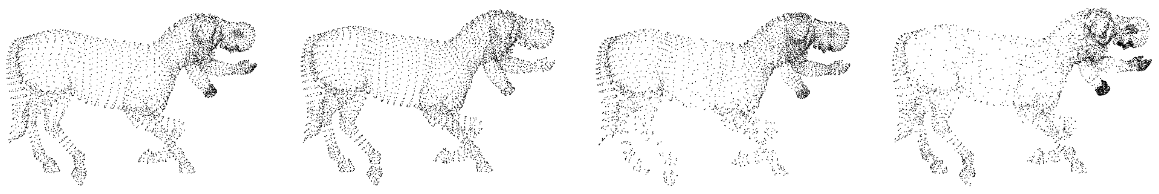


Figure 9.10: *The result of binning Figure 9.9 with (from left to right) GapBin, equal width, equal frequency, and random sampling.*

Much like the Cat structure, the results of binning the Centaur structure with either equal frequency or random sampling results in a high level retention in the facial features at the expense of the overall shape of the centaur - the legs being the most prominent feature missing, as well as the body of the centaur. Using equal width, the overall shape of the centaur is preserved, but features such as the centaurs hands and face are washed away. As has been observed to this point, *GapBin* bridges the gap between these algorithms. The overall shape of the centaur is relatively well preserved, there is no clear distortion in the body or legs, and many of the features of the hands and face are also preserved. For instance, the ear of the centaur is clearly visible in the *GapBin* configuration, while in the equal width configuration, it has been mostly washed away. Similarly, the visible hand of the centaur has clear distinction between the thumb and remaining fingers in *GapBin*, which is not the case in the equal width algorithm.

9.2.3 David

The David data set is the largest under consideration, being 52565 points. Like the Cat and Centaur data, interest lies in the overall shape of the configuration, but in this data set, more attention is paid to the finer detail, specifically in the hands and face. Figure 9.11 illustrates the three images of David that will be under consideration.

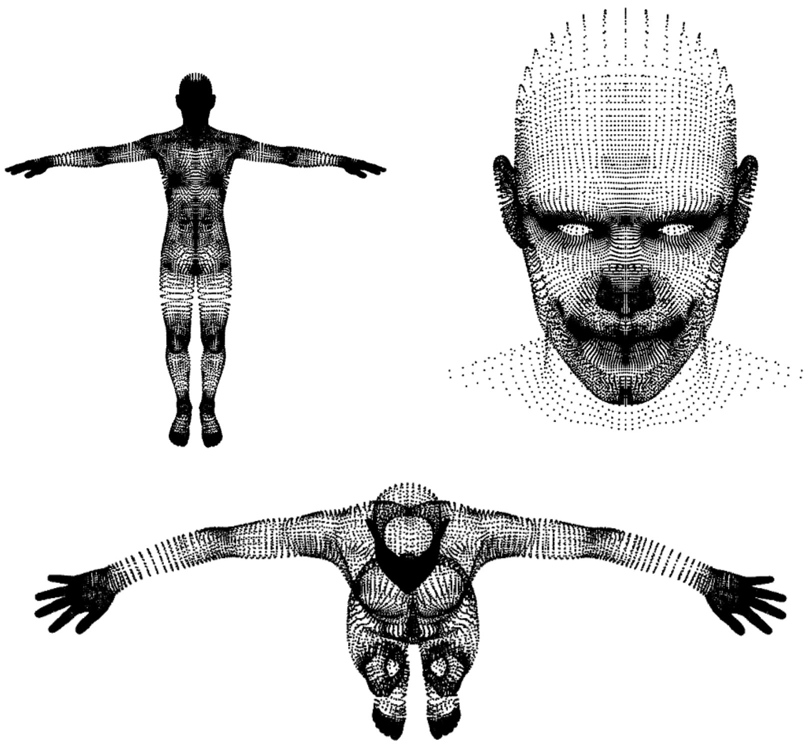


Figure 9.11: *Three different views of the David structure. The top left image will be used to asses the ability of the algorithms to retain a general shape, while the top right and bottom images will be used to assess the ability of the algorithms to retain finer detail.*

Beginning with the entire body, Figure 9.12 illustrates the results of binning the configuration from 52565 to 5000 points using each of the four algorithms.

Looking at the entire data set, the same story that was observed in the Cats and Centaur data sets holds here - equal frequency and random sampling sacrifice shape for detail, equal width

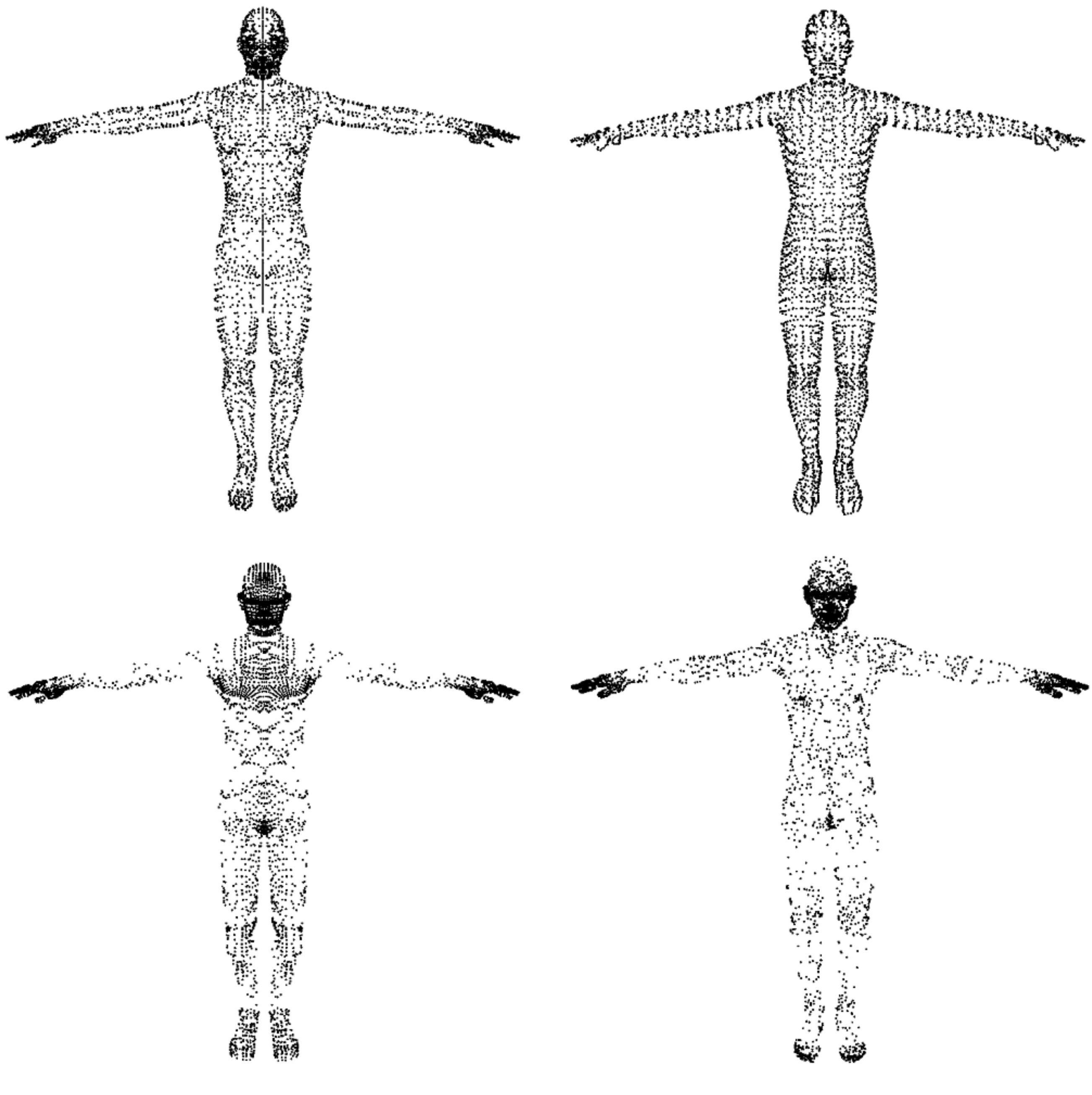


Figure 9.12: *Binning of the David configuration using each of GapBin, equal width, equal frequency, and random sampling.*

sacrifices detail (heavily in this case) to retain shape, and *GapBin* finds itself somewhere in the middle.

Also visible is how much shape is lost by using random sampling and equal frequency, especially in the body, where several holes are visible. Comparably, how much detail is lost using equal width or *GapBin*? For this, examine David's hands, as well as his head. Figure 9.13 provides an overhead view of of David's hands. In particular, notice how the fingers of each hand are formed and defined. From this view, a view of David's feet are also visible. Note how for the equal width algorithm, the hands and feet appear "webbed", while in each of the other algorithms they are recognizable as hands and feet.

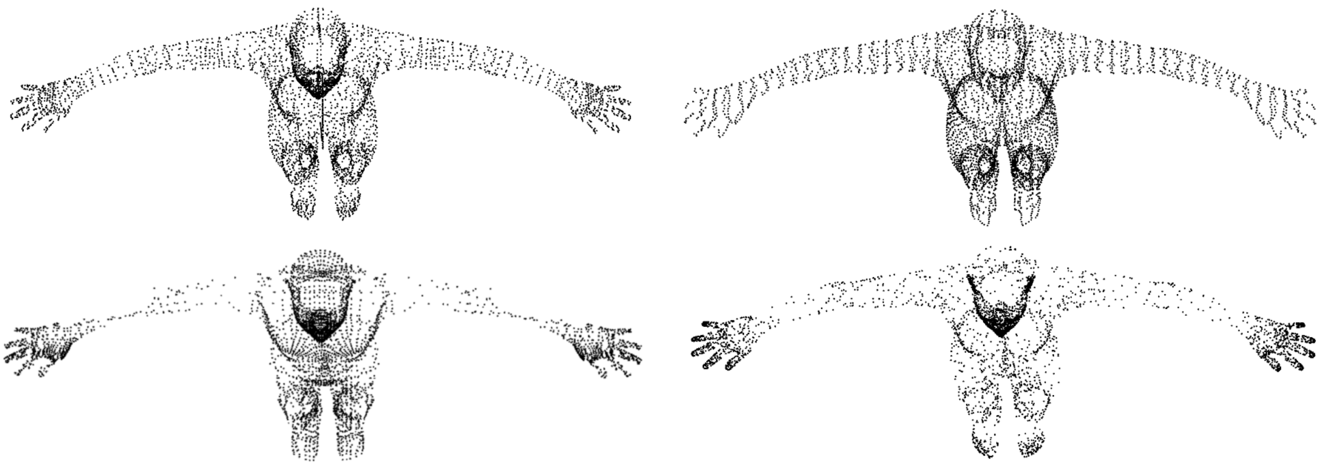


Figure 9.13: *Images of the hands of the binned data configurations using each of GapBin, equal width, equal frequency, and random sampling.*

Next, consider the head of the David configuration. Note that the head alone contains 21505 points, corresponding to 40.9% of the total points in the configuration. This would correspond to approximately 2046 points in the binned configuration were it to retain the same percentage of points. Figure 9.14 shows the detail retained in the face of the configuration.

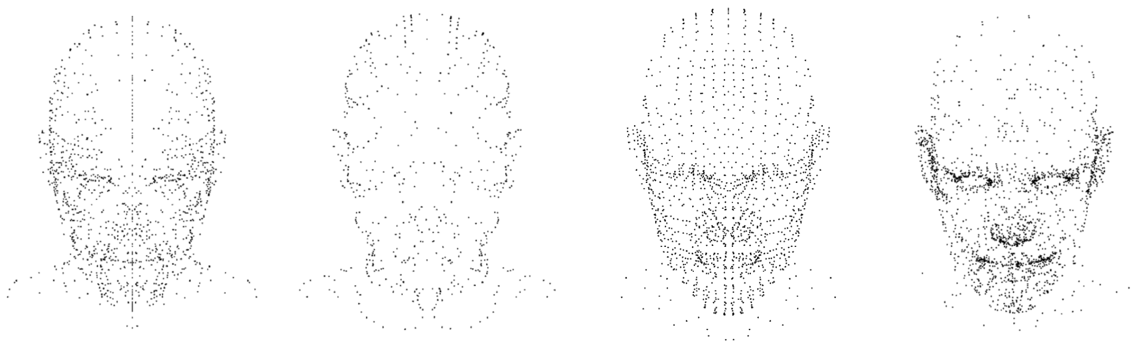


Figure 9.14: *Images of the head of the binned data configurations using each of GapBin, equal width, equal frequency, and random sampling.*

There is large variation in how well the algorithms perform. The equal width algorithm places only 587 points in the head of the configuration (11.74% of the total number of points in the binned configuration), and clearly fares the worst. Comparable are the *GapBin* and equal frequency algorithms, which place 26.02% and 30.56% of the total points in the configuration in the head - still significantly lower than the original, but both are recognizable as compared to the original configuration. Unsurprisingly, the winner is random sampling, which places 2032 (40.64%) of the binned configurations points in the head. This leads to improved detail in the face over the other algorithms, but at the expense the shape of the head (the forehead is almost nonexistent), as well as the overall shape of the entire configuration.

With as much fine detail as exists in the face, it may be more appropriate to bin it directly. Beginning with the original head image shown in Figure 9.11 containing 21505 points, consider a similar compression to what was considered on the entire configuration - the head data is binned to no more than 2500 points. The results for each of the algorithms are shown in Figure 9.15.

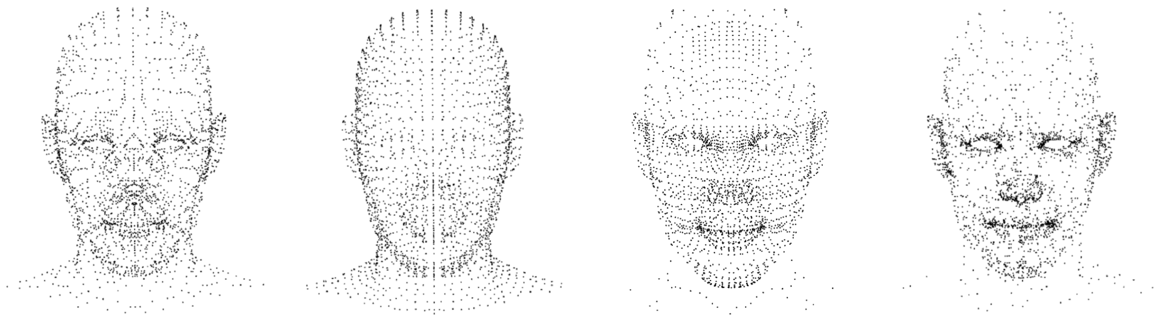


Figure 9.15: *Images of binning only the head of David configuration using each of GapBin, equal width, equal frequency, and random sampling.*

An interesting development emerges in these images. Equal width binning performs the worst by a large margin - the shape of the head is present, but all of the detail (eyes, lips, nose, etc.) have been washed away. Equal frequency does a good job preserving the lips and nose, but seems to eliminate the ears of the head - failing to preserve all of the shape detail in this case. *GapBin* does a relatively good job preserving both the shape and detail of the head. The lips, eyes, nose, and ears are all clearly distinguishable. As was the case previously, random sampling best preserves the detail of the head. As usual, this comes at the cost of the shape - the top of the head and shoulders are not well defined in the random sampling configuration.

9.2.4 Gorilla

The final structure under consideration is the Gorilla structure, which contains 41395 points. Similar to the David structure, the Gorilla structure has some fine detail that can be explored. Of interest is how the algorithms do in total on the entire configuration, and also how well they do in retention of the fine detail of the structure - in the case of the Gorilla,the right hand will be used to assess this. Figure 9.16 illustrates these.

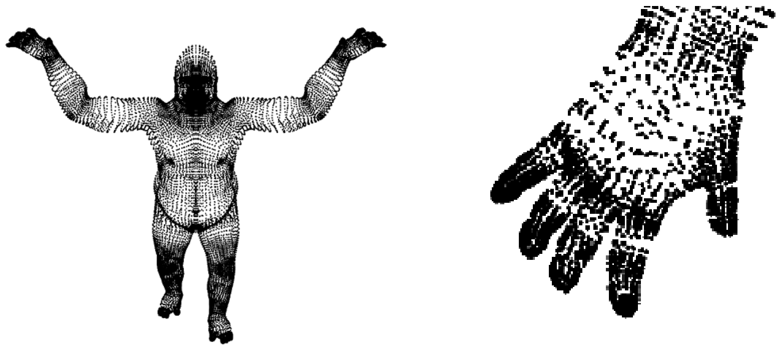


Figure 9.16: *An image of the Gorilla that illustrate its overall shape, and an example of the fine detail in the structure of its hand.*

As with the other structures, first consider how the algorithms fare in binning the entire configuration. Figure 9.17 illustrates the results.

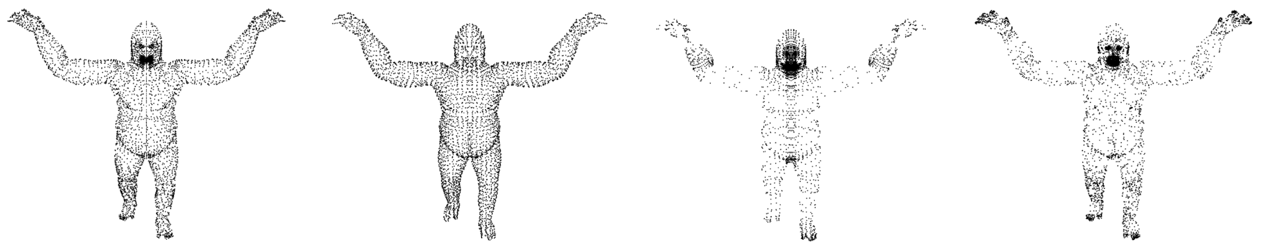


Figure 9.17: *Binning of the gorilla configuration using (from left to right) GapBin, equal width, equal frequency, and random sampling.*

As was observed previously, the algorithms continue to perform in line with expectations. The equal width algorithm washes away all visible detail in the Gorilla, but preserves the overall shape. Equal frequency and random sampling preserve detail in the face and hands, but sacrifice shape in the form of the body of the Gorilla. *GapBin* again is somewhere in-between. The overall shape of the Gorilla is clearly visible, and there is visible density present in the face and hands.

Of greater interest in this configuration is in the the hands of the Gorilla, which can be seen in Figure 9.18. Immediately observable is that, like the head and hands of the David structure, the equal width algorithm does a poor job in retaining the structure of the hands in the Gorilla configuration. The equal frequency algorithm also does quite poorly - the hand of the Gorilla is barely there. This is because the head of the Gorilla is so dense - there is little density in the hand, so the equal frequency algorithm fails to capture it. In this case, *GapBin* does quite well - the hand, and all the individual digits are easily recognizable. As was the case with the David structure, random sampling does perform the best (with the exception of the wrist, which is not well defined), with *GapBin* being a very close second here.

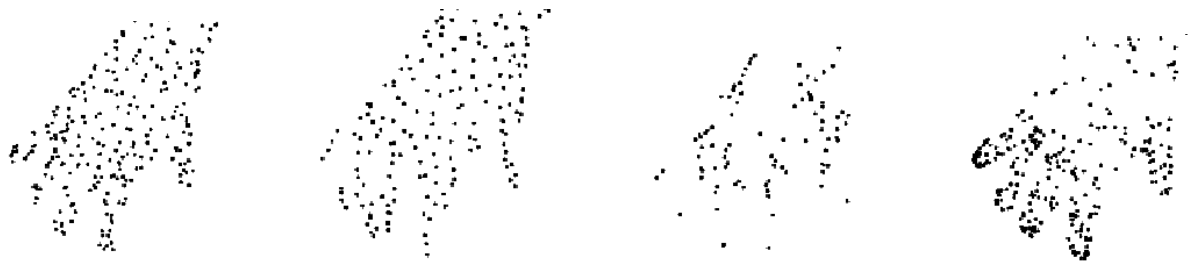


Figure 9.18: *Binning of the hand of the Gorilla configurations using (from left to right) GapBin, equal width, equal frequency, and random sampling.*

9.3 Higher Dimensions

The final step in the simulation experiment is to continue to higher dimensions, up to six dimensions total. The results of these experiments are shown in Figure 9.19 - 9.21

Remarkably notable from this experiment is the improved performance of *GapBin* relative to the other algorithms as dimension increases. *GapBin* continues to be competitive with equal width binning in terms of shape retention (outperforming it on some configurations). Importantly, however, is *GapBin*'s improved performance in terms of density retention, where it begins to outperform both the equal frequency and fixed frequency algorithms. The relative performance of *GapBin* only improves as dimension increases, and in fact as dimension hits six, the *GapBin* algorithm is the best performing algorithm in terms of density retention of all the algorithms under consideration. This makes *GapBin* overall the best performing of the binning algorithms in dimension as low as six.

Building on this success, the next step is to consider the ability of *GapBin* to bin in dimensions much larger than six. Six image data sets will be used to assess *GapBin*'s ability to bin data in very high dimension

1. Statue manifold data (shown in [103])
2. Frey faces data set
3. Olivetti faces data set (from AT&T Laboratories at Cambridge)
4. UMIST faces data set [48]
5. USPS handwritten digits data set [40]

Each of these data sets will be binned in their original dimensional space, and then reduced to (at most) four dimensions using simple multidimensional scaling. Loon [115] will be used to

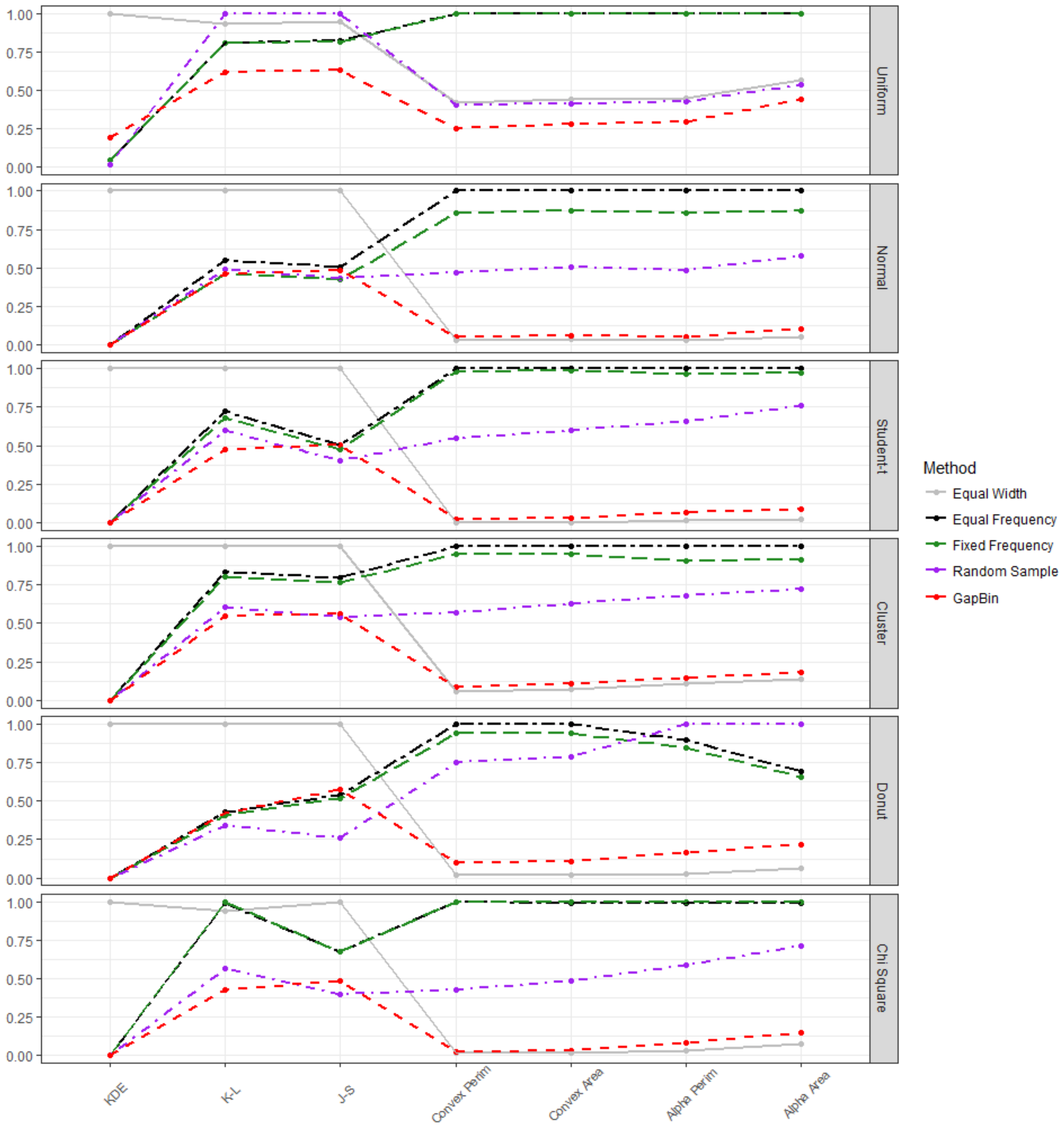


Figure 9.19: *The measures for each of the five binning algorithms on data in four dimensions.*

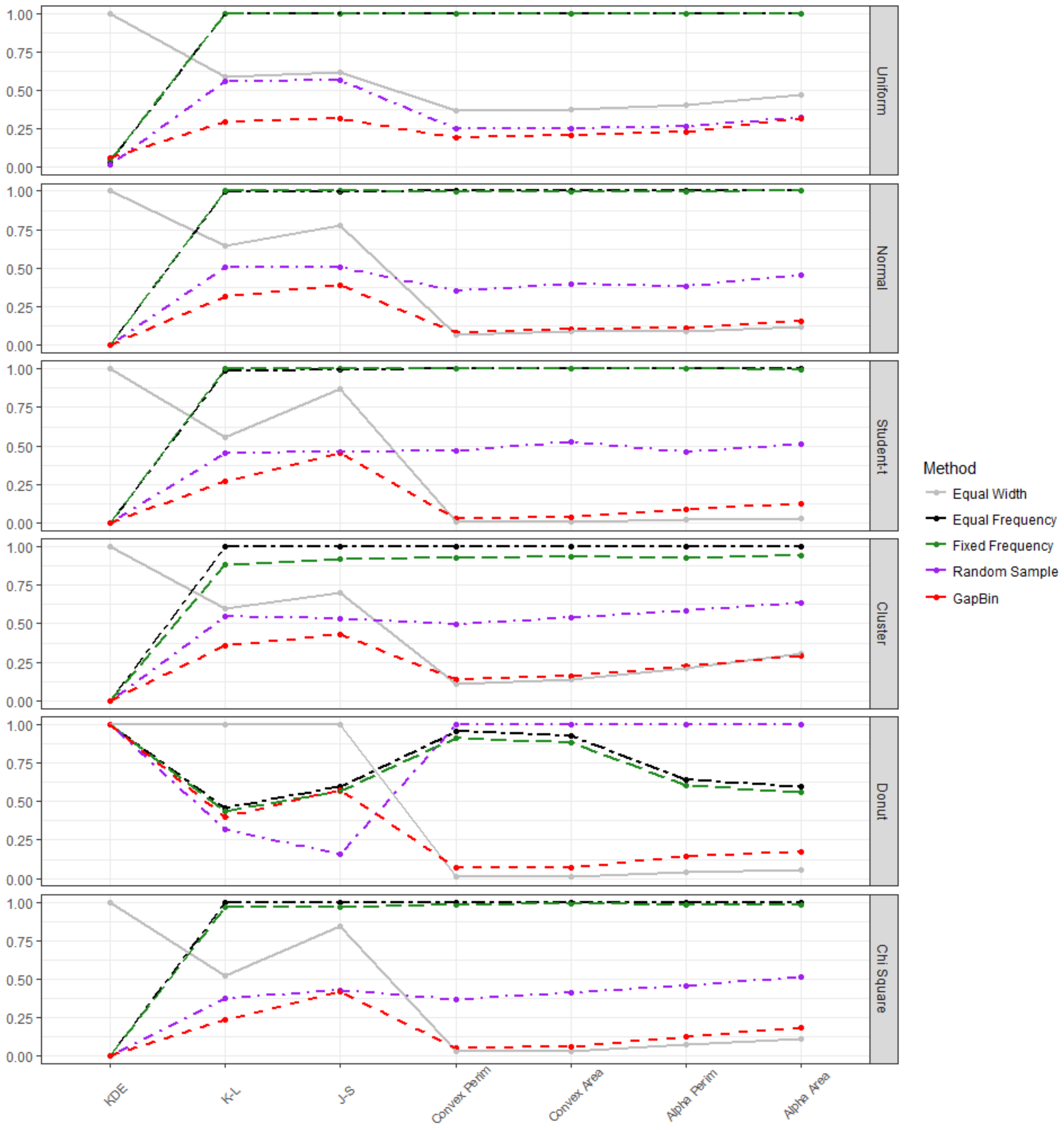


Figure 9.20: *The measures for each of the five binning algorithms on data in five dimensions.*

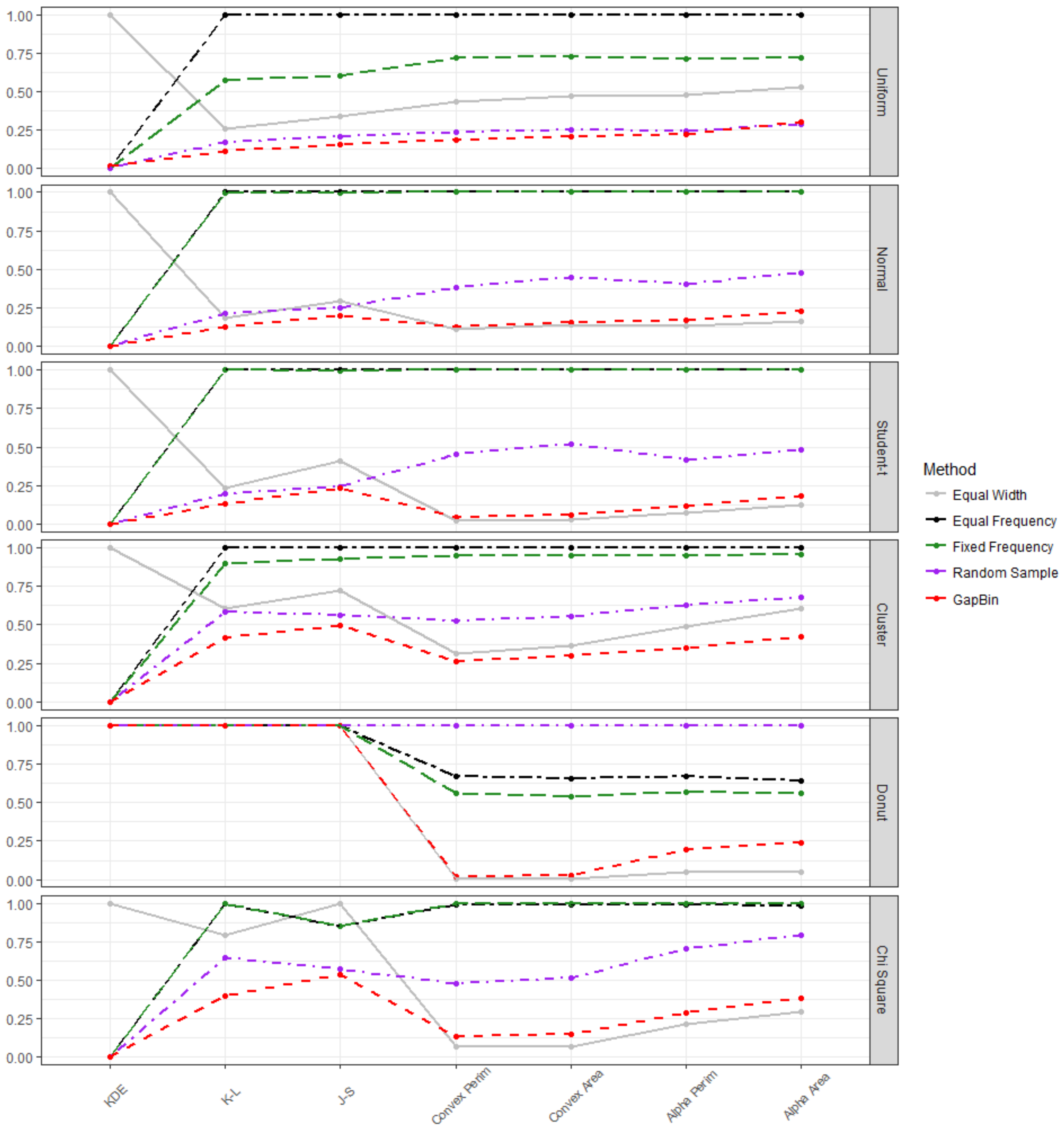


Figure 9.21: *The measures for each of the five binning algorithms on data in six dimensions.*

view the first two principal components of the data sets to observe the differences between the original and binned configurations. In particular, the overall shape of the two dimensional view will be of interest, including the retention of outlying points in the configuration. Also of interest will be the observable pattern of the images in the two dimensional view. For instance, comparing the transition of lighting in the images from light to dark as compared to the original, or the transition of the facial images across the two dimensional view will be observed.

Individual bins created by *GapBin* will also be observed, with the goal being to visualize bins where *GapBin* has performed exceptionally well, and also where it has not.

9.3.1 Statue Manifold

The statue manifold data consists of 698 images of a bust in various positions and lighting conditions. The images are 64×64 grey scale images, meaning the data are in 4096 dimensional space. Using *GapBin*, the data are reduced to 349 images from 698 in the original 4096 dimensional space. The data is then reduced to three dimensions using multidimensional scaling (three dimensions is chosen as the prevailing belief in the dimension reduction community is that this data set adequately represents a three-dimensional manifold). Figure 9.22 shows the original and binned data on its first two principal components, illustrating the density using alpha blending. Figure 9.23 shows the same data, but replaces the data points with the images they represent to give a sense of shape and pattern retention.

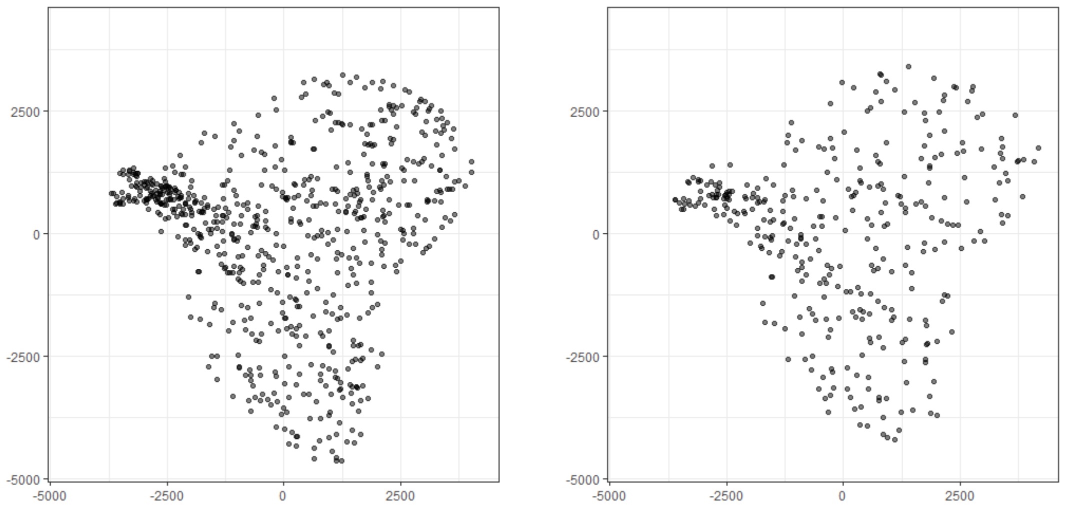
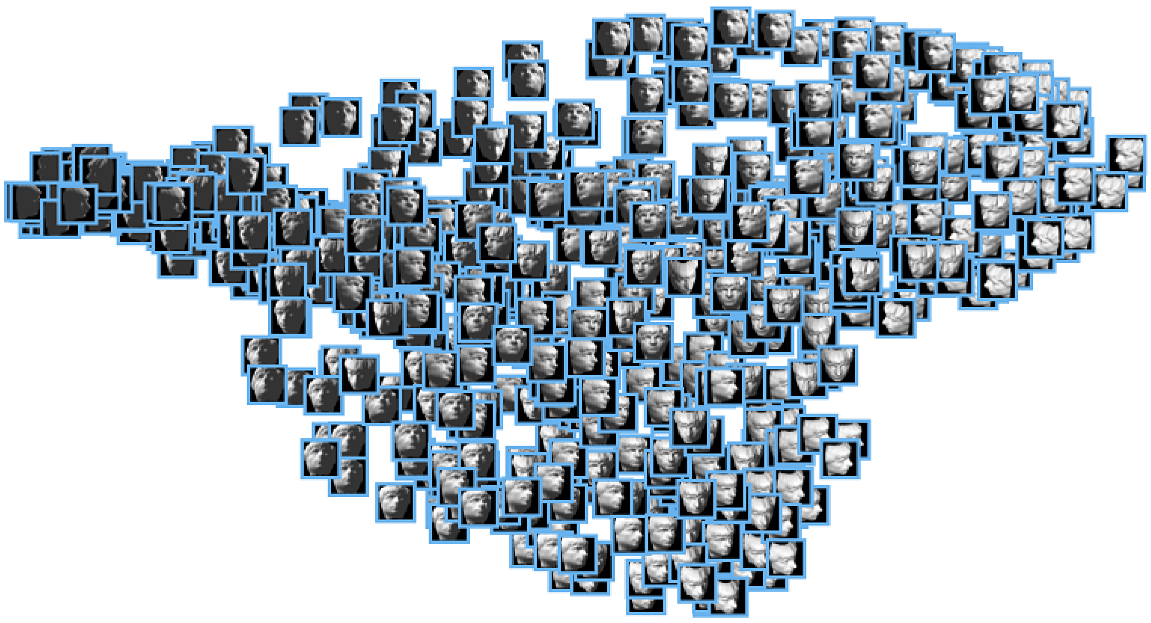


Figure 9.22: *A view of the first two dimensions of the three dimensional manifold produced by multidimensional scaling of the statue data. The left image is the original data, and the right image is the binned data.*



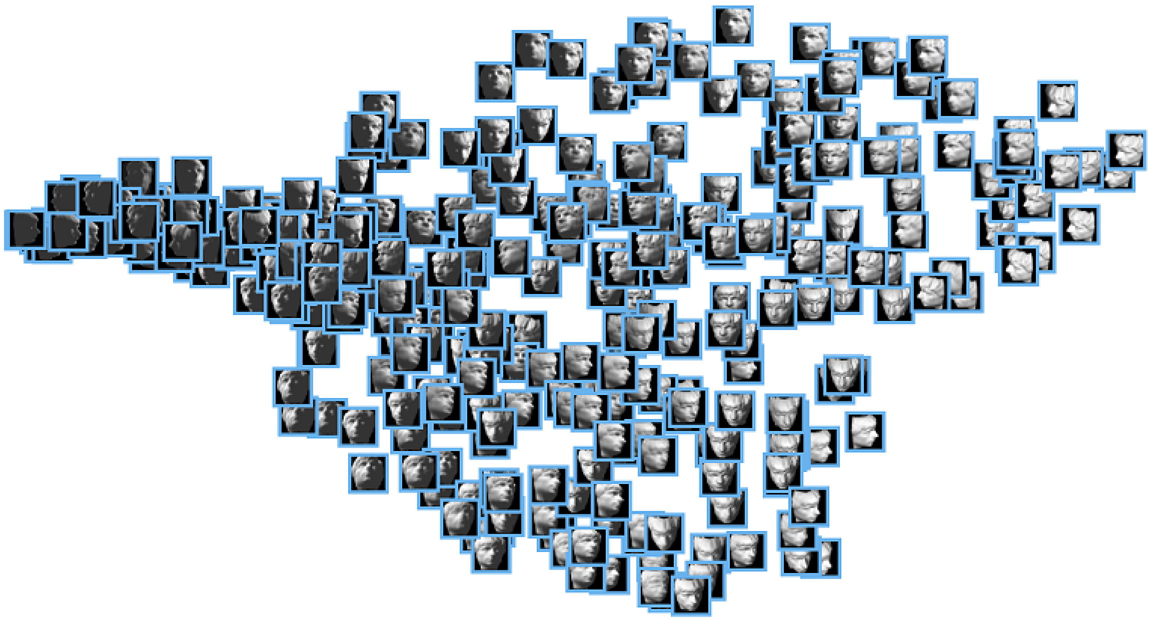


Figure 9.23: *A view of the first two dimensions of the three dimensional manifold produced by multidimensional scaling of the statue data. The top image is the original data, and the bottom image is the binned data.*

The similarities between the two configurations are striking. First note that the overall shape of the two configurations is very similar, and the binned configuration appears as a thinned out version of the original. Looking from left to right in Figure 9.23, a clear change in colour from dark to light among the busts is present in both images.

Next, consider some of the individual bins formed by *GapBin*. Figure 9.24 illustrates some of the best (and largest) bins, while Figure 9.25 illustrates some of the worst.

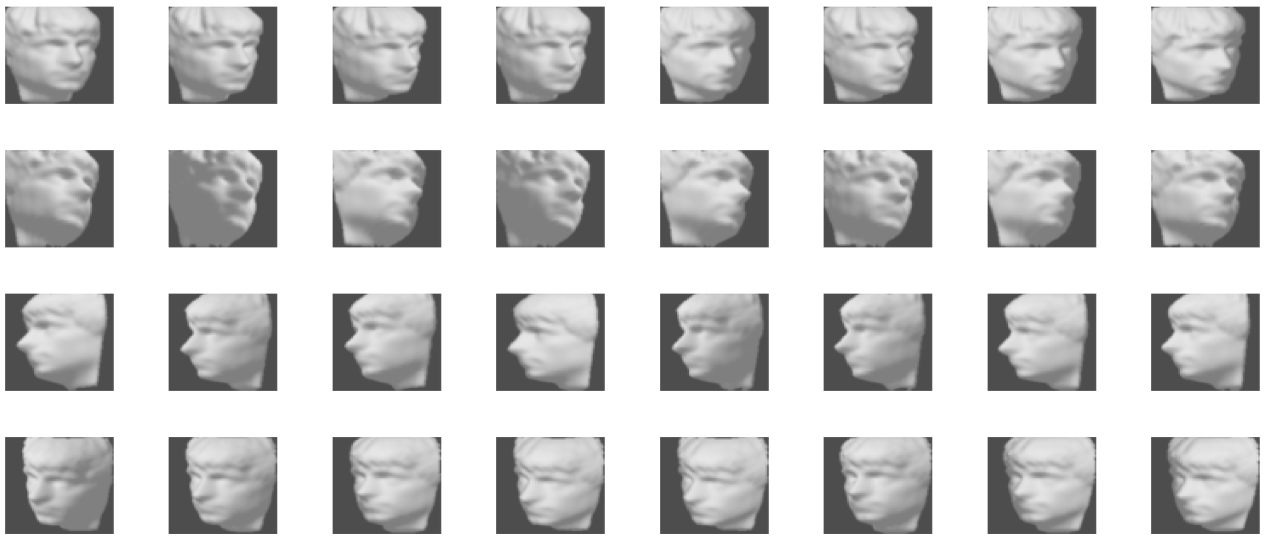


Figure 9.24: *Four different bins (one in each row) produced by GapBin. Each bin contains eight very similar images of the statue manifold data.*

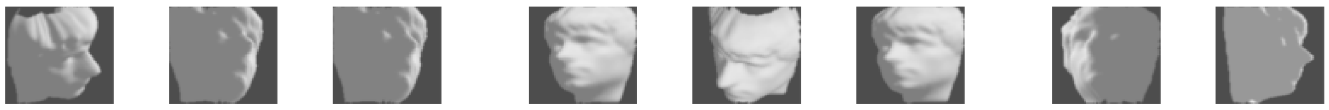


Figure 9.25: *Three different "bad" bins (of size 3, 3, and 2) produced by GapBin. Each of the "bad" bins is relatively small as compared to the larger bins considered in Figure 9.24.*

Interestingly, the four largest bins produced by *GapBin* (shown in Figure 9.24) all contain very similar images, while the poor bins shown in Figure 9.25 are all small by comparison. Also interesting is that the bad bins which contain three images in this case seem to have one "outlier" - two of the images are quite similar, and one is markedly different.

9.3.2 Olivetti Faces

The Olivetti data set compiled by AT&T Laboratories Cambridge consists of ten facial images from forty individuals at varying times, yielding 400 images total. The images are 64×64 grey scale resulting in a 4096 dimensional data set. The data are binned to 250 points using *GapBin*, with multidimensional scaling used to decrease the dimension to four. Note that the Olivetti data set is not that large (only 400 points) and exists in very high dimension, so a 50% compression results in a very poor representation of the original - the resulting binned configuration contains one very large bin, containing over 100 points. For that reason, a much smaller compression ratio is used here.

Figure 9.26 illustrates the density of both the original and binned configurations by showing the first two principal components with alpha blending. Figure 9.27 illustrates the shape and pattern of each configuration by showing the representative image of each point in the configuration - in the case of the binned configuration, this image is taken to be the average face in each bin.

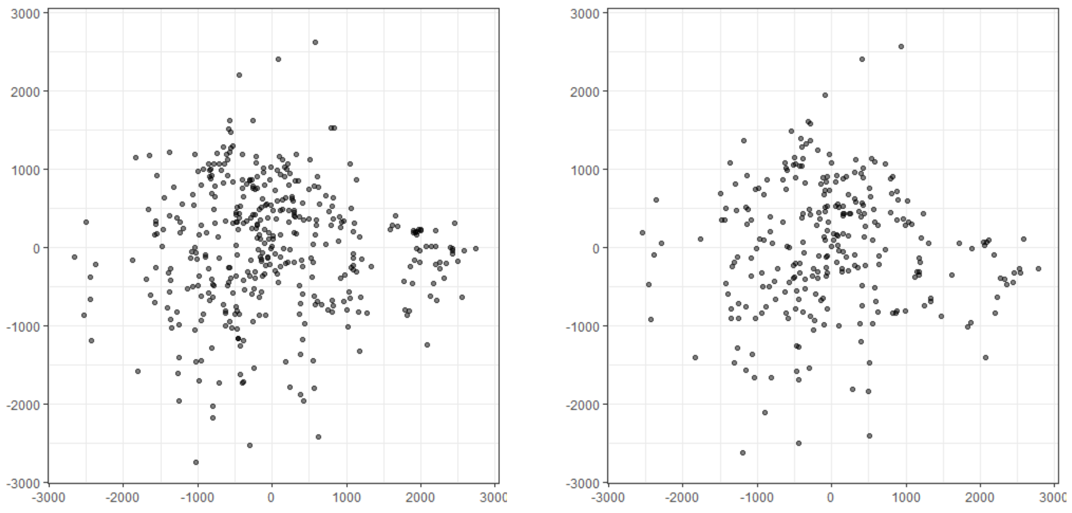
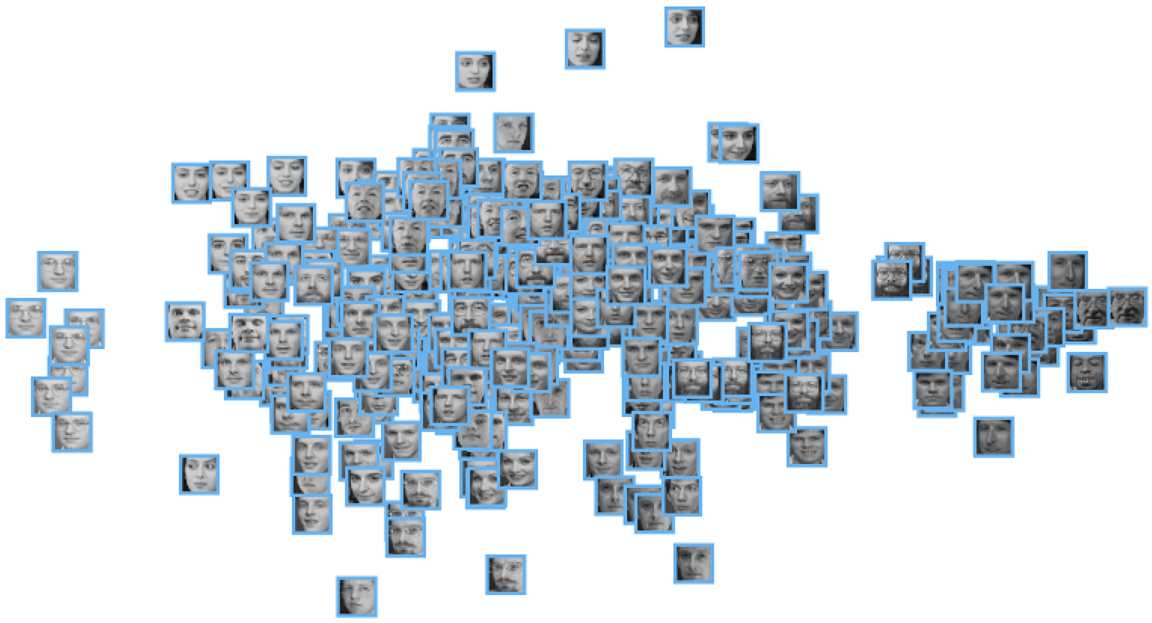


Figure 9.26: *A view of the first two dimensions in the dimension reduction of the Olivetti data to four dimensions total. The left image is the original data, and the right image is the binned data.*

Again, note the similarity between the two configurations. Figure 9.26 shows that overall density is retained. This can be seen in the center of the configuration, as well as the on the right side. Also visible in this figure is the retention of outliers - for instance at the top of the image.



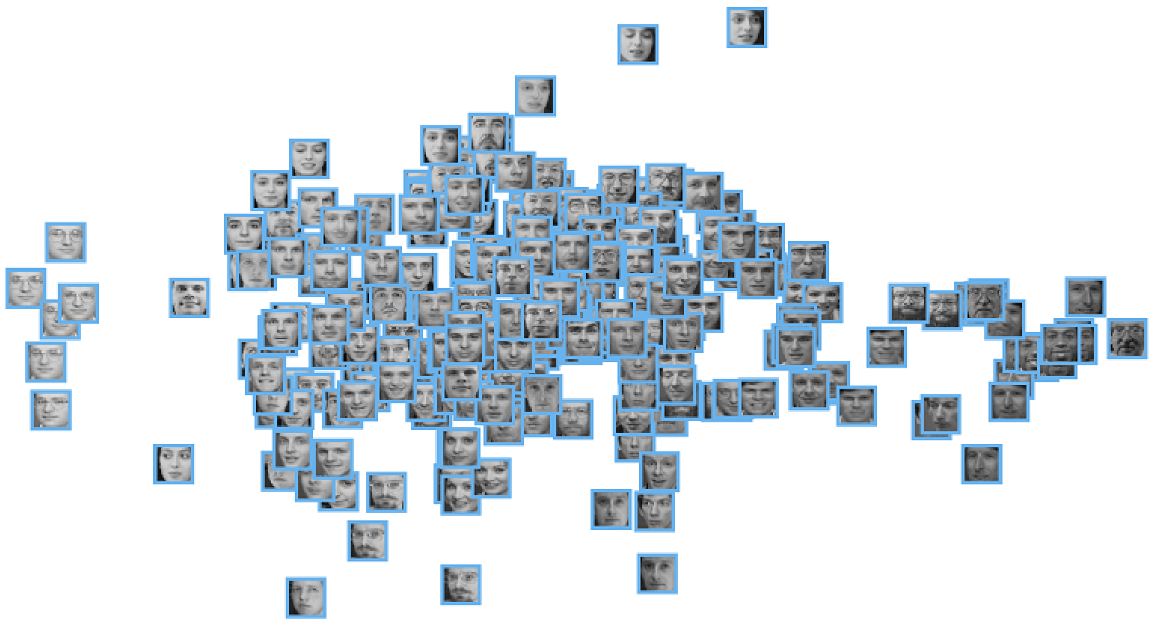


Figure 9.27: *A view of the first two dimensions in the dimension reduction of the Olivetti data to four dimensions total. The top image is the original data, and the bottom the binned data.*

Figure 9.27 illustrates the overall shape and pattern retention of the *GapBin* algorithm. The images move from light to dark when moving left to right, very similar groupings of faces such as the grouping of dark images on the far right of the image can be seen, and the outlying points in the top of the image are also preserved. Note that some some key faces have been lost, especially in the dark clustering of points on the right. Overall though, the similarity between the two images is visible.

As was done with the statue manifold data, consider the individual bins produced by the *GapBin* algorithm. Figure 9.28 illustrates some of the larger bins produced by *GapBin*. Each bin contains five observations of the same person. This is indicative of *GapBin*'s ability to group similar faces together.

Figure 9.29 illustrates the difficulty of binning such a small configuration. It contains 32 images in total, and contains several different people. This can be remedied by asking for more bins in the final binned configuration.

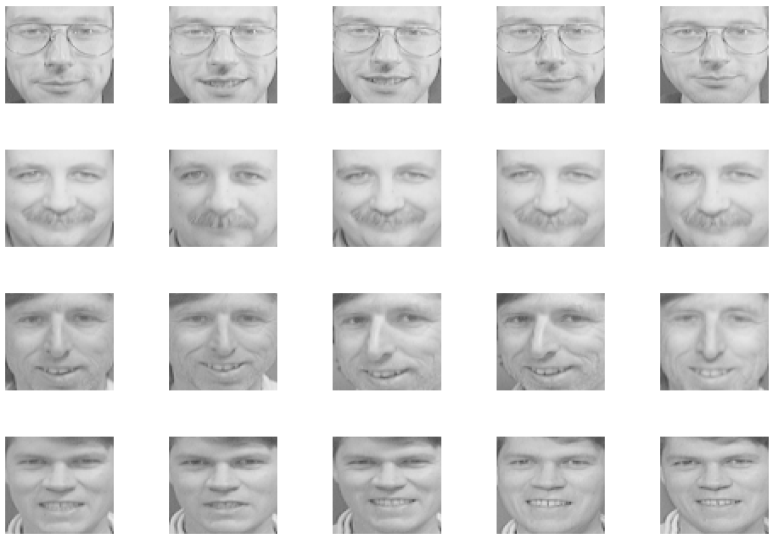


Figure 9.28: *Four different bins produced by GapBin (one in each row) demonstrating the effectiveness of the GapBin algorithm in high dimensions. Each bin contains very similar faces.*

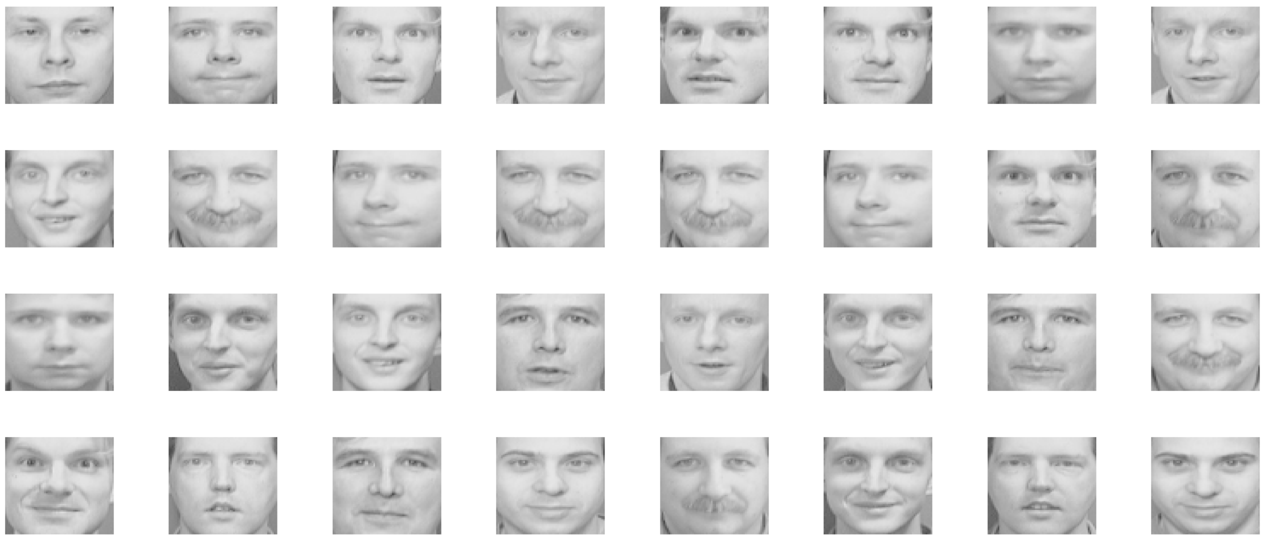


Figure 9.29: *An issue with binning the Olivetti data set due to its high dimensionality and low observation count. Even a modest compression results in one large bin containing the 32 "leftover" images in the center of the configuration. This can be avoided by increasing the number of bins.*

9.3.3 Frey Faces

The Frey faces data set consists of 1964 images of Brandon Frey's face in various poses. The data are 28 × 20 grey scale images, meaning they have a dimensionality of 560. Using *GapBin*, the data set is reduced to 982 images (a 50% compression). The data is reduced to four dimensions using multidimensional scaling. Figure 9.30 illustrates the retention of both shape and density using alpha blending, and Figure 9.31 illustrates the retention of structure using the representative images of each point.

Again, note the similarity between the two manifolds. With there being a large number of points in this data set, the retention of density is much more evident in this image than it was in previous manifolds. The high density regions at the top of the image are still (relatively) high density regions in the top of the binned image. Even some of the high density regions at the lower right corner of the image have been preserved.

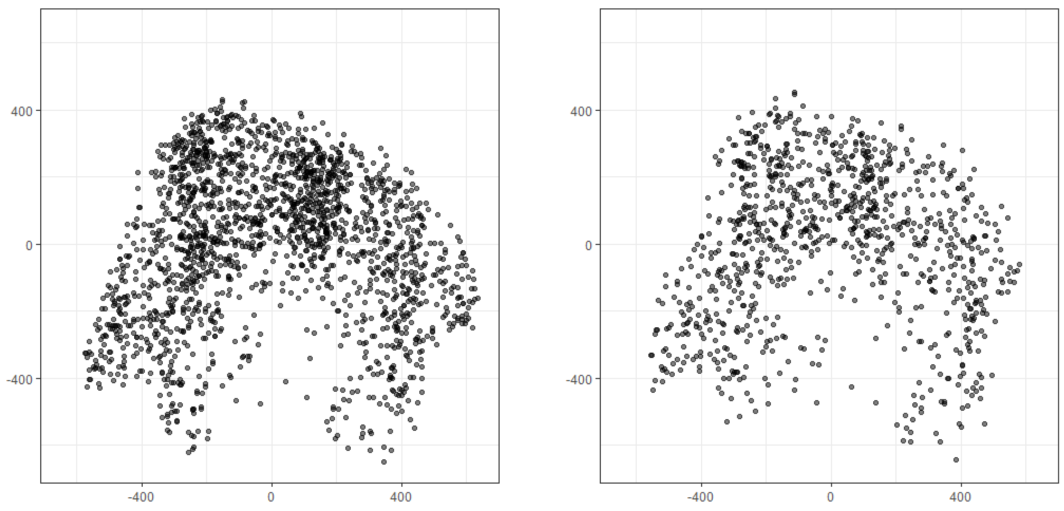


Figure 9.30: *A view of the first two dimensions in the dimension reduction of the Frey data to four dimensions total. The image on the left is the original data, and the image on the right the binned data.*

The overall retention of the shape of the manifold is also very clear. The pattern of the images is also clearly retained, with the faces ranging from frowning to smiling moving up Figure 9.31, and from right looking to left looking moving left to right across the image.

Due to the size and general structure of the Frey faces data set, the *GapBin* algorithm creates several bins (some very large) that contain many similar faces. Figure 9.32 displays three such examples, containing ten images each.

As was the case with the statue manifold data, there are also some bins containing objectively different images. These bins, as was the case with the statue data, tend to be quite small. Figure 9.33 illustrates two such bins. Notable is that, even within these "bad" bins is some similarity between the faces.

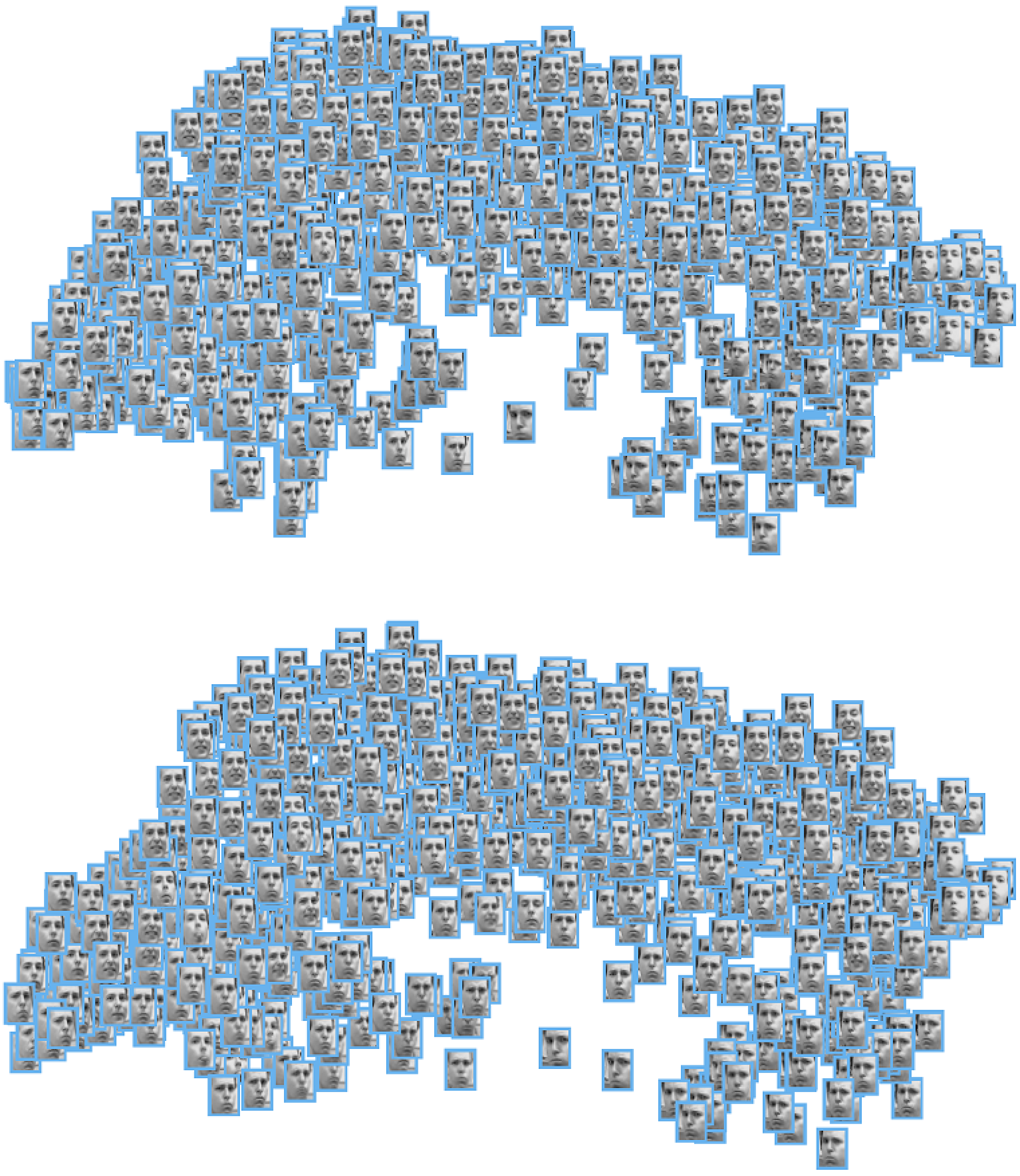


Figure 9.31: *A view of the first two dimensions in the dimension reduction of the Frey data to four dimensions total. The top image is the original data, and the bottom the binned data.*

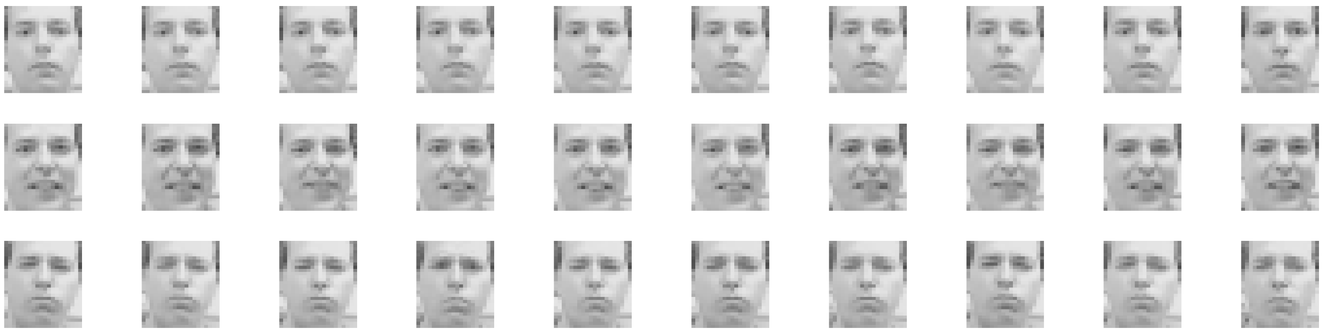


Figure 9.32: *Three bins (one in each row) produced by GapBin containing remarkably similar faces. This is is indicative of the ability of the GapBin algorithm to continue to work well in dimension as high as 560.*

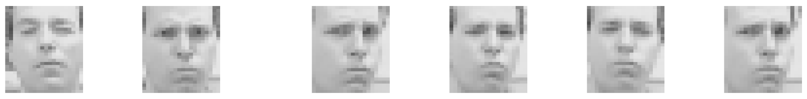


Figure 9.33: *Two bins (of size 2 and 4 respectively) in the binned configuration that contain objectively different faces. Note that, similar to the statue manifold data, the bins with differing faces tend to be quite small.*

9.3.4 UMIST Faces

The UMIST faces data set consists of facial images of 20 individuals. Each image is 112×92 pixels, meaning the data reside in 10304 dimensional space - the highest dimensional space under consideration. In total, there are 575 images, reduced to 288 images using *GapBin* - a 50% compression. Figure 9.34 illustrates the ability of the algorithm to both retain the shape and relative density of the configuration, and Figure 9.35 illustrates the pattern of the images using representative faces.

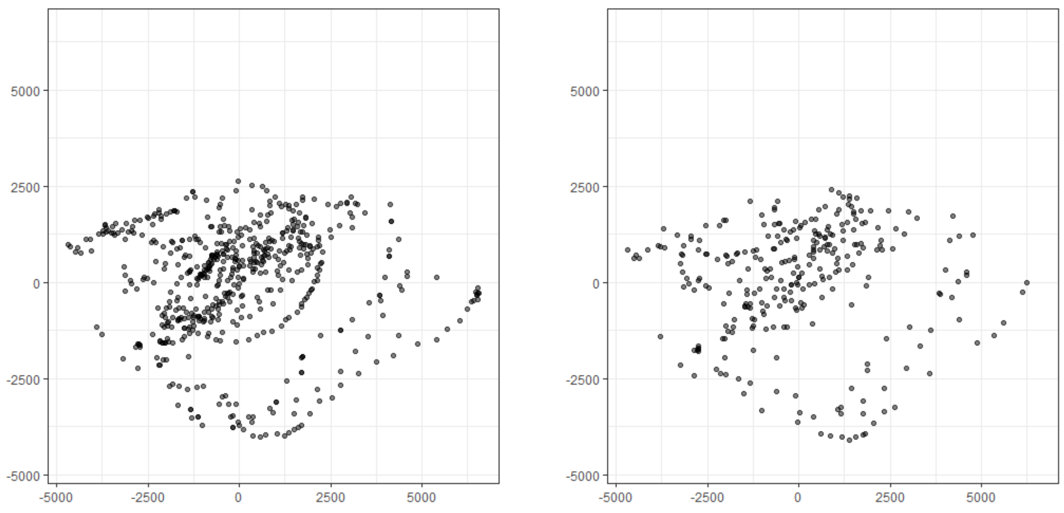


Figure 9.34: *A view of the first two dimensions in the dimension reduction of the UMIST data to four dimensions total. The left image is the original data, and the right the binned data.*

The UMIST faces data set has a very distinct pattern in the point configuration. The lines in the lower right of the image provide a clear point of comparison for the binning method. *GapBin* has managed to preserve large portions of these lines, as they are still visible in the binned configuration. This retained shape does come at a slight cost of density - some of the more dense areas in the image do not appear to have been kept relatively dense in the binned configuration. For example, the dense region beginning around $(-1000, 0)$ does not seem to have been preserved in the binned configuration. This is indicative of the priority placed on preserving the shape of the configuration in this data set - the visible lines in the configuration tended to dominate during binning.

Looking at the representative faces, a very similar story is evident. The pattern of faces along the lines in the lower right portion of the image seem to be very well preserved, while some of the areas of higher density are perhaps more washed away than has been observed previously. Overall, however, the pattern of the two manifolds are still very similar, given the exceptionally high dimension in which this data originally existed.

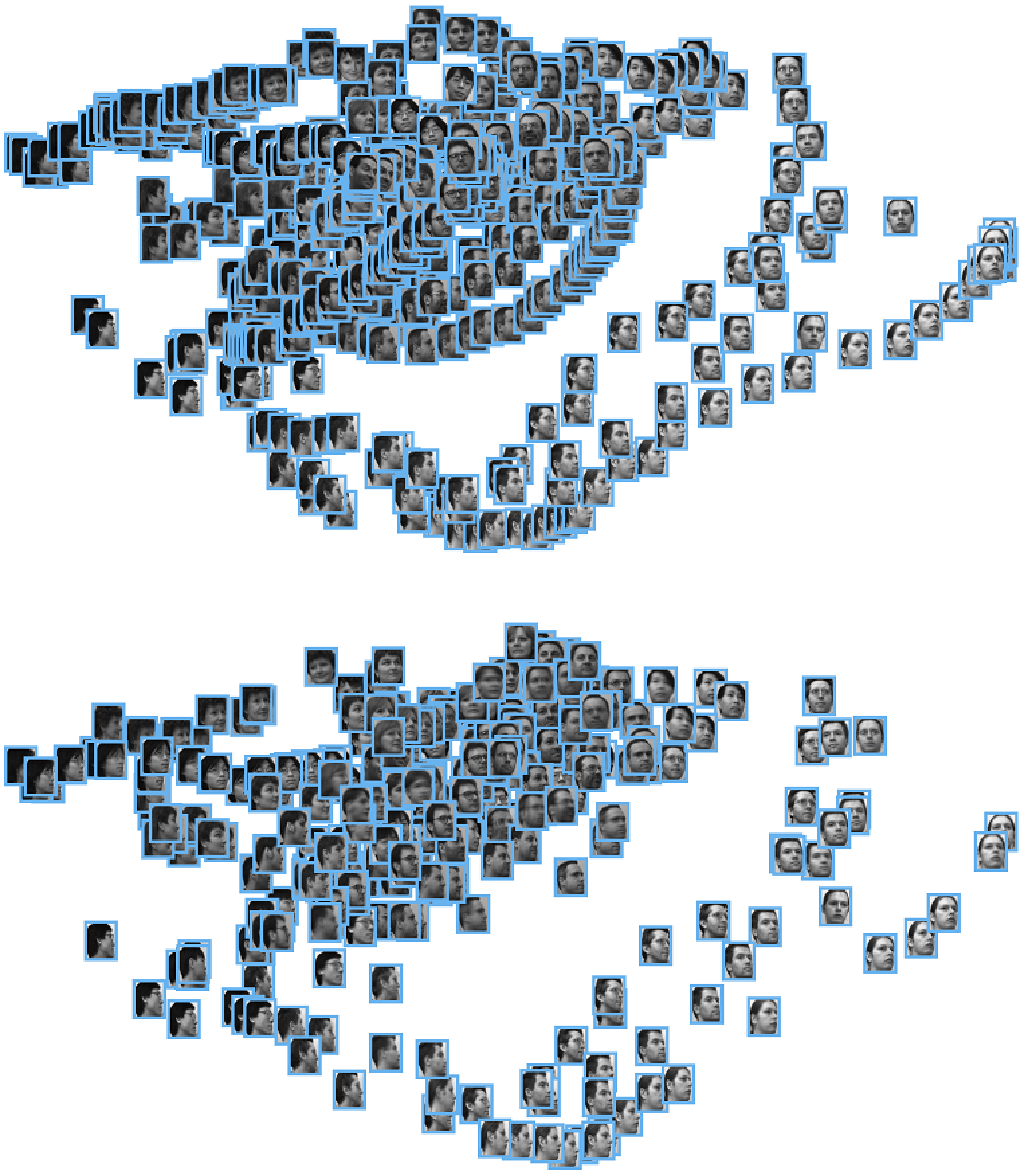


Figure 9.35: *A view of the first two dimensions in the dimension reduction of the UMIST data to four dimensions total. The top image is the original data, and the bottom the binned data.*

As was the case with the Frey data set, running *GapBin* on the UMIST data results in several large bins containing remarkably similar faces, as shown in Figure 9.36. There are also some bins which don't fare as well, such as those in Figure 9.37.

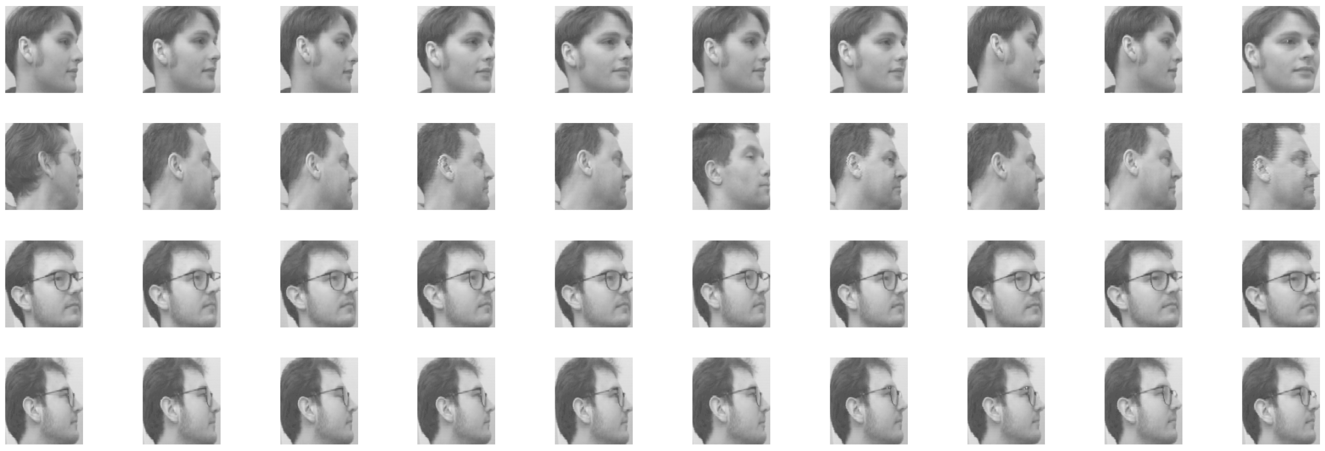


Figure 9.36: *Four different bins (one in each row), containing remarkably similar faces. Each bin contains ten faces which demonstrates the ability of the GapBin algorithm to continue to work in exceptionally high dimensions.*

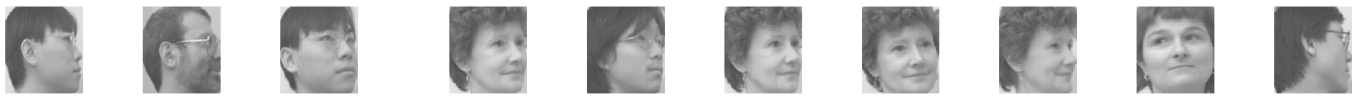


Figure 9.37: *Two bins (of size 3 and 7) in the binned configuration that contain objectively different faces.*

9.3.5 USPS Handwritten Digits

Finally, consider the USPS handwritten digits data set, which contains one thousand handwritten samples of the digits 0 - 9, resulting in a data set containing 11000 images. Each image is 16×16 greyscale, resulting in a dimensionality of 256. Using *GapBin*, the data is reduced to size 5500 (a 50% compression) in its native dimension, and is then reduced to four using multidimensional scaling.

Due to the sheer size of this data set, it's not feasible to look at the entire configuration with each representative image. So, only the alpha blended points are considered, and the bins created by *GapBin*. Figure 9.38 illustrates the original and binned configurations. As was the case in previous experiments, it is again clear that *GapBin* adequately decreases the size of the configuration in its native dimension while preserving much of the inherent structure. The binned configuration retains the characteristic shape of the original configuration, and simply thins out the areas of high density.

In some cases, this thinning out may be to severe. For example, the area around $(-1000, 250)$ seems to have been almost eliminated as compared to the original configuration. This, however, is consistent with what *GapBin* is designed to do - trade-off shape preservation for greater density retention.

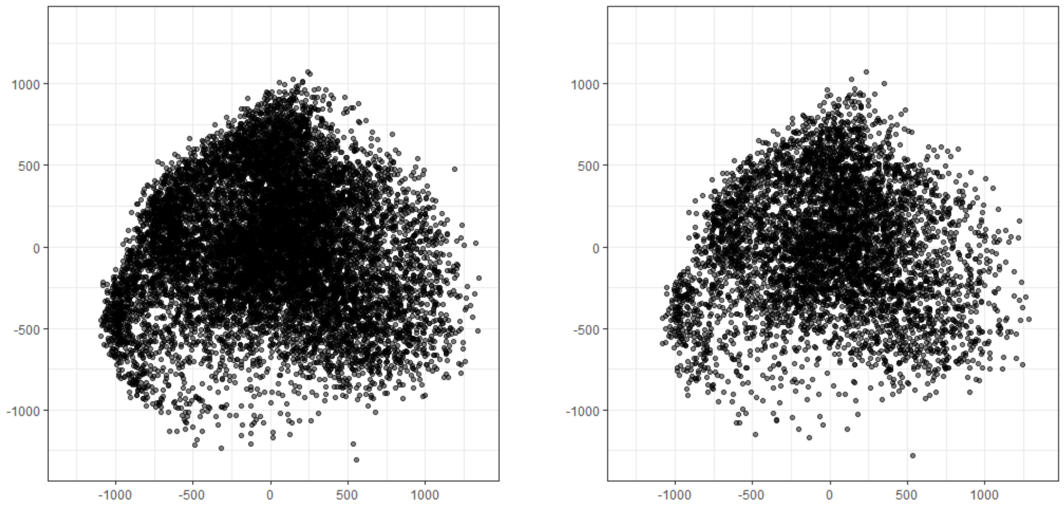


Figure 9.38: *A view of the first two dimensions in the dimension reduction of the USPS binary digits data to four dimensions total. The left image is the original data, and the right the binned data.*

Due to the structure of the USPS data, using *GapBin* results in many bins containing very similar images. Figure 9.39 illustrates bins for various digits, and Figure 9.40 the largest bin produced by *GapBin*.

The USPS handwritten data set is a great indication that *GapBin* is simultaneously capable of binning large data sets and binning in high dimensional space. This is something that none of the other algorithms considered thus far are capable of doing in a reasonable time.

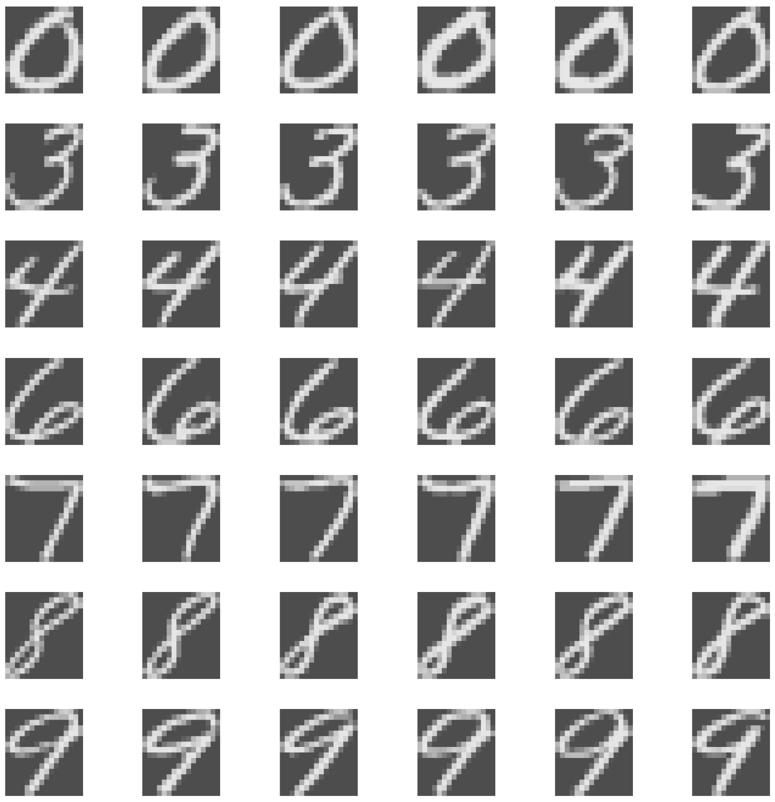


Figure 9.39: *Seven bins (one in each row) produced by GapBin containing different digits.*

Figure 9.40: *The largest bin (of size 20) produced by GapBin containing the digit 1.*

9.4 Concluding Remarks

This chapter studied the viability of binning with various well known binning algorithms (equal width, equal frequency, fixed frequency, hexagonal binning, and random sampling), as well as the newly proposed *GapBin* algorithm, which falls into the tree-based binning framework. A simulation study was undertaken in which configurations of varying dimension were binned down to 500 points from 1000, and then measured for various types of geometric and probabilistic structure. These measures were compared to those taken on the original configuration so the best performing algorithms could be identified.

Overall, the following trend was observed throughout the simulation experiment. The equal width and hexagonal binning algorithms performed very well on the measures designed to capture geometric structure, and poorly on those designed to capture probabilistic structure. The opposite was true for the equal frequency, fixed frequency, and random sampling algorithms. They performed well on the measures designed to capture probabilistic structure, and poorly on those designed to capture geometric structure.

The *GapBin* algorithm seemed to bridge the gap between these sets of "specialist" algorithms. It performed well on all measures, outperforming the "shape" specialists on the density measures, as well as the "density" specialists on the shape measures. As dimension increased, *GapBin* quickly became the best performing algorithm on all probabilistic measures, while often outperforming the equal width algorithm on the geometric measures. This led to the conclusion that, overall, *GapBin* was the best performing of the algorithms considered.

These results were corroborated using real data sets, including the diamonds and TOSCA high resolution data sets. The *GapBin* algorithm was also used exclusively to bin a number of image data sets, which exist in high dimensional space. Here, the goal was to bin the data in its original dimension, and compare it to the original manifold when brought down to three or four dimensions using multidimensional scaling. What was observed was that, even in very high dimensions, *GapBin* was able to accurately reduce the number of observations in the configurations, creating comparable manifolds as compared to the original.

Overall, the effectiveness of the *GapBin* algorithm was shown in multiple ways. It performed well in both the simulation study, and in real data sets, regardless of dimension. This is a positive result moving forward, as, for instance, it could potentially be used as the binning algorithm in a generalized scagnostic framework.

Chapter 10

Modifying the Scagnostic Framework

This chapter will look at various aspects of the current scagnostic framework, and propose solutions and improvements to some of the identified problems. Section 10.1 will propose solutions to the problems identified in the skewed and convex scagnostics. Specifically, it will look at the cause of the inability of these measures to obtain a full range of values in the $[0, 1]$, and propose a potential solution. Section 10.2 will propose two new scagnostics that identify interesting structure in a point configuration that is not currently captured. Finally, Section 10.3 will explore the generation of configurations with a given structure of interest.

10.1 Changes to Existing Scagnostics

In Section 3.1, issues with both the skewed and convex scagnostics were identified during the reproduction of the experiments of [120], and during the scagnostic universe experiment. This section will further explore the issues present with these scagnostics.

10.1.1 Convex and Skewed - The Issue with Weighting

The scagnostic universe experiment of Section 3.1 illustrated the inability of both the convex and skewed scagnostic measures to achieve the entire range of $[0, 1]$. This is problematic as it affects how these scagnostics are interpreted - if one end of the range is not achievable, are configurations that fall on the other end of interest, or not?

Focusing first on the skewed scagnostic, recall that the identified issue centred around the difficulty of finding configurations which score a low value on the scagnostic. The types of distributions needed to achieve certain values of the skewed scagnostic can be explored mathematically by writing the skewed formula as

$$\begin{aligned} q_{skew} &= \frac{q_{90} - q_{50}}{q_{90} - q_{10}} \\ &= \frac{q_{90} - q_{50}}{(q_{90} - q_{50}) + (q_{50} - q_{10})} \\ &= \frac{R}{R + L} \\ &= \frac{1}{1 + \frac{L}{R}} \end{aligned}$$

The fraction $\frac{L}{R}$ represents the relative tail weight of the left tail $(q_{50} - q_{10})$ as compared to the right tail $(q_{90} - q_{50})$. Recall that c_{skew} is written as

$$\begin{aligned} c_{skew} &= 1 - w\,(1 - q_{skew}) \\ &= 1 - w\left(1 - \frac{1}{1 + \frac{L}{R}}\right) \\ &= 1 - w\left(\frac{\frac{L}{R}}{1 + \frac{L}{R}}\right) \end{aligned}$$

where $w = 0.7 + \frac{0.3}{1+(n/500)^2}$. Analyzing the limit of c_{skew} as $\frac{L}{R} \to 0$, the value of $c_{skew} \to 1$, the upper limit of the scagnostic measure (which is a desirable trait). Looking at the limit as $\frac{L}{R} \to \infty$, $c_{skew} \to 1 - w$. The skewed scagnostic can not possibly approach 0 for any sample size (indeed, $w = 1$ only if $n = 0$), which is of course not desirable - the lower range of the skewed scagnostic cannot be obtained, regardless of the relative weight of the left tail to the right tail. This is concerning, as the left and right tail of the configuration are not treated symmetrically, which affects the way in which the scagnostic is interpreted. This also explains why so few low skewness configurations were observed in the scagnostics universe experiment in Section 3.1.

The weighting function w causes similar problems in the convex scagnostic. Recall that the convex scagnostic is given by

$$c_{convex} \;=\; w\,\left(\frac{area(A)}{area(H)}\right)$$

where A and H are the alpha and convex hull respectively. By definition, $area(A) \leq area(H)$, implying that the convex scagnostic falls in the range $[0, w]$, not [0,1]. This range was clearly visible in the scagnostics universe experiment performed in Section 3.1, where only six configurations had observed values of convex above 0.75. Again, the weighting function is not allowing the convex scagnostic to achieve a full range of values in $[0, 1]$, which hampers the ability of a user to accurately interpret the values emerging from a scagnostic analysis.

The question worth exploring then is if the perceived bias in the scagnostics which are weighted by w is worth the inability of the scagnostics to achieve the entire range of values available to the scagnostic measures. A simple analysis to examine the effects of binning on a configuration is to use the same configurations that were used in Section 3.1 for increasing sample size, and observing the difference between scagnostics computed on the original configuration and the binned configuration without adjusting any of the scagnostics for binning. Figure 10.1 presents the results.

As expected, many of the scagnostics do not exhibit any effect when the configuration is binned. The two that were expected to be problematic, skewed and convex, certainly do. The skewed scagnostic exhibits an upward trend, indicating that the observed skew in the binned configuration is smaller than in the original configuration. Starting at 1000 points, there are no observed differences less than 0, indicating that the binned configuration always has a skewed scagnostic value lower than the original, which is consistent with how the skewed scagnostic is weighted.

A similar issue arises in the analysis of the convex scagnostic - as n increases, the difference between the original and binned scagnostic values increases and eventually no difference exceeds 0, indicating that the convex scagnostic in the binned configuration is larger than in the original configuration. This is again consistent with the scaling applied to the convex scagnostic.

The next question to address then is if the scaling remedies the observed issues. Figure 10.2 illustrates the results of running the same experiment, but also applying the scaling function of [119].

The weighting function does appear to compensate for the issues observed in both the skewed scagnostic (with perhaps a slight over-compensation for higher values of n) and convex scagnostic. The boxplots no longer have a downward or upward trend as sample size increases. However, balanced against the downside of being unable to achieve the entire range of the scagnostic, it may be prudent to examine a different solution for these scagnostic values.

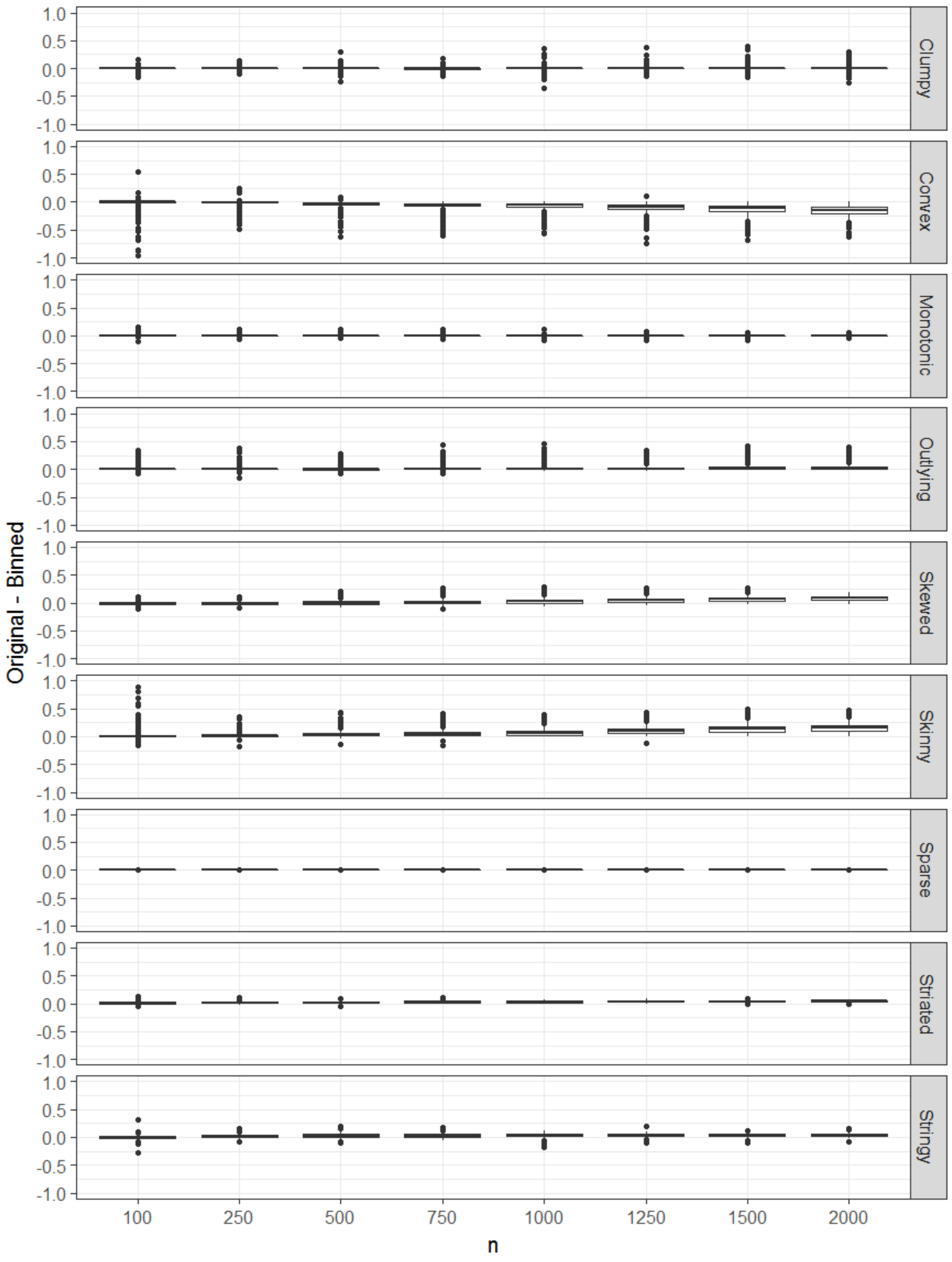


Figure 10.1: *The effect of binning on each of the nine scagnostic measures for point configurations of increasing size. Note that no correction has been applied to these measures.*

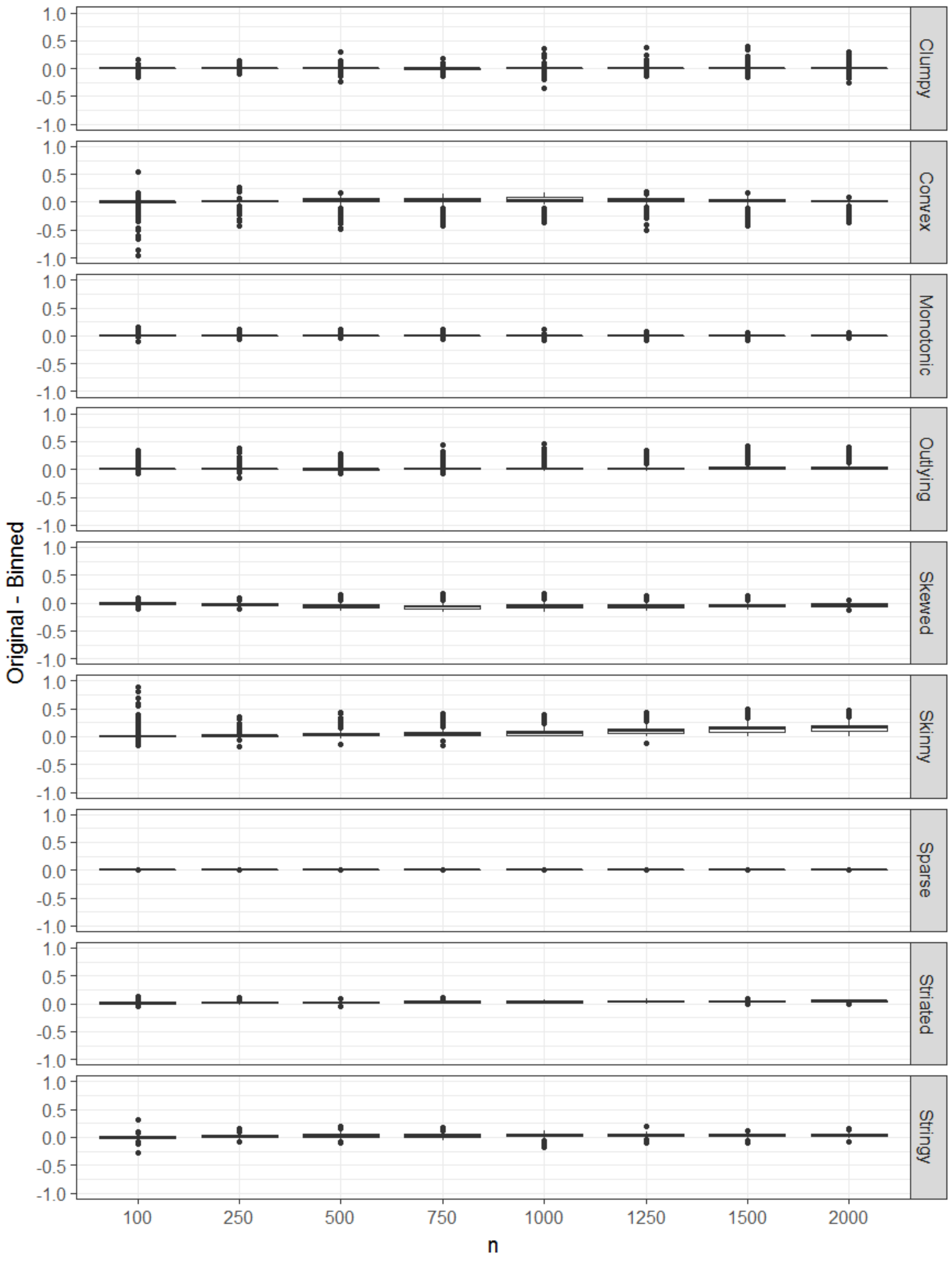
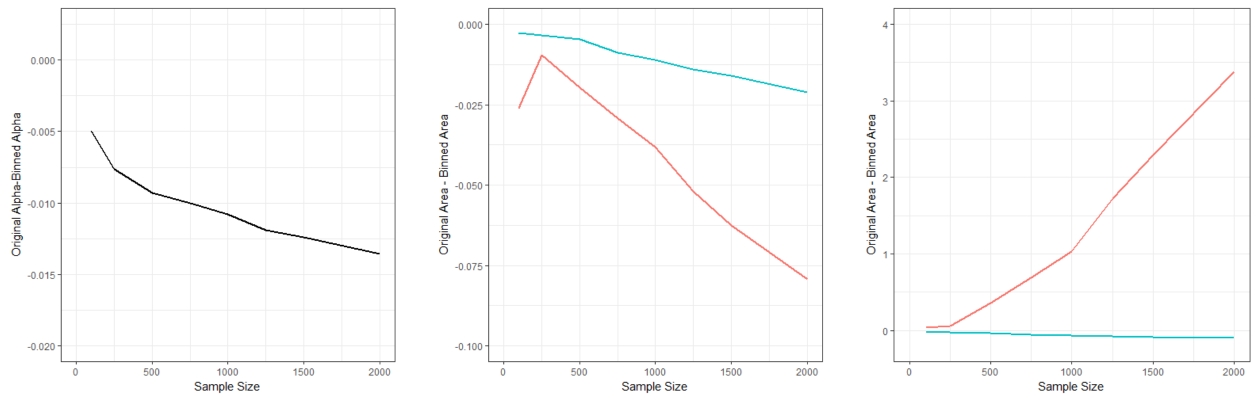


Figure 10.2: *The effect of binning on each of the nine scagnostic measures for point configurations of increasing size. Here, a correction has been applied to these measures where appropriate, as in [119].*

10.1.2 Proposing a New Alpha Value

First, consider the effect of sample size on the value of α used in the alpha hull. Recall that the α value is computed as the 90^{th} percentile of the minimal spanning tree edge length distribution in the current scagnostic framework. The value of α itself is not necessarily important, but its effect on the area and perimeter of the resulting alpha hull is. Figure 10.3 illustrates the difference between the α values used in the computation of the alpha hull in the unbinned configuration and the binned configuration as sample size increases, over various configuration types.



(a) (b) (c)

Figure 10.3: *The average difference over several configurations. (a) the value of α, (b) the area of the resulting alpha hull, (c) the perimeter of the resulting alpha hull. The alpha hull is in red, and the convex hull is in blue.*

As expected, as sample size increases, the difference between the α in the binned configuration and the unbinned configuration increases as well. This isn't necessarily problematic, nor is it particularly surprising, since binning tends to group nearby points together, eliminating small edges in the minimal spanning tree, while preserving most of the large edges. This leads to a larger 90^{th} percentile in the binned configuration as opposed to the unbinned configuration.

The difference in area and perimeter of the alpha and convex hulls between the original and binned configurations is ultimately what is important. Beginning with the convex hull, from Figure 10.3, the area and perimeter to not appear to be overly sensitive to binning. Indeed, the change in area and perimeter are due mostly to the removal of outliers, not necessarily binning. Next, from Figure 10.3 the difference in both the area and perimeter of the alpha hull increases significantly as sample size increases - although in opposite directions. The area of the binned configurations becomes larger as compared to the original, but the perimeter of the original configuration increases compared to the binned configuration. The cause of this is illustrated in Figure 10.4. Here, the alpha hull of the original configuration becomes increasingly overfit, with more interior holes. This results in the decreased area that is observed, as well as the increased perimeter.

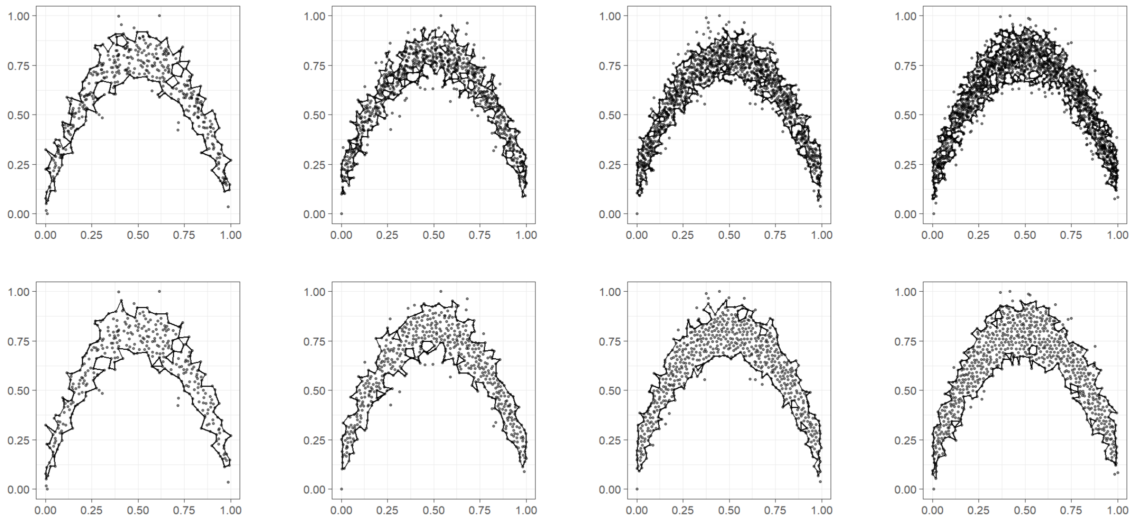


Figure 10.4: *Four quadratic configurations of size 500, 1000, 1500, and 2000. On top, the alpha hull on the original configuration, and on bottom, the alpha hulls on the configuration binned using the hexagonal binning approach of scagnostics.*

For any sample size, the current α seems to overfit the alpha hull to the configuration. This is especially evident in the original configuration, but also holds true for the binned configuration as well. Consider the following value of α, which is similar to the one proposed in [74]

$$\alpha = \sqrt{\sum_{i \in \Omega} \frac{e_i}{n}}$$

where n is the number of points in the configuration, and the sum is over all those minimal spanning tree edges in the set $\Omega = \{e : q_{25} \leq e \leq q_{75}\}$. That is, the sum is over all those edges that are in the middle 50% of the minimal spanning tree edge length distribution. Due to the square root, this α will tend to be larger than than the current value of α, but still decreases in n. Figure 10.5 illustrates the same configurations as in Figure 10.4, but with an alpha hull computed with the new proposal.

Over each of the sample sizes, the alpha hull between the two configurations is very similar - and is actually comparable to a hull (i.e. very few, if any, interior holes) unlike the result of the previous value of α. While it does include some extraneous points in the configuration, the resulting shape is (in this case) still clearly a parabola, thus the new value of α continues to capture the shape of the point configuration.

To confirm the the visual results of the experiment in Figure 10.5, consider repeating the experiment of Figure 10.3. Figure 10.6 presents the difference in α, alpha hull and convex hull

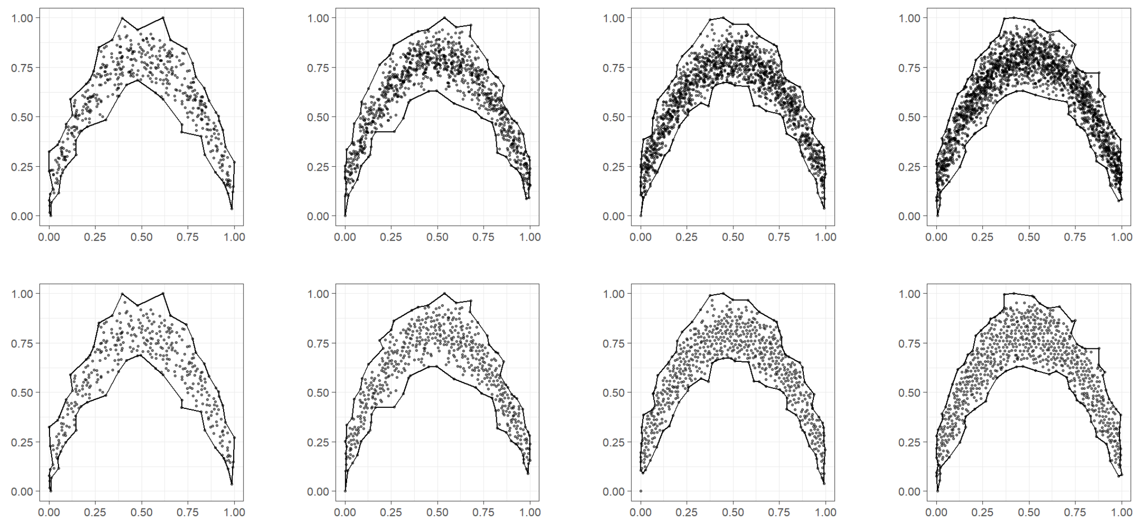


Figure 10.5: *The same configurations as in Figure 10.4, but with the alpha hull computed using the newly proposed alpha value. The resulting alpha hulls contain fewer holes and discontinuities, and represent a marked improvement over the previous proposal.*

area, and alpha hull and convex hull perimeter between the original and binned configurations as sample size increases, over varying configurations.

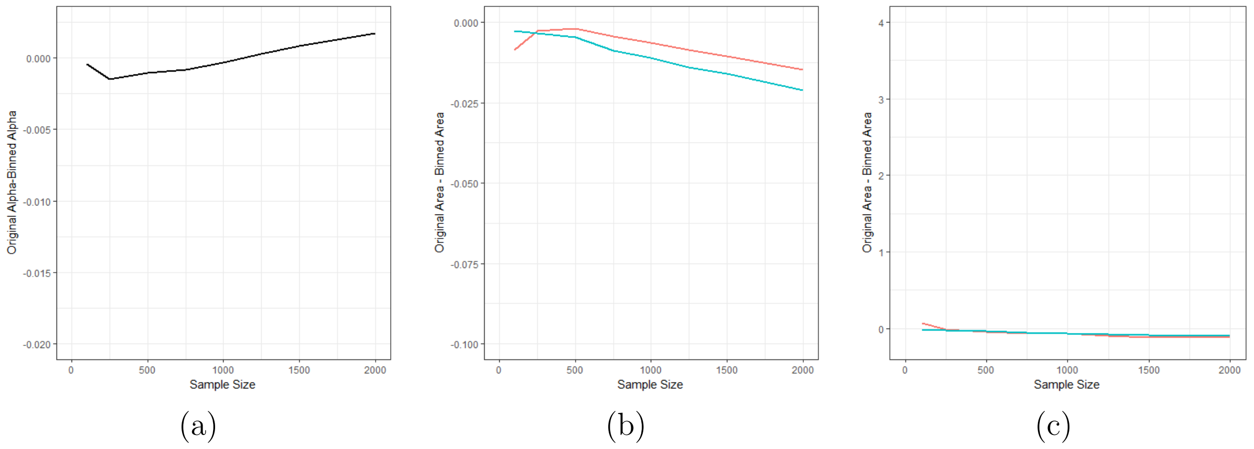


Figure 10.6: *The average difference using the newly proposed value of α over several configurations. (a) the value of α, (b) the area of the resulting alpha hull, (c) the perimeter of the resulting alpha hull. Note that these images have been paced on the same scale as Figure 10.3 so they can be compared directly. The alpha hull is in red, and the convex hull is in blue.*

Noting that each of the images in Figure 10.6 is on the same scale as those in Figure 10.3, the difference between the original and binned configurations has been greatly reduced using the newly proposed α. In fact, the convex hull and alpha hull seem to be effected in a similar fashion, decreasing at the same rate as the sample size increases.

The final step in confirming this result then, is to repeat the experiment of Figure 10.1, to see if the newly proposed α value eliminates the need for scaling in the convex scagnostic. The results are shown in Figure 10.7.

The newly proposed value of α appears to work as intended. Without scaling, the convex scagnostic varies much less with sample size. This eliminates the need for scaling, which allows the convex scagnostic to now achieve a full range of values in $[0, 1]$.

An Additional Benefit - Improving Skinny

While it has not been explicitly addressed thus far, the skinny scagnostic is one which provides some puzzling issues. In [120], the skinny scagnostic is introduced with the following description - "the ratio of perimeter to area of a polygon measures, roughly, how skinny it is. We use a corrected and normalized ratio so that a circle yields a value of 0, a square yields 0.12, and a skinny polygon yields a value near one." In contrast to this statement, however, is the performance of the skinny scagnostic on some "near square" configurations. Consider Figure 10.8(a), which illustrates the distribution of the skinny scagnostic over bivariate uniform distributions of size 500.

Using the current value of α, the skinny scagnostic does not seem to be working as described in [120]. A bivariate uniform distribution, which bears a striking resemblance to a "square", yields unexpectedly high values of skinny. All of the observed skinny values are above 0.5, and the average value over the experiment was 0.715. This is much too high for a configuration that surely should not be considered skinny.

Instead, consider using the newly proposed value of α. The distribution of the skinny scagnostic can be found in Figure 10.8(b). Using the newly proposed value of α, the skinny scagnostic now identifies the bivariate uniform appropriately - it is, in fact, not skinny. All of the observed values are below 0.25, and the average value of skinny over the experiment using the newly proposed α was 0.122 - almost exactly what should be expected given the definition of skinny provided in [120].

While the new value of α was not proposed with skinny in mind, it does help shore up one additional issue with the underlying scagnostic framework.

10.1.3 Tree-based Binning as a Pre-Processing Step

As an aside to the analysis presented in Figure 10.1, recall that one of the goals of Chapter 3 was to establish a solid framework for scagnostics in more than two dimensions. As was shown in Chapter 9, *GapBin* quickly became the best performing binning algorithm as dimension increased

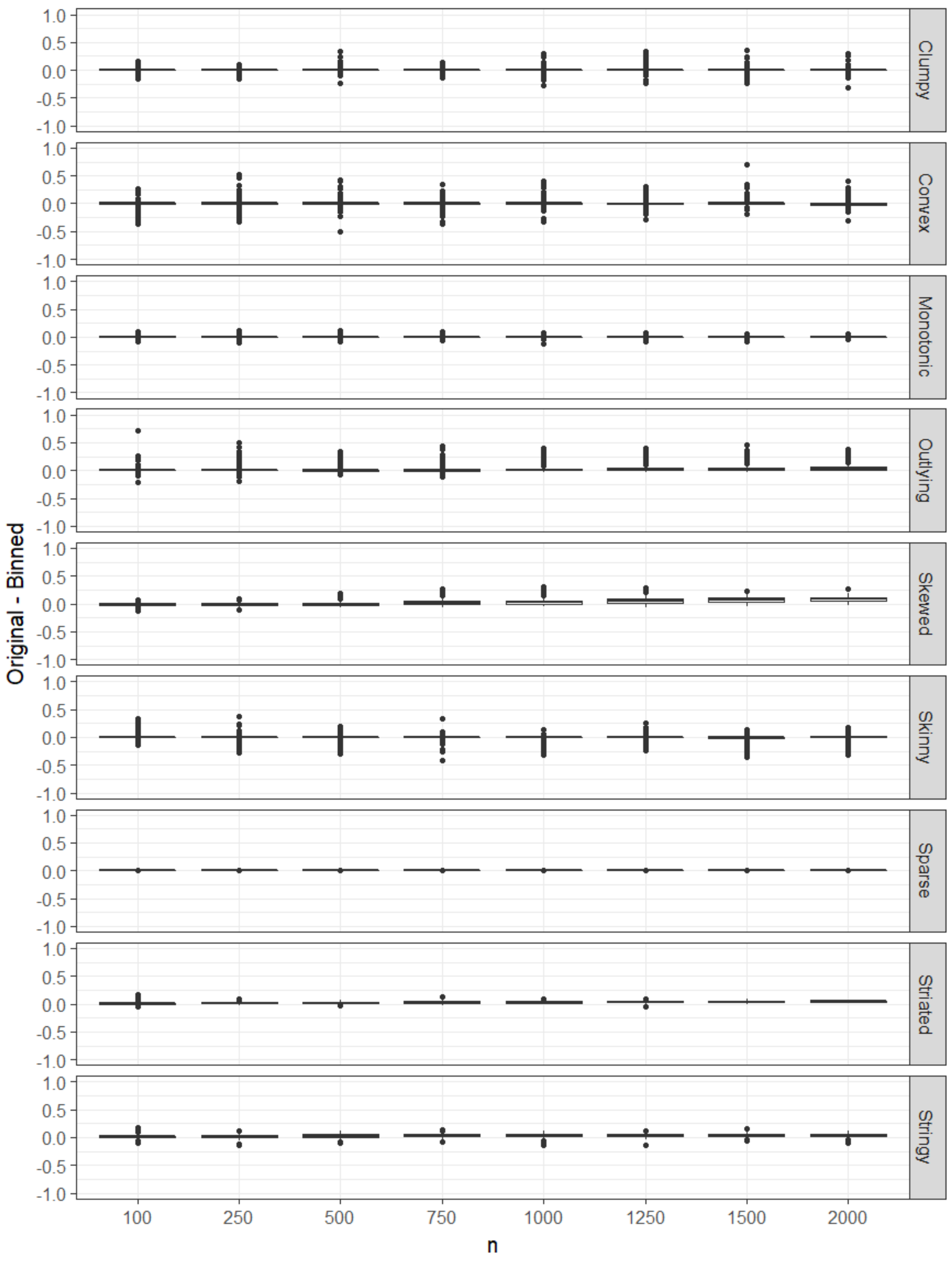
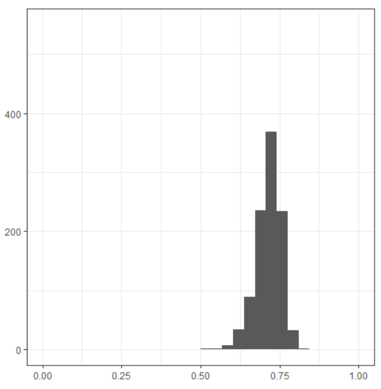
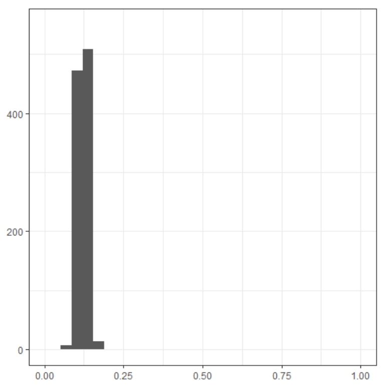


Figure 10.7: *The effect of binning on each of the nine scagnostic measures for point configurations of increasing size. Here, the newly proposed value of α is used.*



(a)



(b)

Figure 10.8: *(a) The distribution of the skinny scagnostic over 1000 simulated bivariate uniform configurations of size 500 using the existing α value in the computation of the alpha hull. (b) The distribution of the skinny scagnostic over the same configurations using the newly proposed α value.*

among those considered in terms of retention of probabilistic and geometric structure. It would be beneficial then to explore the effects of this algorithm on the current scagnostic framework.

The exact same experiment from Figure 10.1 is performed using tree-based binning in place of hexagonal binning. To ensure the two methods can be compared, the data are binned to the exact same number of points using tree-based binning as were achieved during hexagonal binning. The results are shown in Figure 10.9.

Of note is the similarity between Figure 10.9 as compared to Figure 10.7. Many of the scagnostics are similar, regardless of binning method used - for instance, sparse, outlying, striated, stringy, and monotonic seem to have similar distributions regardless of binning method used.

Next, consider the two problematic scagnostic measures, convex and skewed. Using tree-based binning with the newly proposed value of α appears to have had the same effect on the convex scagnostic as using hexagonal binning - namely that binning now has minimal affect on the convex scagnostic as n increases.

The skewed scagnostic however, tells a slightly different story. While there is a clear upward trend in the skewed scagnostic as n increases using hexagonal binning, the same trend is much more subtle when tree-based binning is used. In fact, under tree based binning, the median difference between the binned and non-binned configurations is only 0.014 (under hexagonal binning, the same difference is 0.09!).

This result is actually corroborated by what was observed in the experiments of Chapter 9. Recall that the tree-based binning algorithm better preserved the density of the point configuration - keeping dense areas relatively dense in the binned configuration. This in turn preserves small

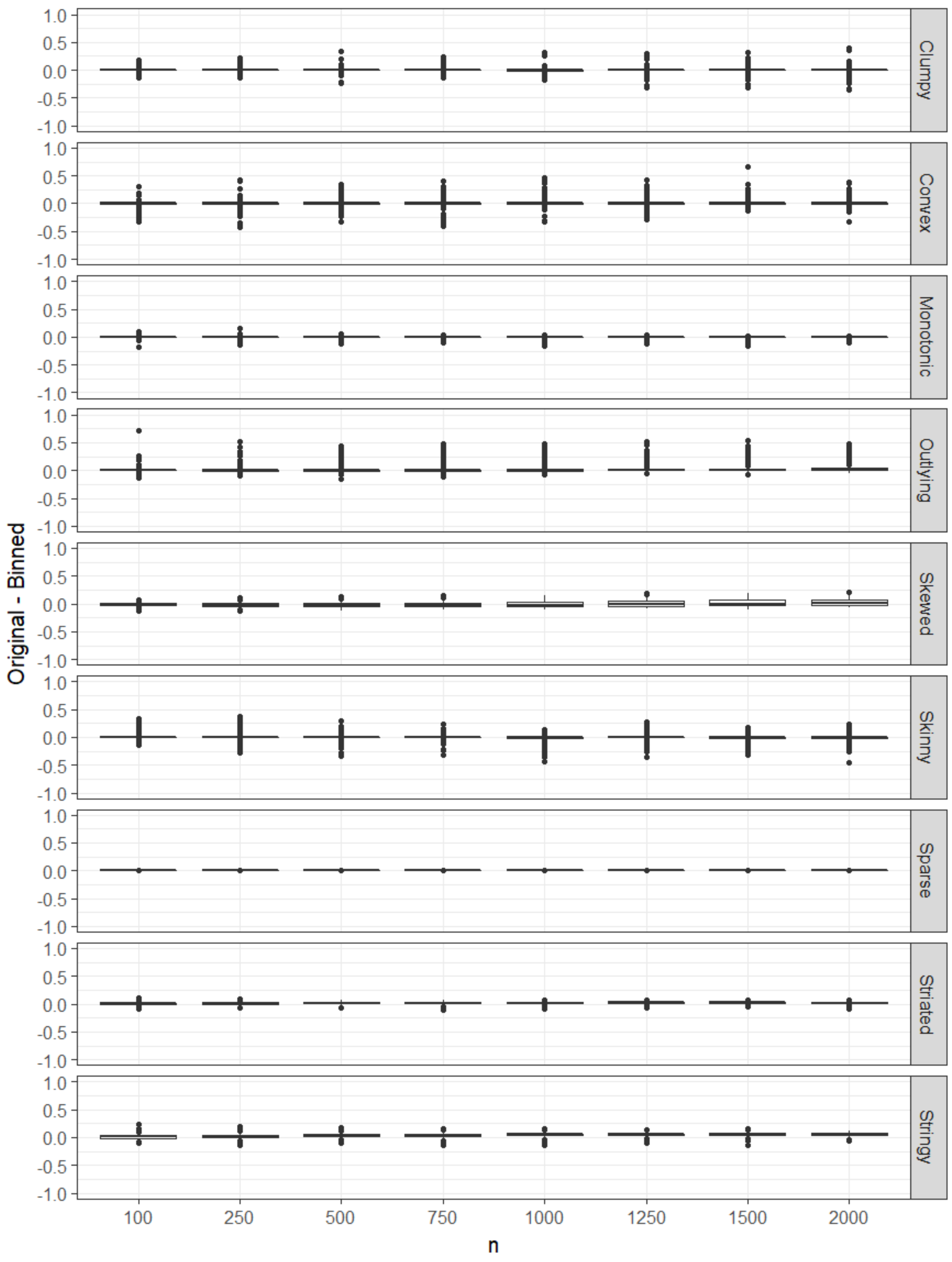


Figure 10.9: *The effect of tree-based binning on each of the nine scagnostic measures for point configurations of increasing size. Note that no correction has been applied to these measures, and the newly proposed value of α has been used.*

edge lengths in the minimal spanning tree, which increases the weight of the left tail to the right, increasing the skewed scagnostic of the binned configuration.

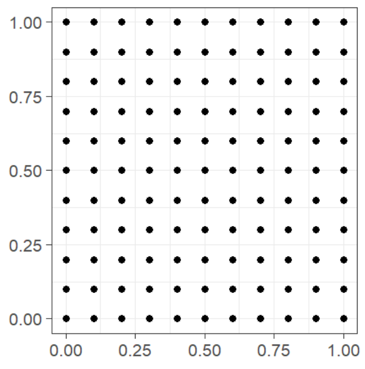
In total then, nothing is lost by converting from hexagonal binning to tree-based binning in the scagnostic framework, and in fact the performance of the skewed scagnostic is significantly improved, essentially eliminating the need to scale the skewed scagnostic. This switch to tree-based binning further progresses the underlying scagnostic framework to supporting scagnostics in higher dimensions.

10.2 New Measures

While the original nine scagnostics do capture a wide array of interesting structure, there is plenty of other interesting structure that they do not distinguish between. This section will propose two new scagnostics, *grid* and *symmetry*, and also discuss why these scagnostics are of interest by examining common data configurations that exhibit the qualities measured.

10.2.1 The Grid Scagnostic

Consider a set of points lying on a perfect grid, such as the one in Figure 10.10. The scagnostics computed on various grid sizes are also shown in Figure 10.10.



Scagnostic	5x5	10x10	20x20
Outlying	0.000	0.000	0.003
Skewed	0.499	0.280	0.534
Clumpy	0.002	0.003	0.002
Sparse	0.201	0.100	0.043
Striated	0.029	0.008	0.011
Convex	0.000	0.098	0.086
Skinny	1.000	0.115	0.110
Stringy	0.356	0.327	0.289
Monotonic	0.000	0.000	0.000

Figure 10.10: *An example of what would be considered a perfect grid - the measured angle between adjacent (and nearest neighbour) points being either 90 or 180 degrees. Scagnostics are computed on 5x5, 10x10, and 20x20 grids.*

Looking at the scagnostic measures, none of them appear to be indicative of a point configuration that contains any type of structure - let alone the rigid structure of a perfect grid. Consider then, a new scagnostic aimed to identify grid and near-grid structures in a point configuration.

A key structure of a grid is the presence of right angles and straight lines. As such, consider a scagnostic that measures the number of 90 and 180 degree angles formed by a point and its two nearest neighbours.

$$c_{grid} = \frac{\sum_{i=1}^{n} I(\theta_{e_{ij}e_{ik}} = \{90, 180\})}{n}$$

where j and k are the nearest neighbours to node i, e_{ij}, e_{ik} are the node length between them, and n is the number of points in the configuration. Note that this formulation will allow for a grid like structure to be identified even if it is not orthogonal to the x-y axis. Clearly, simply considering perfectly formed 90 and 180 degree angles will only capture perfect grids. It is not unreasonable to want to capture grid-like structures as well. As such, the scagnostic above is modified to count all angles that are within some *range* (call it α) of 90 and 180 degrees

$$c_{grid} = \frac{\sum_{i=1}^{n} I(\theta_{e_{ij}e_{ik}} \in \{90 \pm \alpha, 180 - \alpha\})}{n}$$

To test the grid scagnostic, consider computing it on a series of plots beginning with a perfect grid and jittering it by noise of increasing variability. For the next example, consider jittering the perfect grid by Uniform[-a,a] random variable, where the variable a will take on the values

$$a = (.01, .02, .03, .04, .05, .06, .07, .08, .09, .1)$$

To conduct the experiment, each configuration is computed 100 times, and the grid scagnostic is calculated with $\alpha = 20$ for each. As a point of comparison, a uniform sample from the unit square is also provided. Figure 10.11 illustrates the result of this experiment.

As the perfect grid is jittered by more variable noise, the grid scagnostic tends towards the uniform distribution, which exhibits a low value of the scagnostic. Note that the value of $\alpha = 20$ used in this experiment is relatively generous. Using a more conservative α value may lead to a grid scagnostic that tends to zero faster than what was observed.

In real world data, what types of data conform to a grid, or grid like structure, that would be considered interesting (or not!)? Running the grid scagnostic over the "scagnostic universe" data used in Chapter 2, Figure 10.12 illustrates some of the images that score highest on the scagnostic.

The data are all, in fact, categorical data. In many applications, identifying data that are categorical could be of interest. Alternatively, it may beneficial to be able to ignore categorical data in the search of more interesting configurations. In either case, the grid scagnostic provides flexibility to the researcher in their analysis.

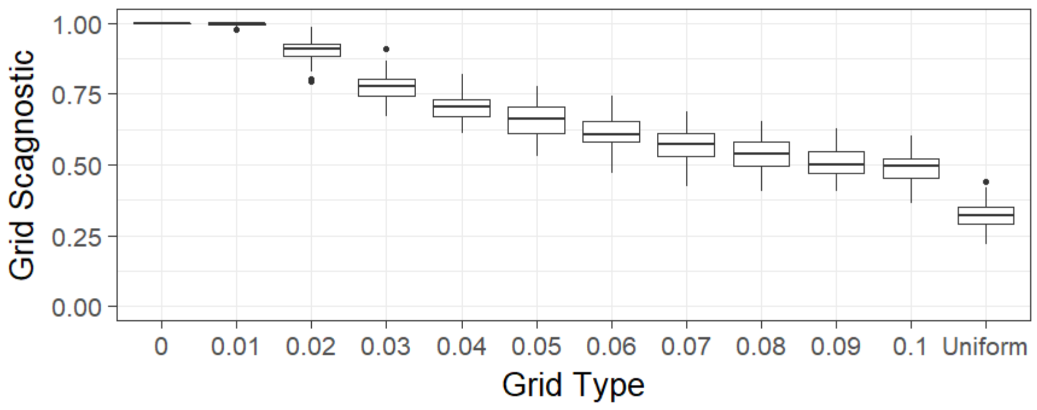


Figure 10.11: *The grid scagnostic computed on a series of grids that are increasingly jittered from a perfect grid.*

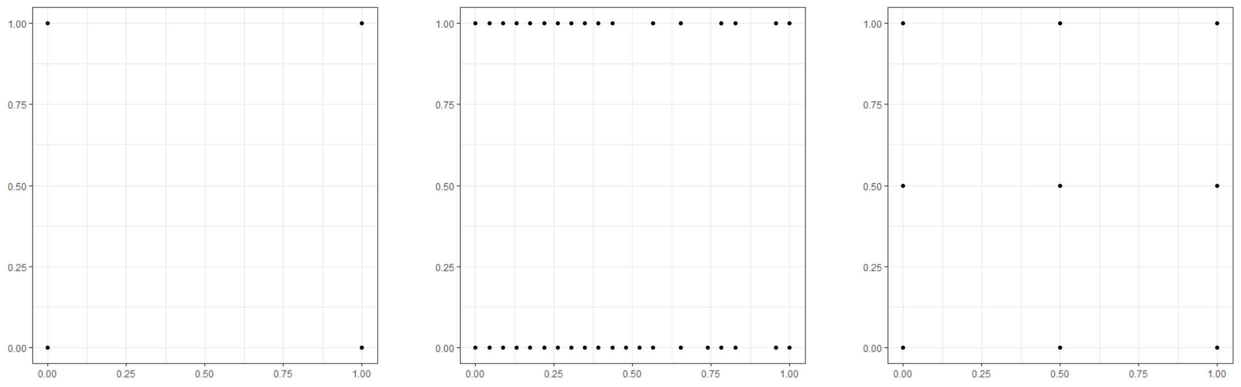


Figure 10.12: *Configurations from the scagnostic universe data set that score highly on the grid scagnostic. Unsurprisingly, these data represent categorical data.*

10.2.2 The Symmetry Scagnostic

In Section 3.3, the inability of the current scagnostic framework to detect the difference between perfectly symmetric and perfectly asymmetric point configurations was briefly explored. For reference, both the perfectly symmetric and asymmetric configurations are reproduced in Figure 10.13. Note that the data are scaled to be in the unit square.

The symmetry scagnostic differs from the typical scagnostic measures seen thus far as it does not rely on a geometric graph object. Instead, it takes the form of a formal test of hypothesis

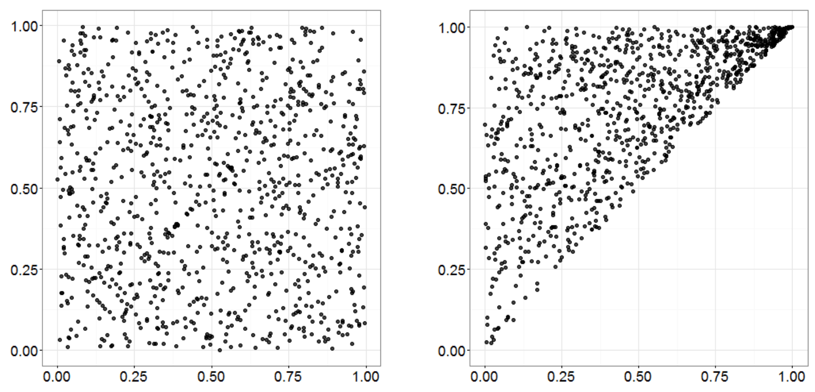


Figure 10.13: *An example of a perfectly symmetric and perfectly asymmetric configuration (along the main diagonal).*

$$H_0: \text{ The data are symmetric}$$
$$H_A: \text{ The data are not symmetric}$$

Note that symmetry can be tested both on the main diagonal (along the line $y = x$), as well as along the off diagonal (along the line $y = 1 - x$), illustrated in Figure 10.14.

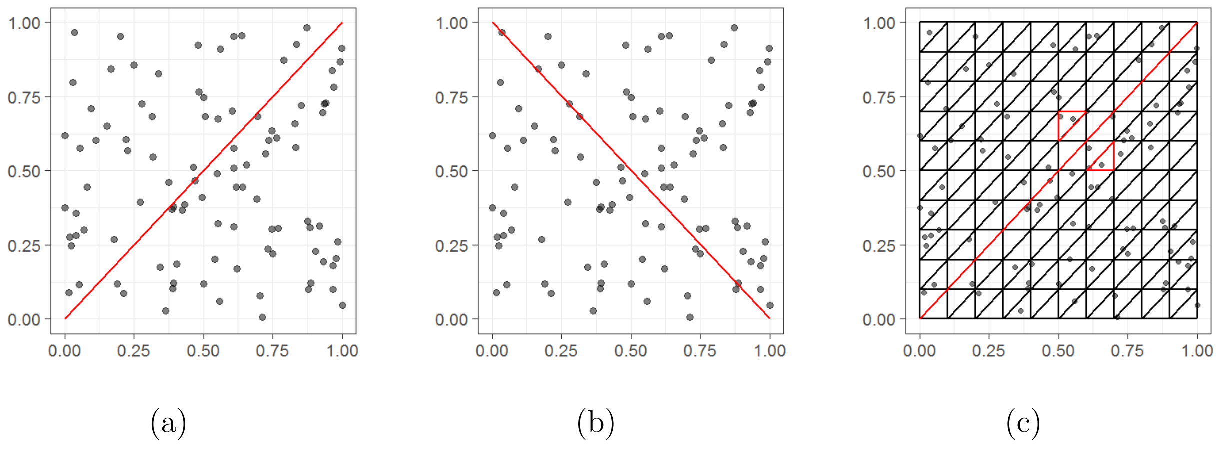


Figure 10.14: *(a) Illustration of main diagonal symmetry. (b) Illustration of off diagonal symmetry. (c) An example showing how triangles are compared when main diagonal symmetry is of interest.*

For the sake of clarity, the symmetry scagnostic will be discussed in terms of the main diagonal only, with the off diagonal being completely analogous. The hypothesis of symmetry is tested by first subdividing the data into equal sized triangles, and then comparing triangles that would overlap if the data were "folded" over the main diagonal. Figure 10.14 illustrates these concepts. Here, the main diagonal is highlighted red, and the two highlighted triangles would be compared during the computation of the scagnostic.

A Chi-square test can be performed to test the proposed hypothesis. Under the null hypothesis (being the data are perfectly symmetric), the expectation is that the lower half of the main diagonal would contain the same number of points as the upper half. Then, the Chi-square test statistic can be written as:

$$\begin{aligned} x &= \sum_{i=1}^{p} \frac{(\text{Observed} - \text{Expected})^2}{\text{Expected}} \\ &= \sum_{i=1}^{p} \frac{((u_i + l_i) - (2u_i))^2}{2u_i} \\ &= \sum_{i=1}^{p} \frac{(u_i + l_i - 2u_i)^2}{2u_i} \\ &= \sum_{i=1}^{p} \frac{(l_i - u_i)^2}{2u_i} \end{aligned}$$

where p is the number of triangles in the upper half of the configuration, and u_i and l_i are the number of points in the i^{th} upper triangle and the corresponding lower triangle. Considering only the triangles that actively contain points (so as to not inflate the test statistic), the degrees of freedom for this test can be computed as

$$\begin{aligned} q &= \text{d.f. under full model} - \text{d.f. under reduced model} - \text{number of empty triangle pairs} \\ &= (\text{Total Number of Triangles} - 1) - (\text{Number of Triangles above main diagonal - 1}) \\ &\quad - \text{number of empty triangle pairs} \end{aligned}$$

The test statistic, and therefore the symmetry measure, is then

$$c_{symmetry} = 1 - P(\mathcal{X}_q^2 \leq x)$$

As a proof of concept, consider computing the newly proposed symmetry scagnostic on both the perfectly symmetric and asymmetric configurations in Figure 10.13. Computed along the main diagonal, the symmetry measure for the perfectly symmetric configuration is 1.00, while for the asymmetric configuration it is 0.00. Along the off-diagonal, the symmetry measures are 0.898 and 0.000 respectively, which is to be expected given the configurations.

As a simple application, consider evaluating the symmetry of a pair of (standardized) stock returns along the main diagonal. As mentioned previously, this is important in applications such as fitting a symmetric copula to the data [30]. Figure 10.15 illustrates four examples that illustrate varying levels of symmetry in the data.

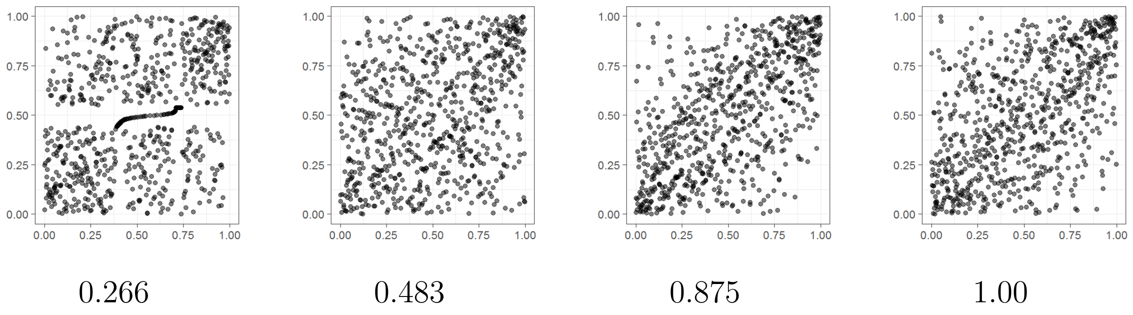


Figure 10.15: *Four examples of bivariate stock data representing varying levels of the newly proposed symmetry scagnostic. The calculated value of the newly proposed symmetry scagnostic is shown underneath each image.*

The symmetry scagnostic performs well overall, correctly capturing increasing symmetry in the configurations. The measure seems to be sensitive to clusters of points, so symmetry close to the main diagonal seems to play a pivotal role (in these examples) in labeling the configurations as being symmetric.

10.3 Configurations from MST Edge Length Distributions

In Section 4.4, the idea of generating a configuration from a simulated minimal spanning tree was proposed. Recall that the minimal spanning tree controls several scagnostic measures, as it determines the degree of each node (controlling the *stringy* scagnostic), the size of the largest edges (controlling both the *outlying* and *sparse* scagnostics), the RUNT dendrogram (controlling the *clumpy* scagnostic), and the distribution of the edge lengths in the minimal spanning tree can be simulated directly (controlling *skewness*). In addition, the guided random search algorithm proposed in Section 4.3 directly controls the angles between minimal spanning tree nodes, which indirectly controls the *striated* scagnostic.

The guided random search algorithm generates a point configuration via the edges of a minimal spanning tree. To create a configuration with a given scagnostic value then, a minimal spanning tree with the desired edge length distribution must first be simulated. This will be the topic of Section 10.3.2-10.3.4, where simulation of the required minimal spanning tree edge length distribution is discussed.

10.3.1 Generating a General Minimal Spanning Tree

The generality of the guided random search algorithm allows a configuration to be computed for any randomly simulated minimal spanning tree, with one slight adjustment. Once the distances of a minimal spanning tree have been simulated, in addition to proposing the angle at which the point will be placed from the parent node, the parent node also needs to be proposed. This is done easily by simply choosing the parent node from the nodes already placed in the configuration.

The only restriction to simulating a minimal spanning tree is to ensure that all distances are non-negative. To generate a configuration containing n observations, one need only simulate $n-1$ non-negative distances. For instance, consider generating 99 minimal spanning tree edges (resulting in a configuration with 100 points), from an Exponential(1) distribution (histogram of edge lengths can be found in Figure 10.16). Using these edge lengths and the slightly modified guided random search algorithm, Figure 10.16 illustrates a configuration with the simulated minimal spanning tree.

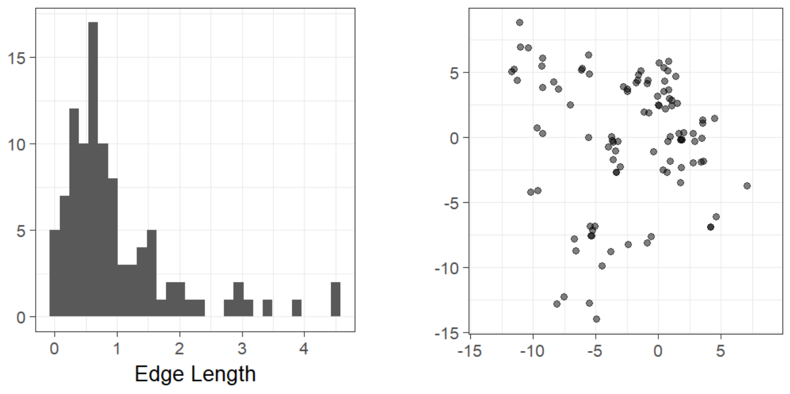


Figure 10.16: *A minimal spanning tree edge length distribution and the corresponding configuration created using the modified guided random search algorithm.*

10.3.2 Controlling Skewness

Recall that the (raw) skewed scagnostic is computed as

$$q_{skew} = \frac{q_{90} - q_{50}}{q_{90} - q_{10}}$$

implying that edge length distributions with heavy right tails (i.e. $q_{90} >> q_{50}$ and $q_{90} >>> q_{10}$) will have a large value of the skewed scagnostic. For example, Figure 10.17 illustrates a heavily right skewed edge length distribution (with a skewed value of 0.85), and a configuration created using the guided random search algorithm. As has been noted previously, these configurations don't appear to be particularly interesting.

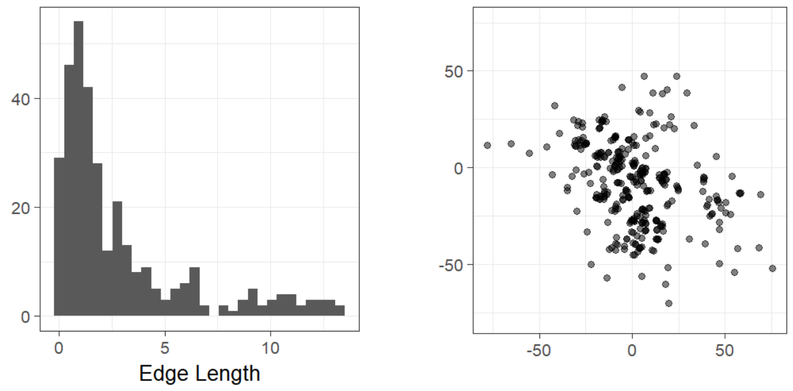


Figure 10.17: *A minimal spanning tree edge length distribution and the corresponding configuration for a high skew configuration created using the modified guided random search algorithm.*

The original motivation for the guided random search algorithm was to produce configurations with a low value of the skewed scagnostic. This would equate to minimal spanning tree edge length distributions that are heavily *left* skewed (i.e. $q_{90} \approx q_{50}$). Figure 10.18 provides an example of such an edge length distribution (with a raw skewed value 0.15), as well as a configuration created using the guided random search algorithm.

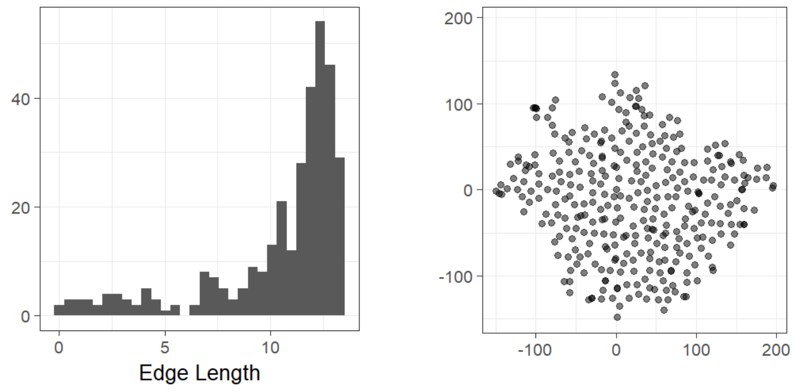
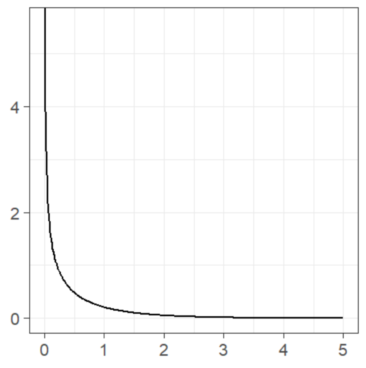


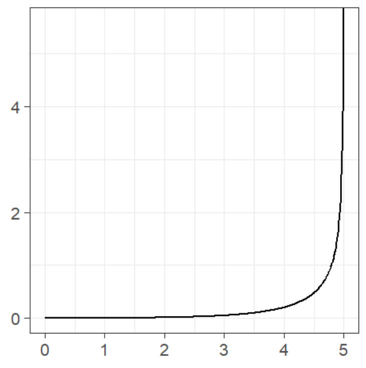
Figure 10.18: *A minimal spanning tree edge length distribution and the corresponding configuration created for a low skew edge length distribution using the modified guided random search algorithm.*

This low skewness configuration looks quite *regular*, in that the points seem to be very evenly spread. That is, there is very little overstriking occurring in the configuration. If it is compared to a uniform configuration, the regularity of the configuration may be worth noting should it occur in a data set - implying that low values of skewness may indeed be interesting and worth investigating should they occur.

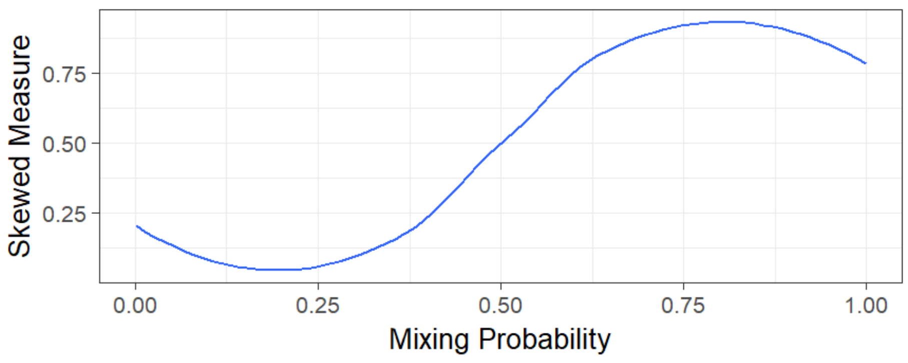
Using a mixture of probability distributions, an edge length distribution with a specific (raw) skewed value can be simulated. Beginning with the two parent distributions in Figure 10.19, with skewed values of 0.84 and 0.15 respectively, configurations with a raw skewed value in [.05,.93] can be achieved by creating minimal spanning tree edge length distributions as a mixture of the two parent distributions, as illustrated in Figure 10.19. This is, of course, just one example of a mixture distribution that could be used to create edge length distributions with varying levels of skewness - more extreme distributions would lead to more extreme ranges of the skewed distribution.



(a)



(b)



(c)

Figure 10.19: *(a) & (b) The two parent distributions mixed to achieve varying values of the skewed scagnostic. (c) The varying levels of skewness achieved by mixing the two parent distributions.*

10.3.3 Controlling Outlying

The outlying scagnostic is computed as a percentage of those minimal spanning tree edge lengths flagged as outliers. Recall that an outlier is any edge length greater than

$$q_{75} + 1.5 * (q_{75} - q_{25})$$

Figure 10.20 illustrates the edge length distribution and configuration produced using the guided random search algorithm for a configuration with a high value of outlying. As was pointed out earlier, configurations with a high outlying scagnostic tend to be interesting - points that deviate significantly from the overall trend in a configuration tend to warrant closer inspection.

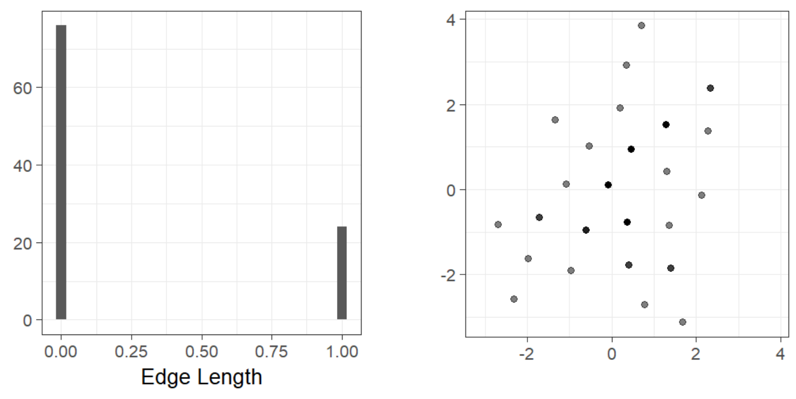


Figure 10.20: *A minimal spanning tree edge length distribution and configuration produced using the modified guided random search algorithm for a configuration with an outlying value of 1.*

Conversely, a low value of outlying can be easily achieved by having a relatively flat edge length distribution. For instance, the distribution in Figure 10.21 (and corresponding configuration) yields an outlying value of 0.

10.3.4 Controlling Stringy

Stringy is one of the few scagnostics that can be controlled that does not depend on the distances in the minimal spanning tree. Instead, it is based on the degree of the nodes in the minimal spanning tree

$$c_{stringy} = \frac{|V^2|}{|V| - |V^1|}$$

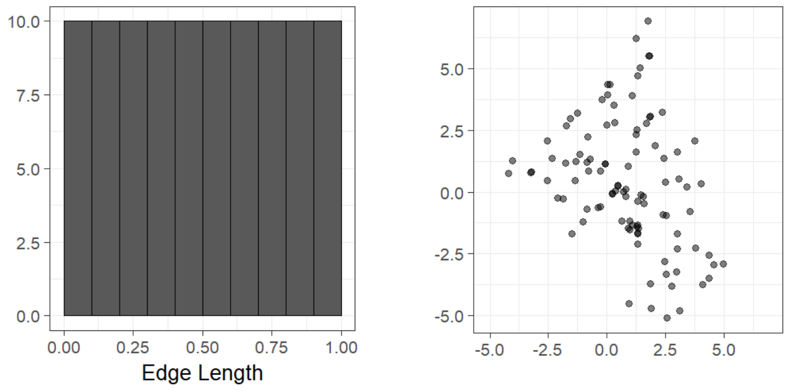


Figure 10.21: *A minimal spanning tree edge length distribution and configuration produced using the modified guided random search algorithm for a configuration with an outlying value of 0.00.*

The way in which a configuration is created to have a specific level of stringy is by measuring the current stringy value based on the points already placed in the configuration, and then executing the following simple decision when placing the next point

```
IF Current Stringy - Desired Stringy > tolerance
    Decrease stringy by choosing new parent from nodes with degree = 2
ELSE IF Current Stringy > Desired Stringy
    Choose new parent randomly from placed nodes
ELSE
    Increase stringy by choosing new parent from nodes with degree = 1
END
```

The only way to actively increase the stringy scagnostic is by turning a node of degree one into a node of degree two (which simultaneously increases both the numerator and denominator of the stringy scagnostic). This is done if the current measure of stringy is below the desired level of stringy. Stringy decreases if a node of degree two or more is increased to a node of higher degree. This is done if the difference between current stringy and desired stringy is beyond a certain threshold. Finally, by randomly choosing a node if the current stringiness of the configuration is close to that desired, the configuration is allowed to evolve naturally, while only curating the configuration if stringiness gets two far away from the desired target.

The configurations in Figure 10.22 show randomly generated configurations with varying levels of stringy.

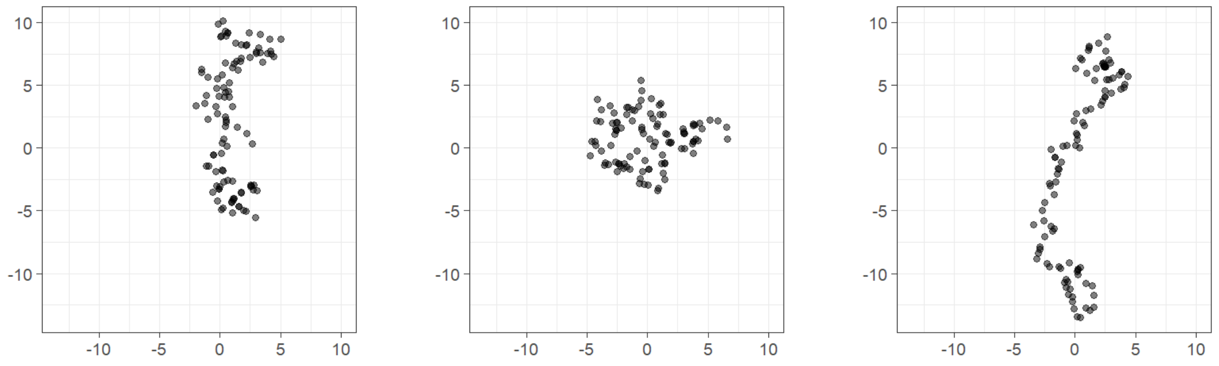


Figure 10.22: *Three configurations created with the guided random search algorithm with increasing levels of stringiness.*

10.3.5 Configurations using Restricted Angle Subsets

Recall that when placing a new point in the configuration, the modified guided random search algorithm proposes a parent node, and then proposes an angle (i.e. a direction on the unit n-sphere) for the position of this point. A simple way to create interesting structure then, is to restrict the subset of angles that can be proposed.

For instance, instead of proposing angles in $[0, 2\pi]$, one could restrict the proposed angles to be in $[0, \frac{\pi}{2}]$, which would lead to configurations such as those in Figure 10.23.

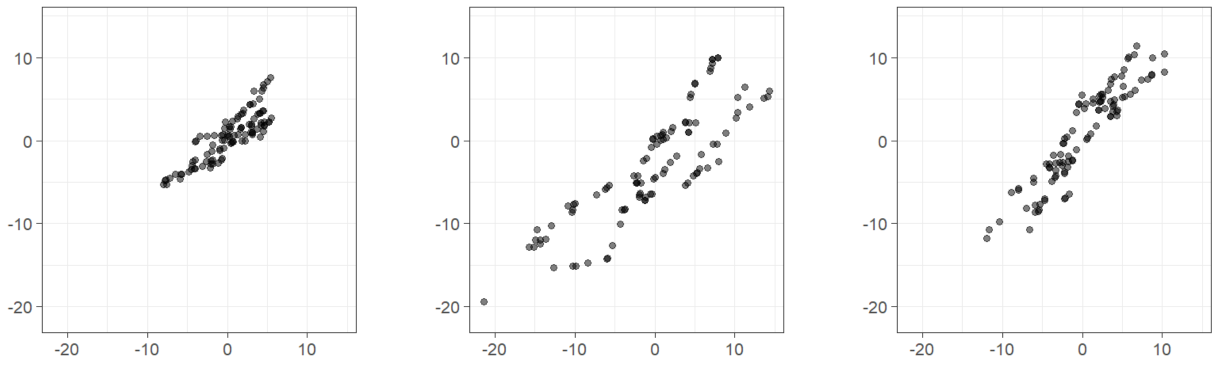


Figure 10.23: *Three configurations created with the guided random search algorithm with an angle set restricted to* $[0, \frac{\pi}{2}]$*. Note that varying the underlying edge length distribution results in varying ranges in the configurations.*

Instead of fully restricting the subset, instead consider a mixture of distributions. As an example, consider generating angles uniformly over $[\frac{\pi}{4}, \frac{\pi}{3}]$ 67% of the time, and from uniformly over $[0, 2\pi]$ 33% of the time. Figure 10.24 illustrates three examples.

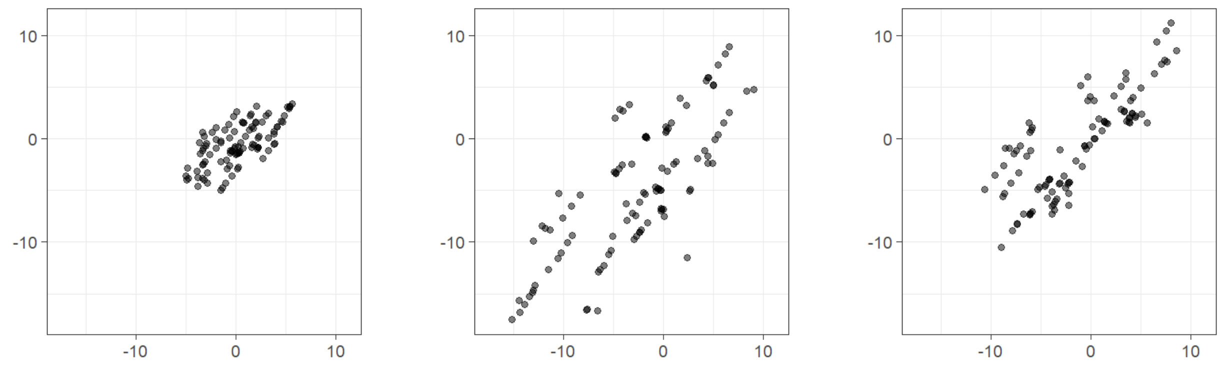


Figure 10.24: *Three configurations created with the guided random search algorithm with angle sets restricted to $[\frac{\pi}{4}, \frac{\pi}{3}]$ 67% of the time, and $[0, 2\pi]$ 33% of the time. Note that varying the underlying edge length distribution results in varying ranges in the configurations.*

This could be easily extended to any distribution of angles. The interest in controlling angles is that it indirectly controls certain scagnostic measures. Consider the simple experiment of increasingly restricting the angle available for the guided random search algorithm by increasing a value of α such that the angles available include $[\alpha, \frac{3\pi}{4} - \alpha]$. That is, the subset of angles is continually restricted to a smaller and smaller subset, forcing the guided random search algorithm to create straight lines in the data set, theoretically increasing the amount of measured striation. The results of such an experiment are shown in Figure 10.25.

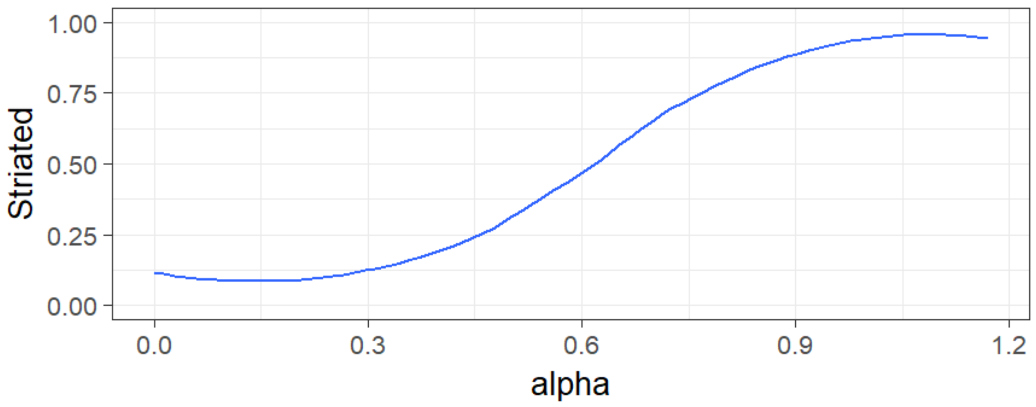


Figure 10.25: *The effect of restricting the subset of available angles in the guided random search algorithm. The more restricted the subset of angles, the more straight lines (and hence striation) is present in the configuration produced by the guided random search algorithm. The line in the image has been fit using LOESS.*

10.4 Future Work

The concept of summarizing a configuration numerically using inherent structure is quite an appealing concept. The current implementation serves as a great base from which to build. This section will briefly discuss some future work projects that would greatly improve the scagnostics framework.

10.4.1 A Modular Framework

The current iteration of scagnostics forces the user to compute every scagnostic measure on their configuration. This means that, regardless of what they may be most interested in, they are forced to compute every geometric graph object, and all of the scagnostic measures every time. A preferable approach is to allow a prospective user to specify which scagnostics they are interested in calculating, which would in turn mean computing only the graph objects required to compute those scagnostics. For instance, if a user is only interested in the outlying scagnostic, only the minimal spanning tree would be computed, as neither the alpha hull or convex hull are required in the computation.

A theoretical call to compute only the outlying scagnostic could be

```
p <- scagnostics(X) +
     outlying()
```

which is very similar to the type of call scene in *ggplot* [117]. The two approaches to computing the outlying scagnostic are compared in Table 10.1.

In the modular framework, any work done would be stored in an R object (`p` in this case). This way, if other scagnostic measures are of interest at a later time, the pre-processing steps and geometric graphs already computed would not need to be repeated - the implementation would simply compute the graphs and scagnostics required that have not been already computed. For example,

```
p + convex()
```

would need to compute only the convex and alpha hulls, and return the convex scagnostic.

Computing scagnostics modularly has two additional benefits. First, it allows the implementation to be made more general, allowing for custom scagnostics to be added. For instance, suppose a user was interested in the mean edge length of the minimal spanning tree. Defining a function such as

Operation	Current Implementation	Modular Implementation
1	Scale	Scale
2	Hexagonal Binning	Hexagonal Binning
3	Compute Distance Matrix	Compute Distance Matrix
4	Compute MST	Compute MST
5	Identify and Remove Outliers	Identify and Remove Outliers
6	Compute Outlying	Compute Outlying
7	Re-compute MST	
8	Compute Alpha and Convex Hulls	
9	Compute RUNT Graph	
10	Compute Remaining Scagnostics	
11	Return Scagnostics	Return Outlying

Table 10.1: *A comparison of the steps needed to produce the outlying scagnostic in the original implementation of scagnostics (left) and the newly proposed modular scagnostic implementation.*

```
mst.mean <- function(...){
  mean(mst@edges)
}
```

which takes the minimal spanning tree object stored in `p`, and returns the mean edge length of the minimal spanning tree with the following call

```
p + mst.mean()
```

This new functionality is greatly beneficial to those users who have a specific structure they are chasing that is not currently captured by the scagnostic measures.

Additionally, computing scagnostics individually also opens the door to the addition of new scagnostics to the existing framework. By simply adding new scagnostics and computing them all each and every time, the computational complexity of the algorithm continues to increase. Modular scagnostics also opens the door to the computation of additional geometric objects (such as the k-nearest neighbour graph) that perhaps are only required for one or two specialty scagnostics and need not be computed every time.

10.4.2 Scagnostics in Higher Dimensions

Perhaps the largest hindrance in the current scagnostics framework is the limitation to two dimensional configurations. With the dimension of data ever increasing, combined with the ability to effectively visualize data in these higher dimensions with tools such as Loon [115], the ability to only identify structure in two dimensions is severely limiting.

Existing work has been done in this field, namely in [41], but there is plenty of work that could be done to improve scagnostics in not only three dimensions, but also in higher dimensions.

While all of the geometric graphs used in defining scagnostics generalize in a straight-forward fashion to high-dimensional space, consideration for whether or not there are natural extensions of the measures on these graphs in high dimensional space needs to be taken into account. For instance, recall that the striated scagnostic considers angles between adjacent edges in the minimal spanning tree. In their implementation in three dimensional space, Fu & Oldford [41] consider not only angles between adjacent edges, but also angles between adjacent planes (formed by three adjacent edges). In doing so, the concept of striation is extended from two dimensions to three dimensions by considering striation that occurs in three dimensional space, in addition to the striation that occurs in two dimensional space. What was not explored, however, was the actual types of configurations that may score highly on this scagnostic and if they represent interesting structure.

Finally, there are some current scagnostics for which a straight-forward generalization does not exist. For instance, consider the difficulty of generalizing the monotonic scagnostic. In two dimensions, it's simply calculated as Spearman's rank correlation. Unfortunately, this correlation does not generalize to three dimensions in a straight forward fashion. [41] considered calculating the pairwise partial Spearman's ρ correlations, and taking the largest value. For instance, calculating the partial correlation between X & Y, conditional on Z, we have

$$\rho_{XY \cdot Z} = \frac{\rho_{XY} - \rho_{XZ}\rho_{YZ}}{\sqrt{1-\rho_{XY}^2}\sqrt{1-\rho_{YZ}^2}}$$

where ρ_{**} represent Spearman's ρ between the specified variables. The monotonic scagnostic is then

$$c_{monotonic} = max(\rho_{XZ \cdot Y}^2, \rho_{YZ \cdot X}^2, \rho_{XY \cdot Z}^2)$$

However, looking at conditional two-dimensional correlations may not be representative of monotonicity in three dimensions, for example if X & Y are highly correlated, but X & Z and Y & Z are not. A new proposal will need to be made that better captures monotonicity in high dimensions.

These issues represent just some of the difficulties of properly generalizing the scagnostic framework to higher dimensional space. More importantly, however, is having a solid two-dimensional base on which to stand before generalizing to even three dimensions is feasible. Several issues have been identified, with proposed solutions, within this thesis, representing a first step in solidifying an appropriate two-dimensional base for which scagnostics can be generalized.

References

[1] Oguz Akbilgic, Hamparsum Bozdogan, and M Erdal Balaban. A novel hybrid RBF neural networks model as a forecaster. *Statistics and Computing*, 24(3):365–375, 2014.

[2] Abdo Y Alfakih, Amir Khandani, and Henry Wolkowicz. Solving Euclidean distance matrix completion problems via semidefinite programming. *Computational optimization and applications*, 12(1-3):13–30, 1999.

[3] Babak Alipanahi Ramandi. *New approaches to protein NMR automation.* PhD thesis, University of Waterloo, 2011.

[4] Shun-ichi Amari, Andrzej Cichocki, and Howard Hua Yang. A new learning algorithm for blind signal separation. In *Advances in neural information processing systems*, pages 757–763, 1996.

[5] E Anderson. The irises of the Gaspe Peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

[6] Ralph G Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Physical Review E*, 64(6):061907, 2001.

[7] Douglas Bates and Martin Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2015. R package version 1.2-2.

[8] Stephen D Bay. Multivariate discretization for set mining. *Knowledge and Information Systems*, 3(4):491–512, 2001.

[9] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pages 585–591, 2001.

[10] Marc Boullé. MODL: a Bayes optimal discretization method for continuous attributes. *Machine Learning*, 65(1):131–165, 2006.

[11] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15. Studies in Applied Mathematics, SIAM, Philadelphia, Pennsylvania, 1994.

[12] Hector Corrada Bravo. *Rcsdp: R Interface to the CSDP Semidefinite Programming Library*, 2016. R package version 0.1.55.

[13] James P Bridge, Sean B Holden, and Lawrence C Paulson. Machine learning for first-order theorem proving. *Journal of Automated Reasoning*, 53(2):141–172, 2014.

[14] Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. *Numerical Geometry of Non-Rigid Shapes*. Springer Science & Business Media, 2008.

[15] Krisztian Buza. Feedback prediction for blogs. In *Data Analysis, Machine Learning and Knowledge Discovery*, pages 145–152. Springer, 2014.

[16] Luis M Candanedo, Véronique Feldheim, and Dominique Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings*, 140:81–97, 2017.

[17] Daniel B Carr. Looking at large data sets using binned data plots. In *Computing and Graphics in Statistics*, pages 7–39. Springer-Verlag, New York, New York, 1991.

[18] Daniel B Carr, Richard J Littlefield, WL Nicholson, and JS Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[19] Jesús Cerquides and Ramon López De Màntaras. Proposal and empirical comparison of a parallelizable distance-based discretization method. In *KDD*, pages 139–142, 1997.

[20] John Y. Ching, Andrew K. C. Wong, and Keith C. C. Chan. Class-dependent discretization for inductive learning from continuous and mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):641–651, 1995.

[21] Moody T Chu and Joel W Wright. The educational testing problem revisited. *IMA Journal of Numerical Analysis*, 15(1):141–160, 1995.

[22] J Conway and N Sloane. Fast quantizing and decoding and algorithms for lattice quantizers and codes. *IEEE Transactions on Information Theory*, 28(2):227–232, 1982.

[23] J Conway and N Sloane. Voronoi regions of lattices, second moments of polytopes, and quantization. *IEEE Transactions on Information Theory*, 28(2):211–226, 1982.

[24] Andrea Coraddu, Luca Oneto, Aessandro Ghio, Stefano Savio, Davide Anguita, and Massimo Figari. Machine learning approaches for improving condition-based maintenance of naval propulsion plants. *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, 230(1):136–153, 2016.

[25] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[26] Paulo Cortez and Aníbal de Jesus Raimundo Morais. A data mining approach to predict forest fires using meteorological data. In J. Neves and J. Machado, editors, *New Trends in Artificial Intelligence, Proceedings of the 13th EPIA*, pages 512–523. Portuguese Conference on Artificial Intelligence, 2007.

[27] Tuan Nhon Dang and Leland Wilkinson. Transforming scagnostics to reveal hidden features. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1624–1632, 2014.

[28] Aritra Dasgupta and Robert Kosara. Pargnostics: Screen-space metrics for parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1017–1026, 2010.

[29] S De Vito, E Massera, M Piga, L Martinotto, and G Di Francia. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical*, 129(2):750–757, 2008.

[30] Stefano Demarta and Alexander J McNeil. The t copula and related copulas. *International Statistical Review/Revue Internationale de Statistique*, pages 111–129, 2005.

[31] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[32] Yichuan Ding, Nathan Krislock, Jiawei Qian, and Henry Wolkowicz. Sensor network localization, Euclidean distance matrix completions, and graph realization. *Optimization and Engineering*, 11(1):45–66, 2010.

[33] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[34] Haw-ren Fang and Dianne P O'Leary. Euclidean distance matrix completion problems. *Optimization Methods and Software*, 27(4-5):695–717, 2012.

[35] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1029. Morgan-Kaufmann, 1993.

[36] Elaine Fehrman, Awaz K Muhammad, Evgeny M Mirkes, Vincent Egan, and Alexander N Gorban. The five factor model of personality and evaluation of drug consumption risk. In *Data Science*, pages 231–242. Springer, 2017.

[37] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[38] Roger Fletcher. A nonlinear programming problem in statistics (educational testing). *SIAM Journal on Scientific and Statistical Computing*, 2(3):257–267, 1981.

[39] HA Friberg. Rmosek: The R-to-MOSEK optimization interface. *R package version*, 1(3), 2012.

[40] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[41] Lijie Fu. Implementation of three-dimensional scagnostics. Master's thesis, University of Waterloo, 2009.

[42] Salvador Garcia, Julian Luengo, José Antonio Sáez, Victoria Lopez, and Francisco Herrera. A survey of discretization techniques: taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750, 2013.

[43] David Gil, Jose Luis Girela, Joaquin De Juan, M Jose Gomez-Torres, and Magnus Johnsson. Predicting seminal quality with artificial intelligence methods. *Expert Systems with Applications*, 39(16):12564–12573, 2012.

[44] W Glunt, TL Hayden, and M Raydan. Molecular conformations from distance matrices. *Journal of Computational Chemistry*, 14(1):114–120, 1993.

[45] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[46] John C Gower and Garmt B Dijksterhuis. *Procrustes Problems*, volume 30. Oxford University Press on Demand, 2004.

[47] John C Gower and GJS Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, pages 54–64, 1969.

[48] Daniel B Graham and Nigel M Allinson. Characterising virtual eigensignatures for general purpose face recognition. In *Face Recognition*, pages 446–456. Springer, 1998.

[49] Peter Güntert. Automated NMR structure calculation with CYANA. *Protein NMR Techniques*, pages 353–378, 2004.

[50] Peter Hall and Matt P Wand. On the accuracy of binned kernel density estimators. *Journal of Multivariate Analysis*, 56(2):165–184, 1996.

[51] Timothy F Havel and Kurt Wüthrich. An evaluation of the combined use of nuclear magnetic resonance and distance geometry for the determination of protein conformations in solution. *Journal of Molecular Biology*, 182(2):281–294, 1985.

[52] Christoph Helmberg, Franz Rendl, Robert J Vanderbei, and Henry Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2):342–361, 1996.

[53] Bruce Hendrickson. The molecule problem: exploiting structure in global optimization. *SIAM Journal on Optimization*, 5(4):835–857, 1995.

[54] Nicholas J Higham. Computing the nearest correlation matrix-a problem from finance. *IMA Journal of Numerical Analysis*, 22(3):329–343, 2002.

[55] KM Ho and PD Scott. Zeta: A global method for discretization of continuous variables. In *3rd International Conference on Knowledge Discovery and Data Mining (KDD99), NewPort Beach, USA*, pages 191–194, 1997.

[56] Toby Dylan Hocking, Armand Joulin, Francis Bach, and Jean-Philippe Vert. Clusterpath an algorithm for clustering using convex fusion penalties. In *28th International Conference on Machine Learning*, page 1, 2011.

[57] Heike Hofmann, Hadley Wickham, and Karen Kafadar. Letter-value plots: Boxplots for large data. *Journal of Computational and Graphical Statistics*, 26(3):469–477, 2017.

[58] Lasse Holmström. The accuracy and the computational complexity of a multivariate binned kernel density estimator. *Journal of Multivariate Analysis*, 72(2):264–309, 2000.

[59] Ramon Huerta, Thiago Mosqueiro, Jordi Fonollosa, Nikolai F Rulkov, and Irene Rodriguez-Lujan. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems*, 157:169–176, 2016.

[60] Fritz John. Extremum problems with inequalities as subsidiary conditions. In *Traces and Emergence of Nonlinear Programming*, pages 197–215. Springer-Verlag, 2014.

[61] Brian A Johnson and Kotaro Iizuka. Integrating openstreetmap crowdsourced data and landsat time-series imagery for rapid land use/land cover (LULC) mapping: case study of the laguna de bay area of the philippines. *Applied Geography*, 67:140–149, 2016.

[62] Charles R Johnson and Pablo Tarazaga. Connections between the real positive semidefinite and distance matrix completion problems. *Linear Algebra and its Applications*, 223:375–391, 1995.

[63] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer-Verlag, 1972.

[64] Randy Kerber. Chimerge: discretization of numeric attributes. In *Proceedings of the tenth national conference on Artificial intelligence*, pages 123–128. Aaai Press, 1992.

[65] Brian J Kernohan, Robert A Gitzen, and Joshua J Millspaugh. Analysis of animal space use and movements. In *Radio Tracking and Animal Populations*, pages 125–166. Elsevier, 2001.

[66] Nathan Krislock and Henry Wolkowicz. Explicit sensor network localization using semidefinite representations and facial reductions. *SIAM Journal on Optimization*, 20(5):2679–2708, 2010.

[67] Nathan Krislock and Henry Wolkowicz. Euclidean distance matrices and applications. In Miguel Anjos and Jean Lasserre, editors, *Handbook on Semidefinite, Cone and Polynomial Optimization: Theory, Algorithms, Software and Applications*, volume 166 of *International Series in Operations Research & Management Science*, chapter 30, pages 879–914. Springer Science & Business Media, 2011.

[68] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[69] Maryann Lawlor. Small systems, big business. Signal Magazine, 2005.

[70] Xuan Liang, Tao Zou, Bin Guo, Shuo Li, Haozhe Zhang, Shuyi Zhang, Hui Huang, and Song Xi Chen. Assessing Beijing's PM2.5 pollution: severity, weather impact, APEC and winter heating. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 471(2182), 2015.

[71] RJ Lyon, BW Stappers, S Cooper, JM Brooke, and JD Knowles. Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 459(1):1104–1123, 2016.

[72] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM, 2002.

[73] Christopher D Manning, Hinrich Schütze, et al. *Foundations of Statistical Natural Language Processing*, volume 2. MIT Press, 1999.

[74] David J Marchette. *Random Graphs for Statistical Pattern Recognition.* John Wiley & Sons, 2004.

[75] James Stephen Marron, Michael J Todd, and Jeongyoun Ahn. Distance-weighted discrimination. *Journal of the American Statistical Association*, 102(480):1267–1271, 2007.

[76] Marvin McNett and Geoffrey M Voelker. Access and mobility of wireless PDA users. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(2):40–55, 2005.

[77] Sameep Mehta, Srinivasan Parthasarathy, and Hui Yang. Toward unsupervised correlation preserving discretization. *IEEE Transactions on Knowledge and Data Engineering*, 17(9):1174–1185, 2005.

[78] Jorge J Moré and Zhijun Wu. Distance geometry optimization for protein structures. *Journal of Global Optimization*, 15(3):219–234, 1999.

[79] Pedro J Moreno, Purdy P Ho, and Nuno Vasconcelos. A Kullback-Leibler divergence based kernel for SVM classification in multimedia applications. In *Advances in Neural Information Processing Systems*, pages 1385–1392, 2004.

[80] Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

[81] Yu E Nesterov and Michael J Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, 1997.

[82] Jari Oksanen, F. Guillaume Blanchet, Michael Friendly, Roeland Kindt, Pierre Legendre, Dan McGlinn, Peter R. Minchin, R. B. O'Hara, Gavin L. Simpson, Peter Solymos, M. Henry H. Stevens, Eduard Szoecs, and Helene Wagner. *vegan: Community Ecology Package*, 2017. R package version 2.4-3.

[83] Christos H Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[84] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[85] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110, 1894.

[86] Bernhard Pfaff. *cccp: cone constrained convex problems*, 2015. R package version 0.2-4.

[87] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.

[88] Peter Radchenko and Gourab Mukherjee. Convex clustering via l1 fusion penalization. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2017.

[89] Adam Rahman and R Wayne Oldford. Euclidean distance matrix completion and point configurations from the minimal spanning tree. *SIAM Journal on Optimization*, 28(1):528–550, 2018.

[90] R. Rangarajan, R. Raich, and A. O. Hero. Euclidean matrix completion problems in tracking and geo-localization. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5324–5327, March 2008.

[91] Murray Rosenblatt et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

[92] Azriel Rosenfeld and Avinash Kak. *Digital Picture Processing.* Academic press, 1976.

[93] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[94] Jeffrey A. Ryan and Joshua M. Ulrich. *quantmod: Quantitative Financial Modelling Framework*, 2017. R package version 0.4-9.

[95] Daniel V Samarov. The fast rodeo for local polynomial regression. *Journal of Computational and Graphical Statistics*, 24(4):1034–1052, 2015.

[96] David W Scott. A note on choice of bivariate histogram bin shape. *Journal of Official Statistics*, 4(1):47, 1988.

[97] Bernard W Silverman. *Density Estimation for Statistics and Data Analysis*, volume 26. CRC press, 1986.

[98] Gregory E Sims, Se-Ran Jun, Guohong A Wu, and Sung-Hou Kim. Alignment-free genome comparison with feature frequency profiles (ffp) and optimal resolutions. *Proceedings of the National Academy of Sciences*, 106(8):2677–2682, 2009.

[99] Kirstine Smith. On the standard deviations of adjusted and interpolated values of an observed polynomial function and its constants and the guidance they give towards a proper choice of the distribution of observations. *Biometrika*, 12(1-2):1–85, 1918.

[100] W. Stuetzle. Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. *Journal of Classification*, 20:25–47, 2003.

[101] Robert Szewczyk, Eric Osterweil, Joseph Polastre, Michael Hamilton, Alan Mainwaring, and Deborah Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40, 2004.

[102] Francis EH Tay and Lixiang Shen. A modified chi2 algorithm for discretization. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):666–670, 2002.

[103] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[104] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[105] Kim-Chuan Toh, Michael J Todd, and Reha H Tütüncü. SDPT3 - a Matlab software package for semidefinite programming, version 1.3. *Optimization Methods and Software*, 11(1-4):545–581, 1999.

[106] Michael W Trosset. Applications of multidimensional scaling to molecular conformation. 1997.

[107] Michael W Trosset. Distance matrix completion by numerical optimization. *Computational Optimization and Applications*, 17(1):11–22, 2000.

[108] Athanasios Tsanas and Angeliki Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567, 2012.

[109] Pınar Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014.

[110] John W Tukey and Paul A Tukey. Computer graphics and exploratory data analysis: An introduction. In *Proceedings of the Sixth Annual Conference and Exposition; Technical Sessions*, volume III, pages 773–785. National Computer Graphics Association, 1985.

[111] John W Tukey and Paul A Tukey. Computer graphics and exploratory data analysis: An introduction. *The Collected Works of John W. Tukey: Graphics: 1965-1985*, 5:419, 1988.

[112] P Tukey and J Tukey. Graphic display of data sets in 3 or more dimensions. *The Collected Works of John Tukey*, 5:189–288, 1988.

[113] Reha H Tütüncü, Kim-Chuan Toh, and Michael J Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming*, 95(2):189–217, 2003.

[114] Lieven Vandenberghe, Stephen Boyd, and Shao-Po Wu. Determinant maximization with linear matrix inequality constraints. *SIAM journal on Matrix Analysis and Applications*, 19(2):499–533, 1998.

[115] Adrian Waddell. *Interactive visualization and exploration of high-dimensional data.* PhD thesis, University of Waterloo, 2016.

[116] Geoffrey S Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372, 1964.

[117] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis.* Springer Science & Business Media, 2009.

[118] Lee Wilkinson and Anushka Anand. *scagnostics: compute scagnostics - scatterplot diagnostics*, 2012. R package version 0.2-4.

[119] Leland Wilkinson, Anushka Anand, and Robert L Grossman. Graph-theoretic scagnostics. In *INFOVIS*, volume 5, page 21, 2005.

[120] Leland Wilkinson and Graham Wills. Scagnostics distributions. *Journal of Computational and Graphical Statistics*, 17(2):473–491, 2008.

[121] Andrew KC Wong and David KY Chiu. Synthesizing statistical knowledge from incomplete mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):796–805, 1987.

[122] Kurt Wüthrich. Protein structure determination in solution by NMR spectroscopy. *Journal of Biological Chemistry*, 265(36):22059–22062, 1990.

[123] Ying Yang and Geoffrey I Webb. Discretization for naive-Bayes learning: managing discretization bias and variance. *Machine Learning*, 74(1):39–74, 2009.

[124] I-C Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, 1998.

[125] I-Cheng Yeh. Modeling slump flow of concrete using second-order regressions and artificial neural networks. *Cement and Concrete Composites*, 29(6):474–480, 2007.

[126] I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480, 2009.

[127] I-Cheng Yeh, King-Jang Yang, and Tao-Ming Ting. Knowledge discovery on RFM model using Bernoulli sequence. *Expert Systems with Applications*, 36(3):5866–5871, 2009.

[128] Fang Zhou, Q Claire, and Ross D King. Predicting the geographical origin of music. In *2014 IEEE International Conference on Data Mining (ICDM)*, pages 1115–1120. IEEE, 2014.

[129] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. LBFGS-B: Fortran subroutines for large-scale bound constrained optimization. *Report NAM-11, EECS Department, Northwestern University*, 1994.

[130] Zhisu Zhu and Yinyu Ye. *Rdsdp: R Interface to DSDP Semidefinite Programming Library*, 2016. R package version 1.0.4-2.

[131] Djamel A Zighed, Sabine Rabaséda, and Ricco Rakotomalala. FUSINTER: a method for discretization of continuous attributes. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(03):307–326, 1998.

Appendices

Appendix A

Mathematical Proofs

A.1 Proof of Inverse Relationship between $\mathcal{T}$ and K

This statement is proven in two parts. First, any matrix $\mathbf{D} \in \mathcal{D}_n^-$ is by definition negative semidefinite on the space

$$\{\mathbf{z} \in \mathcal{R}^n : \mathbf{z}^\mathsf{T}\mathbf{1} = 0\}$$

Then, applying the function $\mathcal{T}$ to this matrix results in

$$\mathcal{T}(\mathbf{D}) = -\frac{1}{2}\mathbf{PDP}$$

For any $\mathbf{x}$ such that $\mathbf{x}^\mathsf{T}1 = 0$, we therefore have

$$\begin{aligned}
\mathbf{x}^\mathsf{T}\mathcal{T}(\mathbf{D})\mathbf{x} &= -\frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{PDPx} \\
&= -\frac{1}{2}\mathbf{x}^\mathsf{T}(\mathbf{I}-\mathbf{H})\mathbf{D}(\mathbf{I}-\mathbf{H})\mathbf{x} \\
&= -\frac{1}{2}\mathbf{x}^\mathsf{T}(\mathbf{I}-\frac{\mathbf{11}^\mathsf{T}}{n})\mathbf{D}(\mathbf{I}-\frac{\mathbf{11}^\mathsf{T}}{n})\mathbf{x} \\
&= -\frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{Dx}
\end{aligned}$$

which is positive semidefinite, and therefore

$$\mathcal{T}(\mathbf{D}) \in \mathcal{G}_n^+$$

To prove the converse case, for any $\mathbf{G} \in \mathcal{G}_n^+$, applying the function $\mathcal{K}$ results in a symmetric and hollow matrix by definition. Then, for any $\mathbf{x}$ such that $\mathbf{x}^\mathsf{T}1 = 0$,

$$\begin{aligned}\mathbf{x}^\mathsf{T}\mathcal{K}(\mathbf{G})\mathbf{x} &= \mathbf{x}^\mathsf{T}(\mathbf{1}\mathbf{g}^\mathsf{T} + \mathbf{g}\mathbf{1}^\mathsf{T} - 2\mathbf{G})\mathbf{x} \\ &= -2\mathbf{x}^\mathsf{T}\mathbf{G}\mathbf{x}\end{aligned}$$

Since $\mathbf{G}$ is positive semidefinite by definition, $-2\mathbf{x}^\mathsf{T}\mathcal{K}(\mathbf{G})\mathbf{x}$ is necessarily negative semidefinite, and therefore

$$\mathcal{K}(\mathbf{G}) \in \mathcal{D}_n^-$$

which completes the proof.

Appendix B

Scagnostic Universe Data Sets

The following data sets were used in the Scagnostics universe experiment of Chapter 2. All data sets can be retrieved from the UCI Machine Learning Data Base [31]. Citations for data sets with specific citation requirements are also present.

1. Abalone
2. Airfoil Self-Noise
3. Air Quality [29]
4. Appliance Energy Prediction [16]
5. Anuran Calls (MFCCs)
6. au2
7. au5
8. Auto MPG
9. Automobile
10. Banknote Authentication
11. Beijing PM2.5 Data [70]
12. BlogFeedback Data Set [15]
13. Blood Transfusion Service Center [127]
14. Breast Tissue

15. Breast Cancer Wisconsin (Prognostic)
16. Cardiotocography
17. Car Evaluation
18. Chess (King-Rook vs. King)
19. Cloud
20. Condition Based Maintenance of Naval Propulsion Plants [24]
21. Contraceptive Method Choice
22. Cover Type
23. Combined Cycle Power Plant [109]
24. Computer Hardware
25. Concrete Slump Test [125]
26. Concrete Compressive Strength [124]
27. Crowdsourced Mapping [61]
28. Dataset for Sensorless Drive Diagnosis
29. Default of Credit Card Clients [126]
30. Dermatology
31. Drug Consumption (quantified) [36]
32. Ecoli
33. Energy Efficiency [108]
34. Epileptic Seizure Recognition [6]
35. Fertility [43]
36. First-order Theorem Proving [13]
37. Forest Fires [26]
38. Gas Sensors for Home Activity Monitoring [59]
39. Geographical Original (sic) of Music [128]
40. Glass Identification

41. Haberman's Survival
42. HTRU2 [71]
43. ILPD (Indian Liver Patient Dataset)
44. Istanbul Stock Exchange [1]
45. MAGIC Gamma Telescope
46. Physiochemical Properties of Protein Tertiary Structure
47. Statlog (Shuttle)
48. Spoken Arabic Digit
49. Wine Quality [25]
50. Yeast

Appendix C

Semidefinite Quadratic Linear Programming in R - sdpt3r

Convex optimization is a well traversed field with far reaching applications. While perhaps unfamiliar to those in the statistical sciences, many problems important to statisticians can be formulated as a convex optimization, perhaps the most well known of which would be the least squares problem. More specifically, many problems in statistics can be formulated as a subset of these convex optimization problems, known as *conic linear optimization problems.*

One such example would be the nearest correlation matrix problem [54], which was first considered when attempting to find correlations between stocks, where incomplete data on daily stock returns are not unusual. Pairwise correlations are only computed when data is available for both pairs of stocks under consideration, resulting in a correlation matrix that contains pairwise correlations, but is not necessarily positive semidefinite - an *approximate* correlation matrix. The goal is to then find the correlation matrix that is nearest to the approximate correlation matrix in some way.

Other examples of problems that can be formulated in terms of a conic linear optimization problem include D-optimal experimental design [99], classification using distance weighted discrimination [75], minimum volume ellipsoids [60], and problems in educational testing [21].

Problems in related fields can also be solved, including finding the maximum cut (or maximum k-cut) of a graph, finding the upper bound of the Shannon entropy of a graph, also known as the Lovasz number [114], as well as problems in control theory, Toeplitz matrix approximation, and Chebyshev approximation.

For the purpose of solving these conic linear optimization problems, we introduce the `R` package *sdpt3r*, an implementation of the `MATLAB` package *SDPT3* [105]. While there are currently functions in `R` available to solve some of the specific problems mentioned above - for instance the function `nearPD` in the *Matrix* package by [7] solves the nearest correlation matrix problem - there currently does not exist a general solver for conic linear optimization in R, making *sdpt3r* a novel addition to the R library.

Of the R packages available to perform optimization, *sdpt3r* is the most general. The *cccp* package [86] allows for linear and quadratic conic optimization, and the *Rdsdp* [130] & *Rcsdp* [12] packages allow for semidefinite optimization, while the *sdpt3r* package allows for all of linear, quadratic, and semidefinite conic optimization to be solved simultaneously (i.e., a problem with any combination of semidefinite, quadratic, or linear cones can be solved). In addition, *sdpt3r* allows for problems with log-barrier terms in the objective function to be solved. None of the packages mentioned allow for this. This, of course, comes with the downside of potentially longer run-times when using *sdpt3r*. While the other packages are optimized for their specific purpose, the algorithm implemented in *sdpt3r* is more general. As such, a problem that can be formulated as a straight-forward semidefinite conic optimization problem, for instance, may be solved faster using *Rdsdp*. Other packages for optimization in R, such as *Rmosek* [39], require the purchase of a commercial license for additional third party software.

In Sections C.1, C.2, and C.3 the mathematical formulation of the linear conic optimization problem is discussed in greater detail, and three examples are introduced to explore the increasing generality of the problem to be solved. Section C.4 discusses the R implementation of *sdpt3r*, and the main function by which conic linear optimization problems are solved, `sqlp`, including the required input, and the output generated. The same examples used in Section C.1 will be used to demonstrate how a standard conic linear optimization problem can be converted to a form solvable by `sqlp`. Section C.5 presents the classic form of several other well known problems that can be solved using *sdpt3r*, as well as the helper functions available to convert them to the appropriate form.

C.1 Conic Linear Optimization

At its simplest, a conic linear optimization problem has the following standard form [113]:

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C},\ \mathbf{X} \rangle \\ \text{subject to} & \\ & \begin{array}{rcl} \langle \mathbf{A}_k,\ \mathbf{X} \rangle & = & \mathbf{b}_k, \quad k = 1, ..., m \\ \mathbf{X} & \in & \mathcal{K} \end{array} \end{array} \tag{C.1}$$

where $\mathcal{K}$ is a cone. Generally, $\mathcal{K}$ is either a

- Semidefinite Cone - $\mathcal{S}^n = \{\mathbf{X} \in \mathcal{R}^{n \times n} : \mathbf{X} \succeq 0, \mathbf{X}_{ij} = \mathbf{X}_{ji} \ \forall\ i \neq j\}$
- Quadratic Cone - $\mathcal{Q}^n = \{\mathbf{x} = [x_0; \tilde{\mathbf{x}}] \in \mathcal{R}^n : x_0 \geq \sqrt{\tilde{\mathbf{x}}^\intercal \tilde{\mathbf{x}}}\}$

- Linear Cone - $\mathcal{L}^n$ - non-negative orthant of $\mathcal{R}^n$

Here, $\tilde{\mathbf{x}} = [x_1, \ldots, x_{n-1}]$, and $\langle \cdot,\cdot \rangle$ represents the standard inner product in the appropriate space. In the semidefinite cone the inner product is $\langle \mathbf{X},\ \mathbf{Y} \rangle = vec(\mathbf{X})^{\mathsf{T}} vec(\mathbf{Y})$, where the operator *vec* is the by-column vector version of the matrix $\mathbf{X}$, that is, for the $n \times n$ matrix $\mathbf{X} = [x_{ij}]$, $vec(\mathbf{X})$ is the $n^2 \times 1$ vector $[x_{11}, x_{12}, x_{13}, \ldots, x_{(n-1)n}, x_{nn}]^{\mathsf{T}}$. Note that *vec* does not require a square matrix in general.

While not inherently statistical, one of the simplest problems that can be formulated in terms of a conic linear optimization problem is finding the maximum cut of a graph. Let $\mathbf{G} = [\mathbf{V}, \mathbf{E}]$ be a graph with vertices $\mathbf{V}$ and edges $\mathbf{E}$. A *cut* of the graph $\mathbf{G}$ is a partition of the vertices of $\mathbf{G}$ into two disjoint subsets $\mathbf{G}_1 = [\mathbf{V}_1, \mathbf{E}_1]$, $\mathbf{G}_2 = [\mathbf{V}_2, \mathbf{E}_2]$, with $\mathbf{V}_1 \cap \mathbf{V}_2 = \varnothing$. The size of the cut is defined to be the number of edges connecting the two subsets. The *maximum cut* is defined to be the cut of a graph $\mathbf{G}$ whose size is at least as large as any other cut. For a weighted graph object, we can also define the maximum cut to be the cut with weight at least as large as any other cut.

Finding the maximum cut is referred to as the **Max-Cut Problem**, and was one of the first problems found to be NP-complete, and is also one of the 21 algorithms on Karp's 21 NP-complete problems [63]. The Max-Cut problem is also known to be *APX hard* [83], meaning in addition to there being no polynomial time solution, there is also no polynomial time approximation.

Using the semidefinite programming approximation formulation of [45], the Max-Cut problem can be approximated to within an *approximation constant*. For a weighted adjacency matrix $\mathbf{B}$, the objective function can be stated as

$$\begin{array}{lll} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C}, \mathbf{X} \rangle & \\ \text{subject to} & & \\ & diag(\mathbf{X}) = & \mathbf{1} \\ & \mathbf{X} \in & \mathcal{S}^n \end{array}$$

where $\mathcal{S}^n$ is the cone of symmetric positive semidefinite matrices of size n, and $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. Here, we define $diag(\mathbf{a})$ for an $n \times 1$ vector $\mathbf{a}$ to be the diagonal matrix $\mathbf{A} = [A_{ij}]$ of size $n \times n$ with $A_{ii} = a_i, \quad i = 1, \ldots, n$. For a matrix $\mathbf{X}$, $diag(\mathbf{X})$ extracts the diagonal elements from $\mathbf{X}$ and places them in a column-vector.

To see that the Max-Cut problem is a conic linear optimization problem it needs to be written in the same form as Equation (C.1). The objective function is already in a form identical to that of Equation (C.1), with minimization occurring over $\mathbf{X}$ of its inner product with a constant matrix $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. There are n equality constraints of the form $x_{kk} = 1, \quad k = 1, ..., n$, where x_{kk} is the k^{th} diagonal element of $\mathbf{X}$, and $b_k = 1$ in Equation (C.1). To represent this in the form $\langle \mathbf{A}_k,\ \mathbf{X} \rangle = x_{kk}$, take $\mathbf{A}_k$ to be

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

Now $\langle \mathbf{A}_k,\ \mathbf{X} \rangle = vec(\mathbf{A}_k)^{\mathsf{T}} vec(\mathbf{X}) = x_{kk}$ as required, and the Max-Cut problem is specified as a conic linear optimization problem.

Allowing for optimization to occur over only one variable at a time is quite restrictive, as only a small number of problems can be formulated in this form. Allowing optimization to occur over multiple variables simultaneously would allow for a broader range of problems to be solved.

C.2 A Separable Set of Variables

The conic linear optimization problem actually covers a much wider class of problems than those expressible as in Equation (C.1). Variables can be separated into those which are constrained to a semidefinite cone, $\mathcal{S}$, a quadratic cone, $\mathcal{Q}$, or a linear cone, $\mathcal{L}$. The objective function is a sum of the corresponding inner products of each set of variables. The linear constraint is simply a sum of variables of linear functions of each set. This more general version of the conic linear optimization problem is

$$\begin{array}{ll} \underset{\mathbf{X}^s,\mathbf{X}^q,\mathbf{X}^l}{\text{minimize}} & \sum_{j=1}^{n_s} \langle \mathbf{C}_j^s,\ \mathbf{X}_j^s \rangle + \sum_{i=1}^{n_q} \langle \mathbf{C}_i^q,\ \mathbf{X}_i^q \rangle + \langle \mathbf{C}^l,\ \mathbf{X}^l \rangle \\ \text{subject to} & \\ & \begin{array}{rcl} \sum_{j=1}^{n_s} (\mathbf{A}_j^s)^{\mathsf{T}} svec(\mathbf{X}_j^s) + \sum_{i=1}^{n_q} (\mathbf{A}_i^q)^{\mathsf{T}} \mathbf{X}_i^q + (\mathbf{A}^l)^{\mathsf{T}} \mathbf{X}^l & = & \mathbf{b} \\ \mathbf{X}_j^s & \in & \mathcal{S}^{s_j} \quad \forall\, j \\ \mathbf{X}_i^q & \in & \mathcal{Q}^{q_i} \quad \forall\, i \end{array} \end{array} \tag{C.2}$$

Here, *svec* takes the upper triangular elements of a matrix (including the diagonal) in a column-wise fashion and vectorizes them. In general for an $n \times p$ matrix $\mathbf{X} = [x_{ij}]$, $svec(\mathbf{X})$ will have the following form $[x_{11}, x_{12}, x_{22}, x_{13}, \ldots, x_{(n-1)p}, x_{np}]^{\mathsf{T}}$. Recall that matrices in $\mathcal{S}$ are symmetric, so it is sufficient to constrain only the upper triangular elements of the matrix $\mathbf{X}^s$. For this formulation, $\mathbf{A}_j^s$, $\mathbf{A}_i^q$ and $\mathbf{A}^l$ are the constraint matrices of the appropriate size.

Some important problems in statistics can be formulated to fit this form of the optimization problem.

C.2.1 The Nearest Correlation Matrix

First addressed by [54] in dealing with correlations between stock prices, difficulty arises when data is not available for all stocks on each day, which is unfortunately a common occurrence. To help address this situation, correlations are calculated for pairs of stocks only when data is available for both stocks on any given day. The resulting correlation matrix is only approximate in that it is not necessarily positive semidefinite.

This problem was cast by [54] as

$$\begin{array}{lrcl} \underset{\mathbf{X}}{\text{minimize}} & ||\mathbf{R}-\mathbf{X}||_F & & \\ \text{subject to} & & & \\ & diag(\mathbf{X}) & = & \mathbf{1} \\ & \mathbf{X} & \in & \mathcal{S}^n \end{array}$$

where $\mathbf{R}$ is the approximate correlation matrix and $||\cdot||_F$ denotes the Frobenius norm. Unfortunately, the Frobenius norm in the objective function prevents the problem being formatted as a conic linear optimization problem.

Since the matrix $\mathbf{X}$ is constrained to have unit diagonal and be symmetric, and the matrix $\mathbf{R}$ is an approximate correlation matrix, meaning it will also have unit diagonal and be symmetric, we can re-write the objective function as

$$||\mathbf{R}-\mathbf{X}||_F = 2 * ||svec(\mathbf{R}) - svec(\mathbf{X})|| = 2 * ||\mathbf{e}||$$

Now, introduce a variable e_0 such that $e_0 \geq ||\mathbf{e}||$, and define $\mathbf{e}^* = [e_0;\ \mathbf{e}]$. The vector $\mathbf{e}^*$ is now restricted to be in the quadratic cone $\mathcal{Q}^{n(n+1)/2+1}$. This work leads to the formulation of [105]

$$\begin{array}{lrcl} \underset{\mathbf{e}^*,\ \mathbf{X}}{\text{minimize}} & e_0 & & \\ \text{subject to} & & & \\ & svec(\mathbf{R}) - svec(\mathbf{X}) & = & [\mathbf{0}, \mathbf{I}_{n(n+1)/2}]\ \mathbf{e}^* \\ & diag(\mathbf{X}) & = & \mathbf{1} \\ & \mathbf{X} & \in & \mathcal{S}^n \\ & \mathbf{e}^* & \in & \mathcal{Q}^{n(n+1)/2+1} \end{array}$$

Here, $[\mathbf{X}, \mathbf{Y}]$ denotes column binding of the two matrices $\mathbf{X}_{n\times p}$ and $\mathbf{Y}_{n\times m}$ to form a matrix of size $n \times (p+m)$. By minimizing e_0, we indirectly minimize $\mathbf{e} = svec(\mathbf{R}) - svec(\mathbf{X})$, since recall we have $e_0 \geq ||\mathbf{e}||$, which is the goal of the original objective function.

To see this as a conic linear optimization problem, notice that e_0 can be written as $\langle \mathbf{C}^q,\ \mathbf{X}^q \rangle$

by letting $\mathbf{C}^q = [1; \mathbf{0}_{n(n+1)/2}]$ and $\mathbf{X}^q = \mathbf{e}^*$. Since the matrix $\mathbf{X}$ (i.e., $\mathbf{X}^s$) does not appear in the objective function, the matrix $\mathbf{C}^s$ is an $n \times n$ matrix of zeros.

Re-writing the first constraint as

$$svec(\mathbf{X}) + [\mathbf{0}, \mathbf{I}_{n(n+1)/2}]\ \mathbf{e}^* = \quad svec(\mathbf{R})$$

we can easily define the constraint matrices and right hand side of the first constraint as

$$\begin{array}{ll} \mathbf{A}_1^s & = \mathbf{I}_{n(n+1)/2} \\ \mathbf{A}_1^q & = [\mathbf{0}, \mathbf{I}_{n(n+1)/2}] \\ \mathbf{b}_1 & = svec(\mathbf{R}) \end{array}$$

The second constraint is identical to the constraint from the Max-Cut problem, where each diagonal element of $\mathbf{X}$ is constrained to be equal to 1. Define $\mathbf{b}_2 = \mathbf{1}$, and for the k^{th} diagonal element of $\mathbf{X}$, define the matrix $\mathbf{A}_k$ as

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

yielding $\langle \mathbf{A}_k, \mathbf{X} \rangle = x_{kk}$. To write this as $(\mathbf{A}_2^s)^\mathsf{T}\mathbf{X}^s$, define

$$\mathbf{A}_2^s = [svec(\mathbf{A}_1), ..., svec(\mathbf{A}_n)]$$

Since $\mathbf{e}^*$ does not appear in the second constraint, $\mathbf{A}_2^q = \mathbf{0}_{n(n+1)/2+1}$.

The final step is to combine the individual constraint matrices from each constraint to form one constraint matrix for each variable, which is done by defining $\mathbf{A}^s = [\mathbf{A}_1^s,\ \mathbf{A}_2^s]$, $\mathbf{A}^q = [\mathbf{A}_1^q,\ \mathbf{A}_2^q]$. We also concatenate both right hand side vectors to form a single vector by defining $\mathbf{b} = [\mathbf{b}_1;\ \mathbf{b}_2]$. Here, the notation $[\mathbf{X}; \mathbf{Y}]$ is used to denote two matrices $\mathbf{X}_{p\times m}$ and $\mathbf{Y}_{q\times m}$ bound vertically to form a matrix of size $(p+q) \times m$. With this, the nearest correlation matrix problem is written as a conic linear optimization.

C.3 Semidefinite Quadratic Linear Programming

While Equation (C.2) allows for additional variables to be present, it can be made more general still to allow even more problems to be solved. We will refer to this general form as a *semidefinite quadratic linear programming* (SQLP) problem.

The first generality afforded by an SQLP is the addition of an unconstrained variable $\mathbf{X}^u$, which, as the name suggests, is not bound to a cone, but instead, it is "constrained" to the reals in the appropriate dimension. The second generalization is to allow for what are known as *log-barrier* terms to exist in the objective function. In general, a barrier function in an optimization problem is a term that approaches infinity as the point approaches the boundary of the feasible region. As we will see, these log-barrier terms appear as log terms in the objective function.

Recall that for any linear optimization problem, there exists two formulations - the primal formulation and the dual formulation. For the purposes of a semidefinite quadratic linear programming problem, the primal problem will always be defined as a minimization, and the associated dual problem will therefore be a maximization

C.3.1 The Primal Problem

The primal formulation of the SQLP problem is

$$\begin{array}{lrcl}
\underset{\mathbf{X}_j^s,\mathbf{X}_i^q,\mathbf{X}^l,\mathbf{X}^u}{\text{minimize}} & \multicolumn{3}{l}{\sum_{j=1}^{n_s}[\langle \mathbf{C}_j^s,\ \mathbf{X}_j^s\rangle - \mathbf{v}_j^s\ \mathit{log\ det}\ \mathbf{X}_j^s]\ +\ \sum_{i=1}^{n_q}[\langle \mathbf{C}_i^q,\ \mathbf{X}_i^q\rangle - \mathbf{v}_i^q\ \mathit{log}\ \gamma(\mathbf{X}_i^q)]} \\
 & \multicolumn{3}{l}{\quad +\ \langle \mathbf{C}^l,\ \mathbf{X}^l\rangle\ -\ \sum_{k=1}^{n_l}\mathbf{v}_k^l\ \mathit{log}\ \mathbf{X}_k^l\ +\ \langle \mathbf{C}^u,\ \mathbf{X}^u\rangle} \\
\text{subject to} & & & \\
 & \sum_{j=1}^{n_s}\mathbf{A}_j^s(\mathbf{X}_j^s) + \sum_{i=1}^{n_q}\mathbf{A}_i^q\mathbf{X}_i^q + \mathbf{A}^l\mathbf{X}^l + \mathbf{A}^u\mathbf{X}^u & = & \mathbf{b} \\
 & \mathbf{X}_j^s & \in & \mathcal{S}^{s_j} \quad \forall\, j \\
 & \mathbf{X}_i^q & \in & \mathcal{Q}^{q_i} \quad \forall\, i \\
 & \mathbf{X}^l & \in & \mathcal{L}^{n_l} \\
 & \mathbf{X}^u & \in & \mathcal{R}^{n_u}
\end{array} \tag{C.3}$$

For each j, $\mathbf{C}_j^s$ and $\mathbf{X}_j^s$ are symmetric matrices of dimension s_j, restricted to the cone of positive semidefinite matrices of the same dimension. Similarly, for all i, $\mathbf{C}_i^q$ and $\mathbf{X}_i^q$ are real vectors of dimension q_i, restricted to the the quadratic cone of dimension q_i. For a vector $\mathbf{u} = [u_0; \tilde{\mathbf{u}}]$ in a second order cone, define $\gamma(u) = \sqrt{u_0^2 - \tilde{\mathbf{u}}^\mathsf{T}\tilde{\mathbf{u}}}$. Finally, $\mathbf{C}^l$ and $\mathbf{X}^l$ are vectors of dimension n_l, restricted to linear cone of the same dimension, and $\mathbf{C}^u$ and $\mathbf{X}^u$ are unrestricted real vectors of dimension n_u.

As before, the matrices $\mathbf{A}_i^q$, $\mathbf{A}^l$, and $\mathbf{A}^u$ are constraint matrices in q_i, n_l, and n_u dimensions respectively, each corresponding to their respective quadratic, linear, or unrestricted block. $\mathbf{A}_j^s$ is defined to be a linear map from $\mathcal{S}^{s_j}$ to $\mathcal{R}^m$ defined by

$$\mathbf{A}_j^{s_j}(\mathbf{X}_j^s) = [\langle \mathbf{A}_{j,1}^s, \mathbf{X}_j^s\rangle; \ldots ; \langle \mathbf{A}_{j,m}^s, \mathbf{X}_j^s\rangle]$$

where $\mathbf{A}^s_{j,1} \dots \mathbf{A}^s_{j,m} \in \mathcal{S}^{s_j}$ are constraint matrices associated with the j^{th} semidefinite variable $\mathbf{X}^s_j$.

C.3.2 The Dual Problem

The dual problem associated with the semidefinite quadratic linear programming formulation is

$$\begin{array}{lll}
\underset{\mathbf{Z}^s_j,\mathbf{Z}^q_i,\mathbf{Z}^l,\mathbf{y}}{\text{maximize}} & \mathbf{b}^\mathsf{T}\mathbf{y} & + \sum_{j=1}^{n_s}[\mathbf{v}^s_j \; log \; det \; \mathbf{Z}^s_j \; + \; s_j \; \mathbf{v}^s_j \; (1 - log \; \mathbf{v}^s_j)] \\
& & + \sum_{i=1}^{n_q}[\mathbf{v}^q_i \; log \; \gamma(\mathbf{Z}^q_i) \; + \; \mathbf{v}^q_i \; (1 - log \; \mathbf{v}^q_i)] \\
& & + \sum_{k=1}^{n_l}[\mathbf{v}^l_k \; log \; \mathbf{Z}^l_k \; + \; \mathbf{v}^l_k \; (1 - log \; \mathbf{v}^l_k)] \\
\text{subject to} & & \\
& & \begin{array}{lclcll}
(\mathbf{A}^s_j)^\mathsf{T}\mathbf{y} & + \; \mathbf{Z}^s_j & = & \mathbf{C}^s_j, & \mathbf{Z}^s_j \in \mathcal{S}^{s_j}, & j = 1, \dots, n_s \\
(\mathbf{A}^q_i)^\mathsf{T}\mathbf{y} & + \; \mathbf{Z}^q_i & = & \mathbf{C}^q_i, & \mathbf{Z}^q_i \in \mathcal{Q}^{q_i}, & i = 1, \dots, n_q \\
(\mathbf{A}^l)^\mathsf{T}\mathbf{y} & + \; \mathbf{Z}^l & = & \mathbf{C}^l, & \mathbf{Z}^l \in \mathcal{L}^{n_l} & \\
(\mathbf{A}^u)^\mathsf{T}\mathbf{y} & & = & \mathbf{C}^u, & \mathbf{y} \in \mathcal{R}^m &
\end{array}
\end{array} \tag{C.4}$$

where $(\mathbf{A}^s_j)^T$ is defined to be the adjoint operator of $\mathbf{A}^s_j$, where $(\mathbf{A}^s_j)^\mathsf{T}\mathbf{y} = \sum_{k=1}^m \mathbf{y}_k \mathbf{A}^s_{j,k}$. Equations (C.3) and (C.4) represent the most general form of the linear conic optimization problem that can be solved using *sdpt3r*.

C.3.3 Optimal Design of Experiments

Consider the problem of estimating a vector $\mathbf{x}$ from measurements $\mathbf{y}$ given by the relationship

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, 1).$$

The variance-covariance matrix of such an estimator is proportional to $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$. A reasonable goal during the design phase of an experiment would therefore be to minimize $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ in some way.

There are many different ways in which $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ might be made minimal. For example, minimization of the trace of $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ (A-Optimality), minimization of the maximum eigenvalue of $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ (E-Optimality), minimization of the determinant of $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ (D-Optimilaity), and maximization of the trace of $\mathbf{A}^\mathsf{T}\mathbf{A}$ (T-Optimality) all have their merits.

Perhaps the most commonly used of these optimality criteria is D-Optimality, which is equivalent to maximizing the determinant of $\mathbf{A}^\mathsf{T}\mathbf{A}$. Typically, the rows of $\mathbf{A} = [\mathbf{a}_1, ..., \mathbf{a}_q]^T$ are chosen from M possible test vectors $\mathbf{u}_i \in \mathcal{R}^p,\ i = 1, ...M$, which are known in advance. That is,

$$\mathbf{a}_i \in \{\mathbf{u}_1, ..., \mathbf{u}_M\}, \quad i = 1, ..., q$$

Given that the matrix $\mathbf{A}$ is made up of these test vectors $\mathbf{u}_i$, [114] write the matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$ as

$$\mathbf{A}^\mathsf{T}\mathbf{A} = q\sum_{i=1}^{M} \lambda_i \mathbf{u}_i {\mathbf{u}_i}^\mathsf{T} \tag{C.5}$$

where λ_i is the fraction of rows in $\mathbf{A}$ that are equal to the vector $\mathbf{u}_i$. Then, [114] write the D-optimal experimental design problem as a minimum determinant problem

$$\begin{array}{ll} \underset{\boldsymbol{\lambda}}{\text{minimize}} & \log\det\ (\sum_{i=1}^{M} \lambda_i \mathbf{u}_i {\mathbf{u}_i}^\mathsf{T})^{-1} \\ \text{subject to} & \\ & \begin{array}{rcll} \lambda_i & \geq & 0, & i = 1, ..., m \\ \sum_{i=1}^{M} \lambda_i & = & 1 & \end{array} \end{array}$$

Due to the inequality constraint, this primal formulation cannot be interpreted as an SQLP of the form of Equation (C.3). By defining $\mathbf{Z} = \mathbf{u}\ diag(\boldsymbol{\lambda})\ \mathbf{u}^\mathsf{T}$, the dual problem is [105]

$$\begin{array}{ll} \underset{\mathbf{Z},\ \mathbf{z}^l,\ \boldsymbol{\lambda}}{\text{maximize}} & \log\det\ (\mathbf{Z}) \\ \text{subject to} & \\ & \begin{array}{rcll} -\sum_{i=1}^{p} \lambda_i(\mathbf{u}_i {\mathbf{u}_i}^\mathsf{T}) + \mathbf{Z} & = & 0, & \mathbf{Z} \in \mathcal{S}^n \\ -\boldsymbol{\lambda} + \mathbf{z}^l & = & 0, & \mathbf{z}^l \in \mathcal{R}^p_+ \\ \mathbf{1}^T\boldsymbol{\lambda} & = & 1, & \boldsymbol{\lambda} \in \mathcal{R}^p \end{array} \end{array}$$

Keeping in mind that this is a dual configuration, and thus follows Equation (C.4), we proceed with writing the D-Optimal design problem as an SQLP by first considering the objective function. The objective function depends only on the determinant of the matrix variable $\mathbf{Z}$, which is the log-barrier. This indicates that the variable v^s in Equation (C.4) is equal to 1 in this formulation, while v^q and v^l are both zero. Since $\boldsymbol{\lambda}$ does not appear in the objective function, the vector $\mathbf{b}$ is equal to $\mathbf{0}$.

The constraint matrices $\mathbf{A}$ are easy to define in the case of the dual formulation, as they multiply the vector $\mathbf{y}$ in Equation (C.4), so therefore multiply $\boldsymbol{\lambda}$ in our case. In the first constraint, each λ_i is multiplied by the matrix formed by $-\mathbf{u}_i\mathbf{u}_i^\mathsf{T}$, so define $\mathbf{A}_i$ to be

$$\mathbf{A}_i = -\mathbf{u}_i\mathbf{u}_i^{\mathsf{T}}, \quad i = 1, ..., p.$$

Then, the constraint matrix is $\mathbf{A}^s = [svec(\mathbf{A}_1), ..., svec(\mathbf{A}_p)]$. In the second constraint containing the linear variable $\mathbf{z}^l$, the constraint matrix is $\mathbf{A}^l = -\mathbf{I}_p$, and in the third constraint containing only the unconstrained variable $\boldsymbol{\lambda}$, the constraint matrix is $\mathbf{A}^u = \mathbf{1}^{\mathsf{T}}$. Since there is no quadratic variable, $A^q = \mathbf{0}$.

Finally, define the right hand side of each constraint

$$\begin{aligned} \mathbf{C}^s &= \mathbf{0}_{n\times n} \\ \mathbf{C}^l &= \mathbf{0}_{p\times 1} \\ \mathbf{C}^u &= 1 \end{aligned}$$

which fully specifies the D-Optimal design problem as an SQLP.

In the next section, we will demonstrate using `R` how these definitions can be translated for use in the main function of *sdpt3r* so an SQLP problem can be solved.

C.4 Solving a Conic Linear Optimization Problem in R

Each of the problems presented in Section C.1 can be solved using the *sdpt3r* package, an `R` implementation of the `MATLAB` program *SDPT3*. The algorithm is an infeasible primal-dual predictor-corrector path-following method, utilizing either an HKM [52] or NT [81] search direction. The interested reader is directed to [113] for further details surrounding the implementation.

The main function available in *sdpt3r* is `sqlp`, which takes a number of inputs (or an `sqlp_input` object) specifying the problem to be solved, and executes the optimization, returning both the primal and dual solution to the problem. This function will be thoroughly discussed in Section C.4.1, and examples will be provided. In addition to `sqlp`, a prospective user will also have access to a number of helper functions for well known problems that can be solved using *sdpt3r*. For example, the function `maxcut` takes as input an adjacency matrix $\mathbf{B}$, and produces an S3 object containing all the input variables necessary to solve the problem using `sqlp`. These functions will be discussed in Sections C.4.2, C.4.4, C.4.6, and C.5.

For *sdpt3r*, each optimization variable will be referred to as a *block* in the space in which it is restricted. For instance, if we have an optimization variable $\mathbf{X} \in \mathcal{S}^n$, we will refer to this as a semidefinite block of size n. It is important to note that it is possible to have multiple blocks from the same space, that is, it is possible to have both $\mathbf{X} \in \mathcal{S}^n$ as well as $\mathbf{Y} \in \mathcal{S}^m$ in the same problem.

C.4.1 Input Variables

The main function call in *sdpt3r* is `sqlp`, which takes the following input variables

`blk`	A matrix object describing the block structure of the optimization variables.
`At`	A matrix object containing constraint matrices $\mathbf{A}^s$, $\mathbf{A}^q$, $\mathbf{A}^l$, and $\mathbf{A}^u$ for the primal-dual problem.
`b`	A vector containing the right hand side of the equality constraints, $\mathbf{b}$, in the primal problem, or equivalently the constant vector in the dual.
`C`	A matrix object containing the constant $\mathbf{C}$ matrices in the primal objective function or equivalently the corresponding right hand side of the equality constraints in the dual problem.
`X0`, `y0`, `Z0`	Matrix objects containing an initial iterate for the $\mathbf{X}$, $\mathbf{y}$, and $\mathbf{Z}$ variables for the SQLP problem. If not provided, an initial iterate is computed internally.
`OPTIONS`	A list object providing additional parameters for use in `sqlp`. If not provided, default values are used.

The input variable `blk` describes the block structure of the problem. Letting `L` be the total number of semidefinite, quadratic, linear, and unrestricted blocks in the SQLP problem, define `blk` to be an $L \times 2$ matrix object, with the first column describing the type of block, and the second denoting the size of the optimization variable, summarized in Table C.1.

Block type	Column 1	Column 2
Semidefinite	s	s_j
Quadratic	q	q_i
Linear	l	n_l
Unrestricted	u	n_u

Table C.1: *Structure of* `blk`*.*

The input variable `At` corresponds to the constraint matrices in Equation (C.3), and `C` the constant matrices in the objective function. The size of these input variables depends on the block they are representing, summarized in Table C.2 for each block type.

	Block type			
	Semidefinite	Quadratic	Linear	Unrestricted
`At`	$\bar{s}_j \times m$	$q_j \times m$	$n_l \times m$	$n_u \times m$
`C`	$s_j \times s_j$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$

Table C.2: *Size of* `At` *and* `C` *for each block type.*

Note that in Table C.2, $\bar{s}_j = s_j(s_j+1)/2$. The size of `At` in the semidefinite block reflects the upper-triangular input format that has been discussed previously. In a semidefinite block, the optimization variable $\mathbf{X}$ is necessarily symmetric and positive semidefinite, it is therefore more efficient to consider only the upper-triangular portion of the corresponding constraint matrix.

It is important to note that both input variables `At` and `C` are *matrices of matrices*, a constraint matrix and a constant matrix for each optimization variable. While `blk` does not have this same requirement, we will nonetheless use the same structure to initialize `blk` as we do `At` and `C` for the sake of consistency.

In general, the user need not supply initial iterates `X0`, `y0`, and `Z0` for a solution to be found using `sqlp`. The infeasible starting point generated internally by `sqlp` tends to be sufficient to find a solution. If the user wishes to provide a starting point however, the size parameters in Table C.3 must be met for each block.

	Block type			
	Semidefinite	Quadratic	Linear	Unrestricted
`X0`	$s_j \times s_j$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$
`y0`	$s_j \times 1$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$
`Z0`	$s_j \times s_j$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$

Table C.3: *Required size for initial iterates `X0`, `y0`, and `Z0`.*

The user may choose to depart from the default values of several parameters which could affect the optimization by specifying alternative values in the `OPTIONS` list. A complete list of all parameters that can be altered can be found in Appendix C.5.9.

An important example is the specification of the `parbarrier` parameter in `OPTIONS`, which specifies the presence of a log-barrier in the objective function. The default case in `OPTIONS` assumes that the parameters $\mathbf{v}_j^s$, $\mathbf{v}_i^q$, $\mathbf{v}_k^l$ in Equation (C.3) are all $\mathbf{0}$. If this, however, is not the case, then the user must specify an $L \times 1$ matrix object in `OPTIONS$parbarrier` to store the values of these parameters (including zeros). If the j^{th} block is a semidefinite block containing p variables, $parbarrier_j = [v_{j1}^s, ..., v_{jn}^s]$. If the j^{th} block is a quadratic block containing p variables, $parbarrier_j = [v_{j1}^q, ..., v_{jn}^q]$. If the j^{th} block is a linear block $parbarrier_j = [v_1^l, ..., v_{n_l}^l]$. Finally, if the j^{th} block is the unrestricted block, then $parbarrier_j = [0, ..., 0]$, where 0 is repeated n_u times. Section C.4.6 contains an example where `OPTIONS$parbarrier` is specified.

As an additional input option, `sqlp` can take an S3 object of class `sqlp_input`, emanating from one of the helper functions (such as `maxcut`) which will be discussed in later sections. If an S3 object is to be provided, the user must indicate this directly by directly indicating the input is an S3 object by directly assigning it to the `sqlp_obj` input variable, and omitting all other inputs. A user is free to create their own helper functions which create `sqlp_input` objects, and need only ensure that it contains the named fields `blk`, `At`, `C`, `b`, and `OPTIONS`.

When executed, `sqlp` simultaneously solves both the primal and dual problems, meaning solutions for both problems are returned. The relevance of each output therefore depends on the problem being solved. The following object of class `sqlp_output` is returned upon completion

The examples in subsequent subsections will demonstrate the output provided by `sqlp`.

`pobj`	the value of the primary objective function
`dobj`	the value of the dual objective function
`X`	A matrix object containing the optimal matrix $\mathbf{X}$ for the primary problem
`y`	A vector object containing the optimal vector $\mathbf{y}$ for the dual problem
`Z`	A matrix object containing the optimal matrix $\mathbf{Z}$ for the dual problem

C.4.2 The Max-Cut Problem

Recall that the maximum cut of a graph $\mathbf{G}$ with adjacency matrix $\mathbf{B}$ can be found as the solution to

$$\begin{array}{ll} \text{Minimize} & \langle \mathbf{C}, \mathbf{X} \rangle \\ \text{subject to} & \\ & \begin{array}{rcl} \mathit{diag}(\mathbf{X}) & = & \mathbf{1} \\ \mathbf{X} & \in & \mathcal{S}^n \end{array} \end{array}$$

where $\mathbf{C} = -(\mathit{diag}(\mathbf{B1}) - \mathbf{B})/4$. In Section C.1, we wrote this in the form of an SQLP

$$\begin{array}{ll} \text{Minimize} & \langle \mathbf{C}, \mathbf{X} \rangle \\ \text{subject to} & \\ & \begin{array}{rclcrcl} \langle \mathbf{A}_k, \mathbf{X} \rangle & = & 1, & & k & = & 1, \ldots, n \\ \mathbf{X} & \in & \mathcal{S}^n & & & & \end{array} \end{array}$$

where we defined $\mathbf{A}_k$ as

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

To convert this to a form usable by `sqlp`, we begin by noting that we have one optimization variable, $\mathbf{X}$. Therefore, with $L = 1$, we initialize the required input variables as follows

```
R> blk <- matrix(list(), nrow = 1, ncol = 2)
R> At <- matrix(list(), nrow = 1, ncol = 1)
R> C <- matrix(list(), nrow = 1, ncol = 1)
```

This initializes `blk`, `At`, and `C` as a matrix of matrices. While not required mathematically in this problem, this is the format required for all problems solved using `sqlp`, and will be required for any problem with more than one optimization variable.

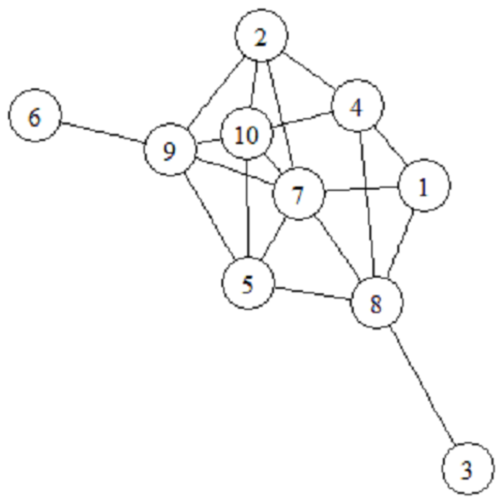
This initialization has the advantage of allowing `blk` to contain character and numerical values, without using the overhead of a data frame. Having $\mathbf{X}$ constrained to the space of semidefinite matrices of size n, we specify `blk` as

```
R> blk[[1,1]] <- "s"
R> blk[[1,2]] <- n
```

With the objective function in the form $\langle \mathbf{C}, \mathbf{X} \rangle$, we define the input `C` as

```
R> one <- matrix(1, nrow = n, ncol = 1)
R> C[[1, 1]] <- -(diag(B %*% one) - B) / 4
```

where $\mathbf{B}$ is the adjacency matrix for a graph on which we would like to find the maximum cut, such as the one in Figure C.1.



$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure C.1: *A graph object and associated adjacency matrix for which we would like to find the maximum cut.*

The matrix `At` is constructed using the upper triangular portion of the $\mathbf{A}_k$ matrices. To do this in `R`, the function `svec` is made available in *sdpt3r*.

```
R> A <- matrix(list(), nrow = 1, ncol = n)
R> for(k in 1:n){
R>   A[[k]] <- matrix(0, nrow = n, ncol = n)
R>   diag(A[[k]])[k] <- 1
R> }

R> At[[1, 1]] <- svec(blk[1, ], A)
```

Having each of the diagonal elements of $\mathbf{X}$ constrained to be 1, `b` is a $n \times 1$ matrix of ones

```
R> b <- matrix(1, nrow = n, ncol = 1)
```

With all the input variables now defined, we can now call `sqlp` to solve the Max-Cut problem

```
R> sqlp(blk, At, C, b)
```

C.4.3 A Numerical Example and the maxcut Function

The built-in function `maxcut` takes as input a (weighted) adjacency matrix `B` and returns an S3 object containing the input necessary for `sqlp`. This object can be passed directly to `sqlp` using `sqlp_obj`, or each input variable can be passed individually. If we wish to find to the maximum cut of the graph in Figure C.1, given the adjacency matrix $\mathbf{B}$ we can compute the input variables for `sqlp` using `maxcut`

```
R> out <- maxcut(B)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b

R> sqlp(blk, At, C, b)
R> sqlp(sqlp_obj = out)

$pobj

[1] -14.67622

$X

      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]  [,10]
V1   1.000  0.987 -0.136 -0.858  0.480  0.857 -0.879  0.136 -0.857  0.597
V2   0.987  1.000  0.026 -0.763  0.616  0.929 -0.791 -0.026 -0.929  0.459
V3  -0.136  0.026  1.000  0.626  0.804  0.394  0.592 -1.000 -0.394 -0.876
V4  -0.858 -0.763  0.626  1.000  0.039 -0.469  0.999 -0.626  0.470 -0.925
V5   0.480  0.616  0.804  0.039  1.000  0.864 -0.004 -0.804 -0.864 -0.417
V6   0.857  0.929  0.394 -0.469  0.864  1.000 -0.508 -0.394 -1.000  0.098
V7  -0.879 -0.791  0.592  0.999 -0.004 -0.508  1.000 -0.592  0.508 -0.907
```

```
V8    0.136 -0.026 -1.000 -0.626 -0.804 -0.394 -0.592  1.000  0.394  0.876
V9   -0.857 -0.929 -0.394  0.470 -0.864 -1.000  0.508  0.394  1.000 -0.098
V10   0.597  0.459 -0.876 -0.925 -0.417  0.098 -0.907  0.876 -0.098  1.000
```

Note that the value of the primary objective function is negative as we have defined $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$ since we require the primal formulation to be a minimization problem. The original formulation given in [45] frames the Max-Cut problem as a maximization problem with $\mathbf{C} = (diag(\mathbf{B1}) - \mathbf{B})/4$. Therefore, the approximate value of the maximum cut for the graph in Figure C.1 is 14.68 (recall we are solving a relaxation).

As an interesting aside, we can show that the matrix $\mathbf{X}$ is actually a correlation matrix by considering its eigenvalues - we can see it clearly is symmetric, with unit diagonal and all elements in [-1,1].

```
R> eigen(X)

$values

 [1] 5.59e+00 4.41e+00 2.07e-07 1.08e-07 4.92e-08 3.62e-08 3.22e-08
 [8] 1.90e-08 1.66e-08 9.38e-09
```

The fact that $\mathbf{X}$ is indeed a correlation matrix comes as no surprise. [45] show that the set of feasible solutions for the Max-Cut problem is in fact the set of correlation matrices. So while we may not be interested in $\mathbf{X}$ as an output for solving the Max-Cut problem, it is nonetheless interesting to see that it is in fact in the set of feasible solutions.

C.4.4 Nearest Correlation Matrix

Recall that the nearest correlation matrix is found as the solution to

$$\begin{array}{ll}\underset{\mathbf{e}^*,\ \mathbf{X}}{\text{minimize}} & e_0 \\ \text{subject to} & \\ & \begin{array}{rcl} svec(\mathbf{R}) - svec(\mathbf{X}) & = & [\mathbf{0}, \mathbf{I}_{n(n+1)/2}]\ \mathbf{e}^* \\ diag(\mathbf{X}) & = & \mathbf{1} \\ \mathbf{X} & \in & \mathcal{S}^n \\ \mathbf{e}^* & \in & \mathcal{Q}^{n(n+1)/2+1} \end{array}\end{array}$$

In Section C.2 we wrote this as the following SQLP

$$\begin{array}{ll}\underset{\mathbf{e}^*,\,\mathbf{X}}{\text{minimize}} & \langle \mathbf{C}, \mathbf{e}^* \rangle \\ \text{subject to} & \\ & \begin{array}{rcl}(\mathbf{A}^s)^\mathsf{T} svec(\mathbf{X}) + (\mathbf{A}^q)^\mathsf{T}\mathbf{e}^* & = & \mathbf{b} \\ \mathbf{X} & \in & \mathcal{S}^n \\ \mathbf{e}^* & \in & \mathcal{Q}^{n(n+1)/2+1}\end{array}\end{array}$$

for $\mathbf{C} = [1, \mathbf{0}_{n(n+1)/2}]$, and

$$\begin{array}{rcl}\mathbf{A}^s & = & [\mathbf{A}_1^s,\ \mathbf{A}_2^s] \\ \mathbf{A}^q & = & [\mathbf{A}_1^q,\ \mathbf{A}_2^q]\end{array} \qquad \mathbf{b} = [\mathbf{b}_1;\ \mathbf{b}_2]$$

where

$$\begin{array}{rcl}\mathbf{A}_1^s & = & \mathbf{I}_{n_2} \\ \mathbf{A}_1^q & = & [\mathbf{0}, \mathbf{I}_{n_2}] \\ & & \\ \mathbf{A}_2^s & = & [svec(\mathbf{A}_1), \ldots, svec(\mathbf{A}_n)] \\ \mathbf{A}_2^q & = & \mathbf{0}_{n_2}\end{array} \qquad \begin{array}{rcl}\mathbf{b}_1 & = & svec(\mathbf{R}) \\ \mathbf{b}_2 & = & \mathbf{1}^\mathsf{T}\end{array}$$

and $\mathbf{A}_1, \ldots, \mathbf{A}_n$ are given by

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

To be solved using `sqlp`, we first define `blk`. There are two optimization variables in the formulation of the nearest correlation matrix - $\mathbf{X}$ is an $n \times n$ matrix constrained to be in a semidefinite cone, and $\mathbf{y}$ is an $n(n+1)/2 + 1$ length vector constrained to be in a quadratic cone, so

```
R> blk <- matrix(list(), nrow = 2, ncol = 2)

R> blk[[1, 1]] <- "s"
R> blk[[1, 2]] <- n

R> blk[[2, 1]] <- "q"
R> blk[[2, 2]] <- n * (n + 1) / 2 + 1
```

Note that $\mathbf{X}$ does not appear in the objective function, so the `C` entry corresponding to the block variable $\mathbf{X}$ is an $n \times n$ matrix of zeros, which defines `C` as

```
R> C <- matrix(list(), nrow = 2, ncol = 1)
```

```
R> C[[1, 1]] <- matrix(0, nrow = n, ncol = n)
R> C[[2, 1]] <- rbind(1, matrix(0, nrow = n2, ncol = 1))
```

Next comes the constraint matrix for $\mathbf{X}$

```
R> At <- matrix(list(), nrow = 2, ncol = 1)

R> A1s <- diag(1, nrow = n2, ncol = n2)
R>
R> Aks <- matrix(list(), nrow = 1, ncol = n)
R> for(k in 1:n){
R>   Aks[[k]] <- matrix(0, nrow = n, ncol = n)
R>   diag(Aks[[k]])[k] <- 1
R> }

R> A2s <- svec(blk[1, ], Aks)

R> At[[1, 1]] <- cbind(A1s, A2s)
```

then the constraint matrix for $\mathbf{e}^*$.

```
R> A1q <- matrix(0, nrow = n, ncol = n2 + 1)

R> A2q1 <- matrix(0, nrow = n2, ncol = 1)
R> A2q2 <- diag(1, nrow = n2, ncol = n2)
R> A2q <- cbind(A211, A212)

R> At[[2, 1]] <- rbind(A1q, A2q)
```

and the right hand side vector `b`

```
R> b <- rbind(svec(blk[1, ], R), matrix(1, n, 1))
```

The nearest correlation matrix problem is now solved by

```
R> sqlp(blk, At, C, b)
```

C.4.5 A Numerical Example and the nearcorr Function

To demonstrate the nearest correlation matrix problem, we will modify an existing correlation matrix by exploring the effect of changing the sign of just one of the pairwise correlations. In the context of stock correlations, we make use of tools available in the R package *quantmod* [94] to access stock data from five tech firms (Microsoft, Apple, Amazon, Alphabet/Google, and IBM) beginning in 2007.

```
R> library("quantmod")

R> getSymbols(c("MSFT", "AAPL", "AMZN", "GOOGL", "IBM"))
R> stock.close <- as.xts(merge(MSFT, AAPL, AMZN,
+    GOOGL, IBM))[, c(4, 10, 16, 22, 28)]
```

The correlation matrix for these five stocks is

```
R> stock.corr <- cor(stock.close)
R> stock.corr

            MSFT.Close AAPL.Close AMZN.Close GOOGL.Close IBM.Close
MSFT.Close   1.0000000 -0.2990463  0.9301085   0.5480033 0.2825698
AAPL.Close  -0.2990463  1.0000000 -0.1514348   0.3908624 0.6887127
AMZN.Close   0.9301085 -0.1514348  1.0000000   0.6228299 0.3870390
GOOGL.Close  0.5480033  0.3908624  0.6228299   1.0000000 0.5885146
IBM.Close    0.2825698  0.6887127  0.3870390   0.5885146 1.0000000
```

Next, consider the effect of having a positive correlation between Microsoft and Apple

```
R> stock.corr[1, 2] <- -1 * stock.corr[1, 2]
R> stock.corr[2, 1] <- stock.corr[1, 2]
R> stock.corr

            MSFT.Close AAPL.Close AMZN.Close GOOGL.Close IBM.Close
MSFT.Close   1.0000000  0.2990463  0.9301085   0.5480033 0.2825698
AAPL.Close   0.2990463  1.0000000 -0.1514348   0.3908624 0.6887127
AMZN.Close   0.9301085 -0.1514348  1.0000000   0.6228299 0.3870390
GOOGL.Close  0.5480033  0.3908624  0.6228299   1.0000000 0.5885146
IBM.Close    0.2825698  0.6887127  0.3870390   0.5885146 1.0000000
```

Unfortunately, this correlation matrix is not positive semidefinite

```
R> eigen(stock.corr)$values

[1]  2.8850790  1.4306393  0.4902211  0.3294150 -0.1353544
```

Given the approximate correlation matrix `stock.corr`, the built-in function `nearcorr` returns an S3 object containing the input necessary for `sqlp`. This object can be passed directly to `sqlp` using `sqlp_obj`, or each input variable can be passed individually.

```
R> out <- nearcorr(stock.corr)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b

R> sqlp(blk, At, C, b)
R> sqlp(sqlp_obj = out)
```

Since this is a minimization problem, and thus a primal formulation of the SQLP, the output `X` from `sqlp` will provide the optimal solution to the problem - that is, `X` will be the nearest correlation matrix to `stock.corr`.

```
$X

          [,1]        [,2]        [,3]      [,4]      [,5]
[1,] 1.0000000  0.25388359  0.86150833 0.5600734 0.3126420
[2,] 0.2538836  1.00000000 -0.09611382 0.3808981 0.6643566
[3,] 0.8615083 -0.09611382  1.00000000 0.6115212 0.3480430
[4,] 0.5600734  0.38089811  0.61152116 1.0000000 0.5935021
[5,] 0.3126420  0.66435657  0.34804303 0.5935021 1.0000000
```

The matrix above is symmetric with unit diagonal and all entries in $[-1, 1]$. By checking the eigenvalues,

```
eigen(X)$values

[1] 2.846016e+00 1.384062e+00 4.570408e-01 3.128807e-01 9.680507e-11
```

we can see that `X` is indeed a correlation matrix.

C.4.6 D-Optimal Experimental Design

Recall from Section C.3 that the D-Optimal experimental design problem was stated as the following dual SQLP

$$\begin{array}{ll}\underset{\mathbf{Z},\ \mathbf{z}^l,\ \boldsymbol{\lambda}}{\text{maximize}} & \log\det\left(\mathbf{Z}\right)\\ \text{subject to} & \\ & \begin{array}{rcll} -\sum_{i=1}^{p}\lambda_i(\mathbf{u}_i\mathbf{u}_i^{\mathsf{T}})+\mathbf{Z} & = & 0, & \mathbf{Z}\in\mathcal{S}^n\\ -\boldsymbol{\lambda}+\mathbf{z}^l & = & 0, & \mathbf{z}^l\in\mathcal{R}^p_+\\ \mathbf{1}^T\boldsymbol{\lambda} & = & 1, & \boldsymbol{\lambda}\in\mathcal{R}^p\end{array}\end{array}$$

which we wrote as

$$\begin{array}{ll}\underset{\mathbf{Z},\ \mathbf{z}^l,\ \boldsymbol{\lambda}}{\text{maximize}} & \log\det\left(\mathbf{Z}\right)\\ \text{subject to} & \\ & \begin{array}{rclcll} (\mathbf{A}^s)^{\mathsf{T}}\boldsymbol{\lambda} & + & \mathbf{Z} & = & \mathbf{C}^s, & \mathbf{Z}\in\mathcal{S}^n\\ (\mathbf{A}^l)^{\mathsf{T}}\boldsymbol{\lambda} & + & \mathbf{z}^l & = & \mathbf{C}^q, & \mathbf{z}^l\in\mathcal{R}^p_+\\ (\mathbf{A}^u)^{\mathsf{T}}\boldsymbol{\lambda} & & & = & \mathbf{C}^u, & \boldsymbol{\lambda}\in\mathcal{R}^p\end{array}\end{array}$$

where $\mathbf{b}=\mathbf{0}$, and

$$\begin{array}{rlcrl}\mathbf{A}^s & = -[svec(\mathbf{A}_1),\ldots,svec(\mathbf{A}_p)] & \qquad & \mathbf{C}^s & = \mathbf{0}_{n\times n}\\ \mathbf{A}^l & = -\mathbf{I}_p & & \mathbf{C}^l & = \mathbf{0}_{p\times 1}\\ \mathbf{A}^u & = \mathbf{1}^{\mathsf{T}} & & \mathbf{C}^u & = 1\end{array}$$

Here, $\mathbf{A}_1,\ldots,\mathbf{A}_p$ are given by

$$\mathbf{A}_i=\mathbf{u}_i\mathbf{u}_i^{\mathsf{T}},\quad i=1,\ldots,p$$

To convert this to a form usable by *sdpt3r*, we initialize our input variables by noting we have three blocks - $\mathbf{X}$, $\mathbf{z}^l$, and $\boldsymbol{\lambda}$

```
R> blk <- matrix(list(), nrow = 3, ncol = 2)
R> At <- matrix(list(), nrow = 3, ncol = 1)
R> C <- matrix(list(), nrow = 3, ncol = 1)
```

As before, we declare the three blocks in `blk`. The first block is semidefinite containing the matrix $\mathbf{Z}$, the second a linear block containing $\mathbf{z}^l$, and the third an unrestricted block containing $\boldsymbol{\lambda}$

```
R> blk[[1, 1]] <- "s"
R> blk[[1, 2]] <- n

R> blk[[2, 1]] <- "l"
R> blk[[2, 2]] <- p

R> blk[[3, 1]] <- "u"
R> blk[[3, 2]] <- 1
```

Next, by noting the variable $\boldsymbol{\lambda}$ does not appear in the objective function, we specify `b` as a vector of zeros

```
R> b <- matrix(0, nrow = p, ncol = 1)
```

Next, looking at the right-hand side of the constraints, we define the matrices `C`

```
R> C[[1, 1]] <- matrix(0, nrow = n, ncol = n)
R> C[[2, 1]] <- matrix(0, nrow = p, ncol = 1)
R> C[[3, 1]] <- 1
```

Finally, we construct `At` for each variable

```
R> A <- matrix(list(), nrow = p, ncol = 1)

R> for(k in 1:p){
R>   A[[k]] <- -uk %*% t(uk)
R> }

R> At[[1, 1]] <- svec(blk[1, ], A)
R> At[[2, 1]] <- diag(-1, nrow = p, ncol = p)
R> At[[3, 1]] <- matrix(1, nrow = 1, ncol = p)
```

The final hurdle necessary to address in this problem is the existence of the log-barrier. Recall that it is assumed that v^s, v^q, and v^l in Equation (C.4) are all zero in `OPTIONS`. In this case, we can see that is not true, as we have a log term containing $\mathbf{Z}$ in the objective function, meaning v^s is equal to one. To pass this to `sqlp`, we define the `OPTIONS$parbarrier` variable as

```
R> OPTIONS$parbarrier <- matrix(list(), nrow = 3, ncol = 1)
R> OPTIONS$parbarrier[[1]] <- 1
R> OPTIONS$parbarrier[[2]] <- 0
R> OPTIONS$parbarrier[[3]] <- 0
```

The D-Optimal experimental design problem can now be solved using `sqlp`

```
R> sqlp(blk, At, C, b, OPTIONS)
```

C.4.7 A Numerical Example and the doptimal Function

To demonstrate the output generated from a D-optimal experimental design problem, we consider a simple 3×25 matrix containing the known test vectors $\mathbf{u}_1, ..., \mathbf{u}_{25}$ (the data is available in the `sqlp` package). To generate the required input for `sqlp`, we use the function `doptimal`, which takes as input an $n \times p$ matrix $\mathbf{U}$ containing the known test vectors, and returns an S3 object containing the input necessary for `sqlp`. This object can be passed directly to `sqlp` using `sqlp_obj`, or each input variable can be passed individually. The output we are interested in is `y`, corresponding to $\boldsymbol{\lambda}$ in our formulation, the percentage of each $\mathbf{u}_i$ necessary to achieve maximum information in the experiment.

```
R> data("DoptDesign")

R> out <- doptimal(DoptDesign)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)

$y
       [,1]
 [1,] 0.000
 [2,] 0.000
 [3,] 0.000
 [4,] 0.000
 [5,] 0.000
 [6,] 0.000
 [7,] 0.154
 [8,] 0.000
 [9,] 0.000
[10,] 0.000
[11,] 0.000
[12,] 0.000
```

```
[13,] 0.319
[14,] 0.000
[15,] 0.000
[16,] 0.240
[17,] 0.000
[18,] 0.000
[19,] 0.000
[20,] 0.000
[21,] 0.000
[22,] 0.000
[23,] 0.287
[24,] 0.000
[25,] 0.000
```

The information matrix $\mathbf{A}^{\mathsf{T}}\mathbf{A}$ is a linear combination of the test vectors $\mathbf{u}_i$, weighted by the optimal vector `y` above.

C.5 Additional Problems

The *sdpt3r* package considerably broadens the set of optimization problems that can be solved in `R`. In addition to those problems presented in detail in Section C.4, there are a large number of well known problems that can also be formulated as an SQLP.

Each problem presented will be described briefly, with appropriate references for the interested reader, and presented mathematically in its classical form, not as an SQLP as in Equation (C.3) or (C.4). Accompanying each problem will be an `R` helper function, which will produce an S3 object containing the input variables `blk`, `At`, `C`, and `b` so that the problem can be solved using `sqlp`. This object can be given directly to `sqlp` to solve the problem, or the individual input variables can be extracted. Each method is presented. Each helper function is made available to the user in the *sdpt3r* package. Each of these problems were originally presented in [105], and have simply been transcribed here.

C.5.1 Minimum Volume Ellipsoids

The problem of finding the ellipsoid of minimum volume containing a set of points $\mathbf{v}_1, ..., \mathbf{v}_n$ is stated as the following optimization problem [114]

$$\begin{array}{ll} \underset{\mathbf{B},\ \mathbf{d}}{\text{maximize}} & \mathit{log}\ \mathit{det}(\mathbf{B}) \\ \text{subject to} & \\ & ||\mathbf{Bx}+\mathbf{d}|| \ \le \ 1, \quad \forall\ ]vex \ \in \ [\mathbf{v}_1, ..., \mathbf{v}_n] \end{array}$$

The function `minelips` takes as input an $n \times p$ matrix $\mathbf{V}$ containing the points around which we would like to find the minimum volume ellipsoid, and returns the input variables necessary to solve the problem using `sqlp`.

```
R> out <- minelips(V)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

C.5.2 Distance Weighted Discrimination

Given two sets of points in a matrix $\mathbf{X} \in \mathcal{R}^n$ with associated class variables [-1,1] in $\mathbf{Y} = \mathit{diag}(\mathbf{y})$, distance weighted discrimination [75] seeks to classify the points into two distinct subsets by finding a hyperplane between the two sets of points. Mathematically, the distance weighted discrimination problem seeks a hyperplane defined by a normal vector, $\boldsymbol{\omega}$, and position, β, such that each element in the residual vector $\bar{\mathbf{r}} = \mathbf{YX}^\mathsf{T}\boldsymbol{\omega} + \beta\mathbf{y}$ is positive and large. Since the class labels are either 1 or -1, having the residuals be positive is equivalent to having the points on the proper side of the hyperplane.

Of course, it may be impossible to have a perfect separation of points using a linear hyperplane, so an error term ξ is introduced. Thus, the perturbed residuals are defined to be

$$\mathbf{r} = \mathbf{YX}^\mathsf{T}\boldsymbol{\omega} + \beta\mathbf{y} + \boldsymbol{\xi}$$

Distance weighted discrimination [75] solves the following optimization problem to find the optimal hyperplane.

$$
\begin{array}{ll}
\underset{\mathbf{r},\ \boldsymbol{\omega},\ \beta,\ \boldsymbol{\xi}}{\text{minimize}} & \sum_{i=1}^{n}(1/r_i) + C\mathbf{1}^{\mathsf{T}}\boldsymbol{\xi} \\
\text{subject to} & \\
& \begin{array}{rcl}
\mathbf{r} & = & \mathbf{Y}\mathbf{X}^{\mathsf{T}}\boldsymbol{\omega} + \beta\mathbf{y} + \boldsymbol{\xi} \\
\boldsymbol{\omega}^{\mathsf{T}}\boldsymbol{\omega} & \leq & 1 \\
\mathbf{r} & \geq & \mathbf{0} \\
\boldsymbol{\xi} & \geq & \mathbf{0}
\end{array}
\end{array}
$$

where $C > 0$ is a penalty parameter to be chosen.

The function `dwd` takes as input two $n \times p$ matrices `X1` and `X2` containing the points to be separated, as well as a penalty term `C` ≥ 0 penalizing the movement of a point on the wrong side of the hyperplane to the proper side, and returns the input variables necessary for `sqlp` to solve the distance weighted discrimination problem.

```
R> out <- dwd(X1, X2, C)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

C.5.3 Max-kCut

Similar to the Max-Cut problem, the Max-kCut problem asks, given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and an integer k, does a cut exist of at least size k. For a given (weighted) adjacency matrix $\mathbf{B}$ and integer k, the Max-kCut problem is formulated as the following primal problem

$$
\begin{array}{ll}
\underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C},\ \mathbf{X} \rangle \\
\text{subject to} & \\
& \begin{array}{rcll}
diag(\mathbf{X}) & = & \mathbf{1} & \\
X_{ij} & \geq & 1/(k-1) & \forall\, i \neq j \\
\mathbf{X} & \in & \mathcal{S}_n &
\end{array}
\end{array}
$$

Here, $\mathbf{C} = -(1 - 1/k)/2 * (diag(\mathbf{B1}) - \mathbf{B})$. The Max-kCut problem is slightly more complex than the Max-Cut problem due to the inequality constraint. In order to turn this into a standard

SQLP, we must replace the inequality constraints with equality constraints, which we do by introducing a slack variable $\mathbf{x}^l$, allowing the problem to be restated as

$$\begin{array}{lrcll} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C},\ \mathbf{X} \rangle & & & \\ \text{subject to} & & & & \\ & diag(\mathbf{X}) & = & \mathbf{1} & \\ & X_{ij} - x^l & = & 1/(k-1) & \forall\ i \neq j \\ & \mathbf{X} & \in & \mathcal{S}^n & \\ & \mathbf{x}^l & \in & \mathcal{L}^{n(n+1)/2} & \end{array}$$

The function `maxkcut` takes as input an adjacency matrix `B` and an integer `k`, and returns the input variables necessary for the problem to be solved using `sqlp`.

```
R> out <- maxkcut(B, k)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

C.5.4 The Graph Partitioning Problem

The graph partitioning problem can be formulated as the following primal optimization problem

$$\begin{array}{lrcl} \underset{\mathbf{X}}{\text{minimize}} & tr(\mathbf{CX}) & & \\ \text{subject to} & & & \\ & tr(\mathbf{11}^{\mathsf{T}}\mathbf{X}) & = & \alpha \\ & diag(\mathbf{X}) & = & \mathbf{1} \end{array}$$

Here, $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})$, for an adjacency matrix $\mathbf{B}$, and α is any real number.

The function `gpp`, takes as input a weighted adjacency matrix `B` and a real number `alpha` and returns the input necessary to solve the problem using `sqlp`.

```
R> out <- gpp(B, alpha)
R> blk <- out$blk
R> At <- out$At
```

```
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

C.5.5 The Lovasz Number

The Lovasz Number of a graph $\mathbf{G}$, denoted $\vartheta(\mathbf{G})$, is the upper bound on the Shannon capacity of the graph. For an adjacency matrix $\mathbf{B} = [B_{ij}]$ the problem of finding the Lovasz number is given by the following primal SQLP problem

$$
\begin{array}{lrcl}
\underset{\mathbf{X}}{\text{minimize}} & tr(\mathbf{CX}) & & \\
\text{subject to} & & & \\
 & tr(\mathbf{X}) & = & 1 \\
 & X_{ij} & = & 0 \quad \text{if } B_{ij} = 1 \\
 & \mathbf{X} & \in & \mathcal{S}^n
\end{array}
$$

The function `lovasz` takes as input an adjacency matrix $\mathbf{B}$, and returns the input variables necessary for the Lovasz number to be found using `sqlp`.

```
R> out <- lovasz(B)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

Toeplitz Approximation

Given a symmetric matrix $\mathbf{F}$, the Toeplitz approximation problem seeks to find the nearest symmetric positive definite Toeplitz matrix. In general, a Toeplitz matrix is one with constant descending diagonals, i.e.,

$$\mathbf{T} = \begin{bmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{bmatrix}$$

is a general Toeplitz matrix. The problem is formulated as the following optimization problem

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{maximize}} & -y_{n+1} \\ \text{subject to} & \\ & \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & -\beta \end{bmatrix} + \sum_{k=1}^{n} y_k \begin{bmatrix} \mathbf{0} & \gamma_k \mathbf{e}_k \\ \gamma_k \mathbf{e}_k^T & -2q_k \end{bmatrix} + y_{n+1}\mathbf{B} \geq \mathbf{0} \\ & [y_1, ..., y_n]^{\mathsf{T}} + y_{n+1}\mathbf{B} \geq \mathbf{0} \end{array}$$

where $\mathbf{B}$ is an $(n+1) \times (n+1)$ matrix of zeros, and $\mathbf{B}_{(n+1)(n+1)} = 1$, $q_1 = -tr(\mathbf{F})$, q_k = sum of k^{th} diagonal upper and lower triangular matrix, $\gamma_1 = \sqrt{n}$, $\gamma_k = \sqrt{2 * (n-k+1)}$, $k = 2, ..., n$, and $\beta = ||\mathbf{F}||_F^2$.

The function `toep` takes as input a symmetric matrix `F` for which we would like to find the nearest Toeplitz matrix, and returns the input variables required to solve the problem using `sqlp`.

```
R> out <- toep(F)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

C.5.6 The Educational Testing Problem

The educational testing problem arises in measuring the reliability of a student's total score in an examination consisting of a number of sub-tests [38]. In terms of formulation as an optimization problem, the problem is to determine how much can be subtracted from the diagonal of a given symmetric positive definite matrix $\mathbf{S}$ such that the resulting matrix remains positive semidefinite [21].

The Educational Testing Problem (ETP) is formulated as the following dual problem

$$
\begin{array}{ll}
\underset{\mathbf{d}}{\text{maximize}} & \mathbf{1}^{\mathsf{T}}\mathbf{d} \\
\text{subject to} & \\
& \begin{array}{rcl} \mathbf{A} - diag(\mathbf{d}) & \succeq & \mathbf{0} \\ \mathbf{d} & \geq & \mathbf{0} \end{array}
\end{array}
$$

where $\mathbf{d} = [d_1,\ d_2, ...,\ d_n]$ is a vector of size n and $diag(\mathbf{d})$ denotes the corresponding $n \times n$ diagonal matrix. In the second constraint, having each element in $\mathbf{d}$ be greater than or equal to 0 is equivalent to having $diag(\mathbf{d}) \succeq 0$.

The corresponding primal problem is

$$
\begin{array}{ll}
\underset{\mathbf{X}}{\text{minimize}} & tr(\mathbf{AX}) \\
\text{subject to} & \\
& \begin{array}{rcl} diag(\mathbf{X}) & \geq & \mathbf{1} \\ \mathbf{X} & \succeq & 0 \end{array}
\end{array}
$$

The function `etp` takes as input an $n \times n$ positive definite matrix `A`, and returns the input variables required to solve the educational testing problem using `sqlp`.

```
R> out <- etp(A)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

C.5.7 Logarithmic Chebyshev Approximation

For a $p \times n$ ($p > n$) matrix $\mathbf{B}$ and $p \times 1$ vector $\mathbf{f}$, the Logarithmic Chebyshev Approximation problem is stated as the following optimization problem [114]

$$
\begin{array}{ll}
\underset{\mathbf{x},\ t}{\text{minimize}} & t \\
\text{subject to} & \\
& 1/t \ \leq \ (\mathbf{x}^{\mathsf{T}}\mathbf{B}_{i\cdot})/\mathbf{f}_i \ \leq \ t, \quad i = 1, ..., p
\end{array}
$$

where $\mathbf{B}_{i\cdot}$ denotes the i^{th} row of the matrix $\mathbf{B}$. Note that we require each element of $\mathbf{B}_{\cdot j}/\mathbf{f}$ to be greater than or equal to 0 for all j.

The function `logcheby` takes as input a matrix `B` and vector `f`, and returns the input variables necessary to solve the Logarithmic Chebyshev Approximation problem using `sqlp`.

```
R> out <- logcheby(B, f)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

C.5.8 Linear Matrix Inequality Problems

We consider three distinct linear matrix inequality problems, all written in the form of a dual optimization problem. The first linear matrix inequality problem we will consider is defined by the following optimization equation for some $n \times p$ matrix $\mathbf{B}$ known in advance

$$\begin{array}{ll} \underset{\eta,\ \mathbf{Y}}{\text{maximize}} & -\eta \\ \text{subject to} & \\ & \begin{array}{rcl} \mathbf{B}\mathbf{Y} + \mathbf{Y}\mathbf{B}^{\mathsf{T}} & \preceq & 0 \\ -\mathbf{Y} & \preceq & -\mathbf{I} \\ \mathbf{Y} - \eta\mathbf{I} & \preceq & 0 \\ Y_{11} & = & 1, \quad \mathbf{Y} \in \mathcal{S}^n \end{array} \end{array}$$

The function `lmi1` takes as input a matrix $\mathbf{B}$, and returns the input variables `blk`, `At`, `C`, and `b` for `sqlp`.

```
R> out <- lmi1(B)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
```

```
R> sqlp(sqlp_obj = out)
```

The second linear matrix inequality problem is

$$\begin{array}{ll} \underset{\mathbf{P},\ \mathbf{d}}{\text{maximize}} & -tr(\mathbf{P}) \\ \text{subject to} & \\ & \begin{array}{rcl} \mathbf{A}_1\mathbf{P} + \mathbf{P}{\mathbf{A}_1}^{\mathsf{T}} + \mathbf{B} * diag(\mathbf{d}) * \mathbf{B}^{\mathsf{T}} & \preceq & 0 \\ \mathbf{A}_2\mathbf{P} + \mathbf{P}{\mathbf{A}_2}^{\mathsf{T}} + \mathbf{B} * diag(\mathbf{d}) * \mathbf{B}^{\mathsf{T}} & \preceq & 0 \\ -\mathbf{d} & \preceq & 0 \\ \sum_i^p d_i & = & 1 \end{array} \end{array}$$

Here, the matrices $\mathbf{B}$, $\mathbf{A}_1$, and $\mathbf{A}_2$ are known in advance.

The function `lmi2` takes the matrices `A1`, `A2`, and `B` as input, and returns the input variables necessary for `sqlp`.

```
R> out <- lmi2(A1, A2, B)
R> blk <- out$blk
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

The final linear matrix inequality problem originates from a problem in control theory [11] and requires three matrices be known in advance, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{G}$

$$\begin{array}{ll} \underset{\eta,\ \mathbf{P}}{\text{maximize}} & \eta \\ \text{subject to} & \\ & \begin{bmatrix} \mathbf{AP} + \mathbf{PA}^{\mathsf{T}} & \mathbf{0} \\ \mathbf{BP} & \mathbf{0} \end{bmatrix} + \eta \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \preceq \begin{bmatrix} -\mathbf{G} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \end{array}$$

The function `lmi3` takes as input the matrices `A`, `B`, and `G`, and returns the input variables necessary to solve the problem using `sqlp`.

```
R> out <- lmi3(A, B, G)
R> blk <- out$blk
```

```
R> At <- out$At
R> C <- out$C
R> b <- out$b
R> OPTIONS <- out$OPTIONS

R> sqlp(blk, At, C, b, OPTIONS)
R> sqlp(sqlp_obj = out)
```

C.5.9 OPTIONS

`vers`	specifies the search direction 0, HKM if semidefinite blocks present, NT otherwise (default) 1, HKM direction 2, NT direction
`predcorr`	TRUE, use Mehrotra prediction-correction (default) FALSE, otherwise
`gam`	step-length (default 0)
`expon`	exponent used to decrease sigma (default 1)
`gaptol`	tolerance for duality gap as a fraction of the objective function (default $1e-8$)
`inftol`	tolerance for stopping due to infeasibility (default 1e-8)
`steptol`	tolerance for stopping due to small steps (default 1e-6)
`maxit`	maximum number of iterations (default 100)
`stoplevel`	0, continue until successful completion, maximum iteration, or numerical failure 1, automatically detect termination, restart if small steps is cause (default) 2, automatically detect termination
`scale_data`	TRUE, scale data prior to solving FALSE, otherwise (default)
`rmdepconstr`	TRUE, remove nearly dependent constraints FALSE, otherwise (default)
`parbarrier`	declare the existence of a log barrier term default value is 0 (i.e., no log barrier)