A new framework for clustering

by

Wu Zhou

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, 2010

Abstract

The difficulty of clustering and the variety of clustering methods suggest the need for a theoretical study of clustering. Using the idea of a standard statistical framework, we propose a new framework for clustering.

For a well-defined clustering goal we assume that the data to be clustered come from an underlying distribution and we aim to find a high-density cluster tree. We regard this tree as a parameter of interest for the underlying distribution. However, it is not obvious how to determine a connected subset in a discrete distribution whose support is located in a Euclidean space. Building a cluster tree for such a distribution is an open problem and presents interesting conceptual and computational challenges. We solve this problem using graph-based approaches and further parameterize clustering using the high-density cluster tree and its extension.

Motivated by the connection between clustering outcomes and graphs, we propose a graph family framework. This framework plays an important role in our clustering framework. A direct application of the graph family framework is a new cluster-tree distance measure. This distance measure can be written as an inner product or kernel. It makes our clustering framework able to perform statistical assessment of clustering via simulation. Other applications such as a method for integrating partitions into a cluster tree and methods for cluster tree averaging and bagging are also derived from the graph family framework.

Acknowledgements

I would like to thank my supervisor, Dr. Wayne Oldford, for his advice, encouragement, and help that kept me moving forward. Working with him has been a wonderful experience. I would also like to thank Dr. Werner Stuetzle, Dr. Paul Marriott, Dr. Ali Ghodsi and Dr. Fakhri Karray for their valuable comments and suggestions. I am very grateful to my wife, Cathy Wang, for her understanding and her belief in me. I would also like to thank my parents, Shangyuan Zhou and Yifang Wu, for their support.

Contents

Chapter 1

Introduction

The purpose of clustering is to separate data into different groups such that similar data objects are assigned to the same group and dissimilar objects to different groups [65].

As in classification (or supervised learning), we must understand how multidimensional measurements contribute to defining the classes. However, in clustering (or unsupervised learning) the class label of each object is unknown and so we must come up with the class definition itself from the measurements alone. This is considerably more difficult. The difficulty is compounded when the clusters themselves can have arbitrary shapes in high dimensions. Sometimes only the similarity (or dissimilarity) between the objects is known and even then many choices might be suitable.

Despite these difficulties, clustering methods are a good means for discovering patterns in data with little prior knowledge. They have been used in various academic and industrial applications. A brief literature review of clustering and typical clustering methods is given in Appendix A.

Both the broad range of applications and the difficulties make clustering a hot research area and many different approaches have been proposed. As reviewed in Appendix A, different clustering approaches have different motivations. For example, the motivation of k-means [48] is to discover centre-based clusters; that of DBSCAN

[17] is to assign sample points to the same cluster if they come from the same high-density region.

Different clustering approaches also have different assumptions. For example, k-means assumes data from multiple disjoint hyper-spheres. Both mixture-model-based clustering [21] and some density-based clustering methods such as runt pruning [62] assume that the data to be clustered come from an underlying mixture of statistical distributions. However, mixture-model-based clustering assumes that the sample points from each cluster come from a component of the mixture, whereas runt pruning assumes that the sample points come from a mode or bump of the density of the mixture.

Different motivations and assumptions lead to different methodologies for different clustering approaches. For example, mixture-model-based clustering often uses an EM algorithm to maximize the likelihood derived from the mixture of distributions; k-means minimizes the sum of squares of the distance from each sample point to its cluster mean; runt pruning applies a pruning process to cut the edges of a minimum spanning tree.

The different methodologies can lead to different results. For example, runt pruning produces a cluster tree [62]. K-means produces a partition, which is a set of disjoint subsets of the sample points to be clustered such that the union of these subsets is identical to the full set of sample points. K-means with different k, or even with the same k but different random starts, can produce different partitions.

The differences in the outcomes raise some questions. How to choose a clustering method? How to evaluate the results? How to deal with different outcomes if none of them is good enough? Given these questions, it is clear that clustering needs a framework. Theoretical studies of clustering have received more and more attention in recent years. Many of them try to build a framework that generalizes some aspect of clustering. A brief review of some typical approaches is given in Appendix B. The review indicates that few of these approaches both capture the generality of clustering and have practical value. The difficulty and variety of clustering together with the lack of a general and practical framework motivate us to propose a new clustering framework.

Statistics relies on distributions such as the Normal distribution, $N(\mu, \sigma^2)$. There are parameters for each distribution, such as μ and σ^2 for the Normal distribution. If we are interested only in μ then although it can not uniquely determine a Normal distribution, it is a parameter of interest. We estimate parameters by observed samples. For example, μ can be estimated by the sample mean or median, or by a single observation in a sample. Given different estimates, we have measures to evaluate their performance. For example, we can use bias, mean squared error, or variation to compare different estimates and choose among them. The above four components—a distribution, its parameters, estimates of the parameters, and evaluation measures—form a standard statistical framework. Figure 1.1 shows this

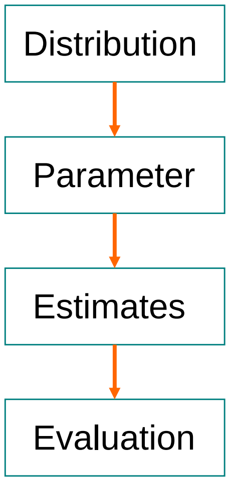


Figure 1.1: A standard statistical framework

framework. By borrowing this idea, we construct a new clustering framework, shown in Fig. 1.2.

We assume that the data to be clustered come from an underlying distribution. Clusters correspond to modes or bumps in the density for that distribution. A high-density cluster tree [31, 62] is therefore defined to capture the information on

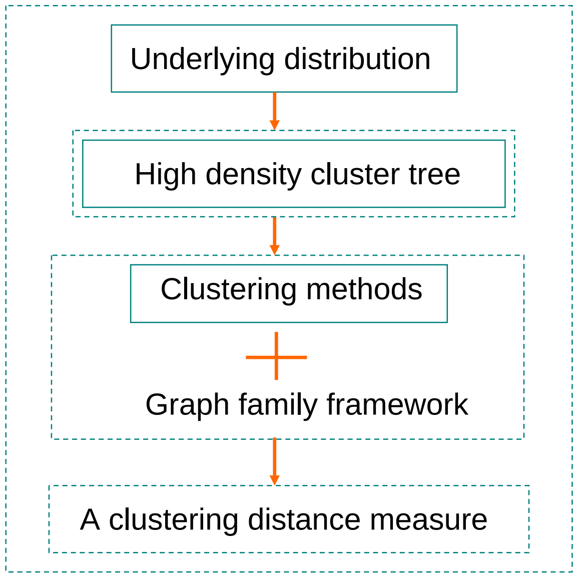


Figure 1.2: A new clustering framework

modes in the density function. It can be regarded as a parameter of the distribution corresponding to its density function. We further extend the tree to discrete cases to make it more general.

Basically, the high-density cluster tree can be estimated by a clustering method that can produce a cluster tree for a sample. For example, a dendrogram from a single linkage [31] of a sample is an estimate of the high-density cluster tree of the underlying distribution. The question of how to choose among different estimates is an interesting issue. Our approach is to use a cluster-tree distance measure to assess the performance of the estimates. The measure is defined based on a graph family framework that is derived directly from the clustering outcomes.

These four components—the underlying distribution density function, the high-density cluster tree and its extension to discrete cases, the clustering methods and the graph family framework, and the clustering distance measure—tie in together and form our new clustering framework.

Note that each dashed box in Fig. 1.2 represents a contribution made by this thesis to the clustering framework. The dashed box surrounding the whole framework represents our contribution in combining the components to form a new clustering framework. To get an overview of the framework, we describe each component briefly in the following section.

1.1 Brief description of the clustering framework

In this section we briefly describe each component of the framework shown in Fig. 1.2. Detailed discussions are given in the following chapters.

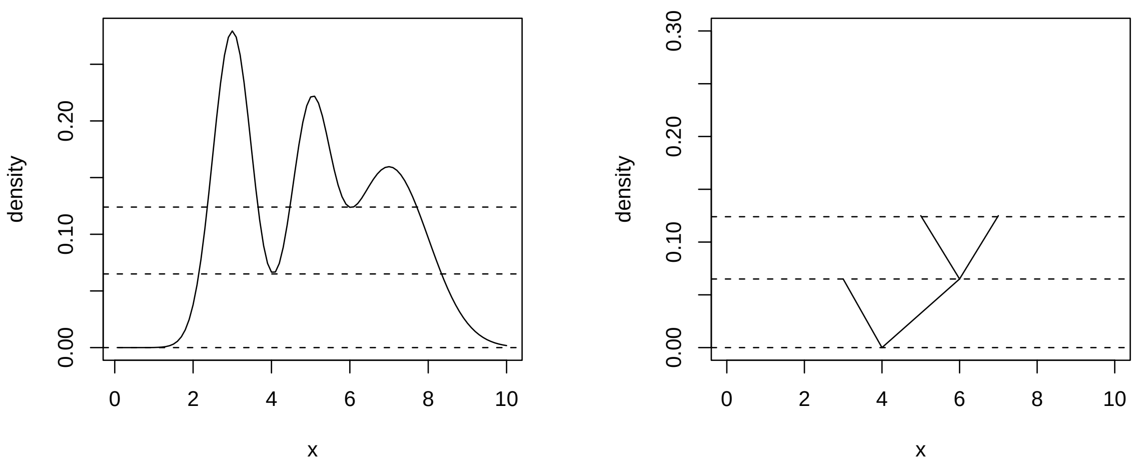
1.1.1 Underlying distribution and clustering

From a statistical point of view, observed data come from an underlying distribution. A well-defined clustering goal is to discover from a sample modes in the density of the corresponding distribution. The determining factor for clustering is not the observed sample but rather the underlying distribution and more specifically the corresponding density function. Samples contain information that enables the clustering to discover modes of the underlying density. Since density modes can be nested, a high-density cluster tree is defined to capture the information on the modes of the density function of a continuous distribution.

1.1.2 High-density cluster tree

A high-density cluster tree [31, 62] is constructed to capture the information on the modes of a density function. Its construction is based on a high-density level set at a certain level [31]. A high-density level set at $\lambda \geq 0$ is a subset of the support for the density such that the density over this subset is greater than λ and it contains all the points in the support with density greater than λ. A cluster at level λ is a maximally connected subset in the level set at λ. It is natural to define connectivity for points in a continuous distribution.

The example in Fig. 1.3 shows how a high-density cluster tree is constructed.



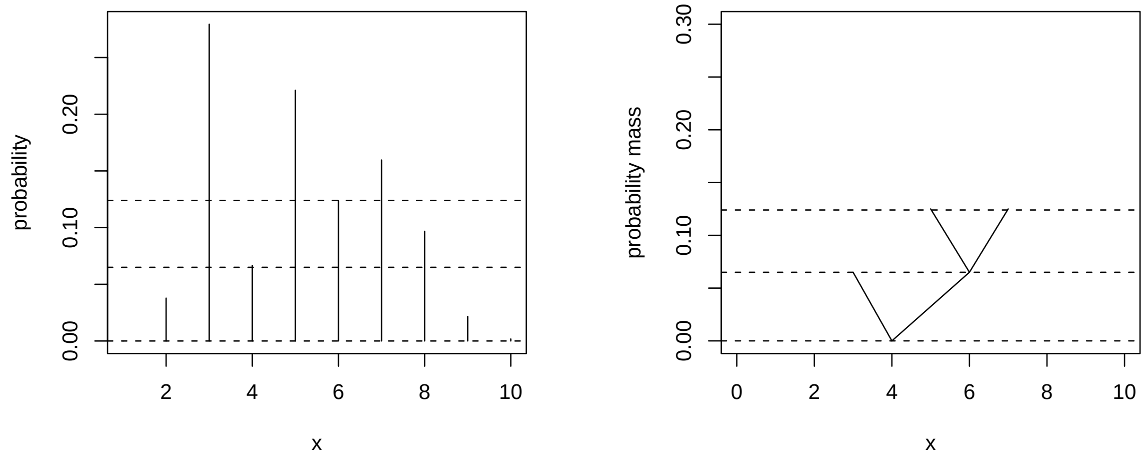
(a) P.d.f. of a continuous distribution (b) High-density cluster tree

Figure 1.3: Density level set and cluster tree

Figure 1.3(a) shows a density function. We first set $\lambda = 0$, so the high-density level set contains the whole support which is a continuous set. We therefore construct a root node corresponding to the current level set. When we increase λ slightly above the middle dashed line, the current level set splits into two disjoint continuous subsets, and we therefore add two child nodes to the root such that each node contains a subset. When we further increase the level slightly above the upper dashed line, the subset corresponding to the right child node of the root splits and therefore two child nodes are added to it. The corresponding high-density cluster tree is shown in Fig. 1.3(b).

The density function of a distribution uniquely determines a high-density cluster tree. Although the tree can not uniquely determine a density function, it captures information on the modes in a density function and therefore can be regarded as a parameter of interest for the corresponding distribution. The goal of clustering now becomes finding a good estimate of the high-density cluster tree given a sample. However, we must address the issue of how to construct such a tree for a probability mass function of a discrete distribution if its support is located in a Euclidean space.

In the discrete case, we can define a high-probability mass level set at λ to be a subset of the support for the probability mass function such that the mass over the subset is not less than λ. A cluster at λ is the maximally connected subset in the level set at λ. However, there is no natural way to determine connectivity or contiguity among discrete points in a level set. There is a straightforward way to determine contiguity for one-dimensional cases: every pair of points in a level set at λ is contiguous at λ if there is no point outside the level set located in between them. In Fig. 1.4(b) we can easily construct a high-probability mass cluster tree for



(a) Mass function of a discrete distribution (b) Cluster tree

Figure 1.4: Mass level set and cluster tree

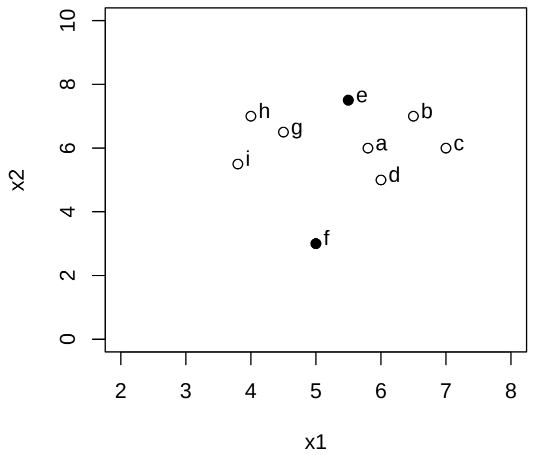
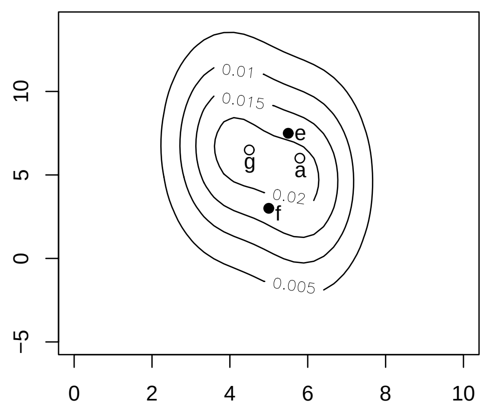


Figure 1.5: Scatter plot of a 2-d discrete distribution

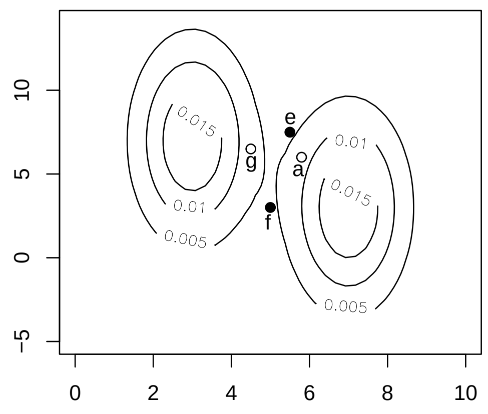
the probability mass function shown in Fig. 1.4(a).

Now we must consider higher dimensional spaces. For example, in Fig. 1.5, we have a level set in a two-dimensional space that contains every point except e and f. Clearly points a, b, c, and d are contiguous with each other and so are points g, h, and i. Are points a and g contiguous or not? If they are, we have one cluster in the level set, and if they are not, we have two clusters. The determination of contiguity for points a and g changes the clustering in this case. There is no obvious way to answer this question. A contiguous analogue may be helpful, but we can have at least two different scenarios as shown in Fig. 1.6. In a contour with a ridge, it is reasonable to claim that a and g are contiguous. In a contour with a valley, they are not.

The determination of contiguity is an open problem. To solve this problem, we first try a four-spring model. This model is natural and easy to explain. However, it is suitable only for a discrete distribution in two-dimensional space and is computationally expensive. We therefore propose a coupling graph method. The coupling



(a) P.d.f. contour with ridge



(b) P.d.f. contour with valley

Figure 1.6: Determining contiguity by p.d.f. contours

graph approach is simple and easy to implement. To construct a high-probability mass cluster tree, the coupling graph method requires only the distances between each pair of points and the probability mass of each point in the support. It is efficient for high-dimensional cases because it does not depend on the dimensions.

We further discover that in a special case, the tree constructed by Hartigan's approach is identical to the high-probability mass cluster tree constructed by the coupling graph method. Hartigan's approach works only for a distribution whose support is located on a regular lattice. His approach connects each pair of points inside a level set if they share a common gridline [32]. Our approach is more general and not constrained by the support of a discrete distribution.

Definitions and further exploration of the high-density cluster tree are given in Chapter 4. The definitions for a high-probability mass cluster tree and the coupling graph method to construct such a tree are also given in Chapter 4. The four-spring model is discussed in Appendix D.

1.1.3 Clustering methods and a graph family framework

The high-density cluster tree is a parameter of the corresponding underlying distribution. We have extended the tree for discrete cases whose support is located in a Euclidean space. A clustering method that produces a cluster tree is an estimator of the high-density/mass cluster tree. For example, a hierarchical clustering method such as single-linkage is an estimator; a dendrogram produced by single-linkage is an estimate. Even a method that produces a partition of a sample can be regarded as an estimator if we force a root node containing all the sample points to be the parent node of each node that contains a cluster in the partition. This is a two-layer tree. Such a method is not a good estimator for a high-density cluster tree with more than two layers, but it can still be regarded as an estimator.

There are many clustering methods that can be regarded as estimators. An immediate issue is how to assess their performance. Our approach is to use a cluster-tree distance measure that calculates the difference between an estimate (cluster tree) and its estimand (high-density cluster tree). The distance measure is designed based on a graph family framework, and this framework is derived directly from clustering outcomes.

The following example shows how a graph family is constructed from clusterings. A toy data set is shown in the centre of Fig. 1.7. Figures 1.7(a) to (e) show five different partitions of this data set. Points in the same colour (except grey) form a cluster in each partition, and each point in grey forms a trivial cluster. If we connect each pair of points in a cluster by an undirected edge, we construct a graph for each partition; Fig. 1.8. shows these graphs. Since they share a common vertex set, they form a graph family. Since we can not reorder them such that a successor is a subgraph of its predecessor in the family, the graph family is not monotonic.

A monotonic family of graphs can be constructed from a cluster tree. The above toy data set with an index on each point is shown in the centre of Fig. 1.9.

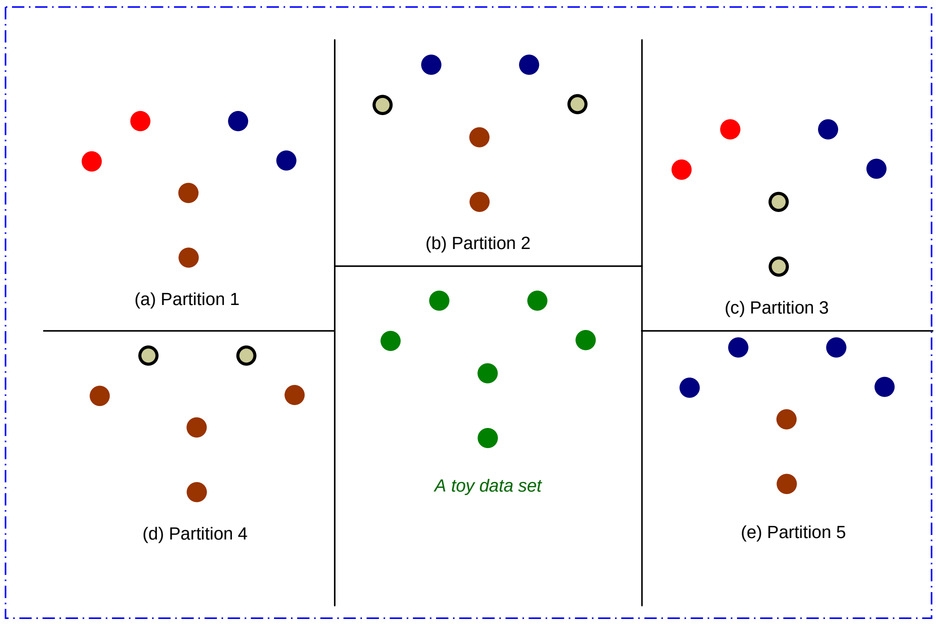


Figure 1.7: Partitions on a toy data set

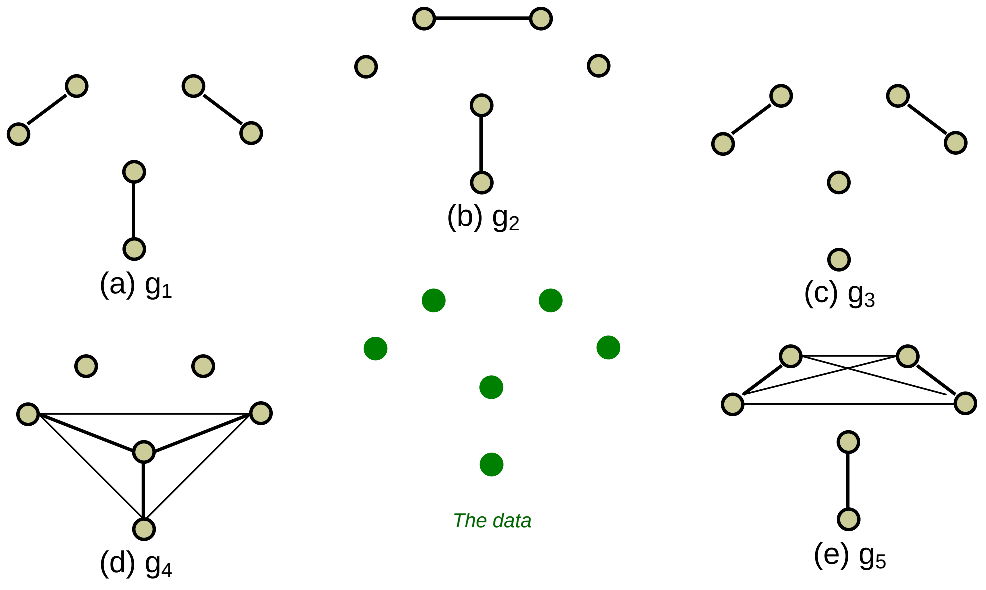


Figure 1.8: A graph family constructed from clustering partitions

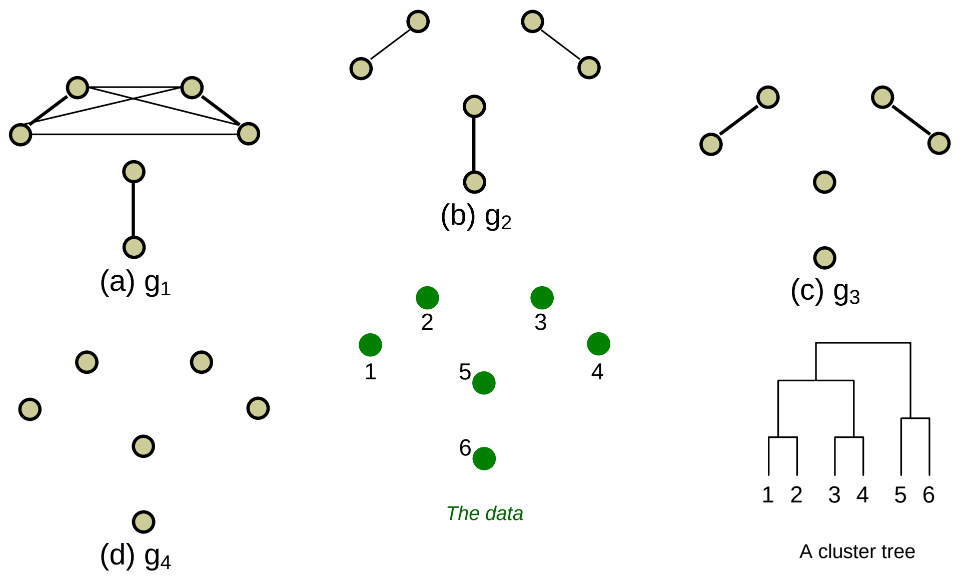


Figure 1.9: A monotonic family of graphs constructed from a cluster tree

A cluster tree constructed by single-linkage on the data set is shown at the bottom right of Fig. 1.9. Each horizontal bar in the cluster tree represents a non-leaf node. There are six leaf nodes representing six points. This is a five-layer tree. We can construct a graph corresponding to each layer starting from the root. Each graph is constructed by connecting every pair of points contained in the same node with an undirected edge. Figures 1.9(a) to (d) show the graphs corresponding to the first four layers of the tree. These graphs with the order shown in the figure form a monotonic family of graphs. Since the fifth layer of the tree contains only the leaf nodes, which have a single point, its corresponding graph contains no edge. We do not include this trivial graph in the graph family.

By further exploration we construct a graph family framework as shown in Fig. 1.10. In this framework, $\mathcal{G}$ represents a graph family that can be monotonic or

$$\begin{tikzcd}
\mathcal{G} \arrow[r, "Tot(\mathcal{G})"] & G_w \arrow[r, shift left, "\text{generated}"] & \mathcal{G}_m \arrow[l, shift left, "Tot(\mathcal{G}_m)"] \arrow[r, shift left, "\text{component tree}"] & T_{com}(\mathcal{G}_m) \arrow[l, shift left, "\text{component-generated}"]
\end{tikzcd}$$

Figure 1.10: The graph family framework

not. G_w represents a weighted graph, which is a graph with a weight on each edge. A weighted graph can be constructed from a graph family total. $\mathcal{G}_m$ represents a monotonic family of graphs. $T_{com}(\mathcal{G}_m)$ represents a tree corresponding to $\mathcal{G}_m$. Applications of clustering can be derived from this graph family framework, and a cluster-tree distance measure is among them.

Chapter 2 gives the definitions, properties, and construction of such a graph family framework. Chapter 3 shows how this graph family framework is connected to clustering.

1.1.4 Clustering distance measure

By following the arrows in the graph family framework from $T_{com}(\mathcal{G}_m)$ to G_w, we define a new cluster-tree distance measure. Figure 1.11 shows how this measure is

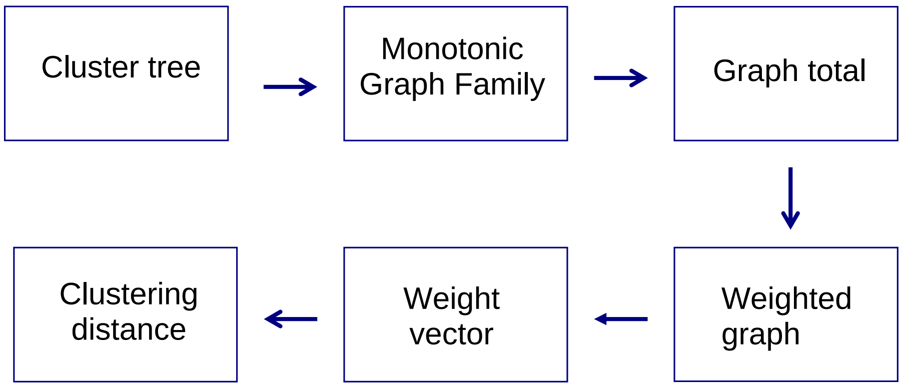


Figure 1.11: A new cluster tree distance measure

defined.

Given a cluster tree built for a sample, which corresponds to $T_{com}(\mathcal{G}_m)$ in the graph family framework, such as the example shown in Fig. 1.9, we can construct a monotonic family of graphs denoted $\mathcal{G}_m$. By calculating the total of $\mathcal{G}_m$, we obtain a weighted graph denoted G_w. A weight vector can be constructed from G_w such that each element in the vector corresponds to each pair of points in the sample. If an edge connecting a pair of points exists in G_w, the corresponding element in the vector is the weight on that edge. If no edge exists, the corresponding element in the vector is zero.

The distance between two cluster trees of a sample is the Euclidean distance between the two normalized weight vectors constructed from these two cluster trees. The measure is actually obtained by mapping the space of cluster trees constructed for a fixed finite sample into a Euclidean space. This mapping is achieved through the graph family framework. The measure was originally designed for cluster trees. Since a tree can also be constructed for a partition, we call the measure a clustering distance measure. It can be written as a kernel or an inner product, or as an angle between two weight vectors.

In Chapter 2, we introduce a new tree distance measure. In Chapter 3, we discuss its application to clustering, which is the clustering distance measure.

1.2 More applications of the graph family framework

Each component of the clustering framework has been briefly introduced. By following the arrows from $\mathcal{G}$ to $T_{com}(\mathcal{G}_m)$ in the graph family framework of Fig. 1.10, we can derive three additional applications.

1.2.1 Partition integration

We can build a single cluster tree for a sample from a set of different partitions constructed for the same sample. As shown in Fig. 1.12, given a sample, using different partitioning methods, we can obtain a set of different partitions. The set can also be obtained by just one method such as k-means if we use different values of k or fix k but use random starts. As in the example shown in Fig. 1.8, a graph family can be obtained from these partitions. From the total of the graph family, we get a weighted graph from which a monotonic family of graphs can be constructed. A single cluster tree is then constructed based on the monotonic family of graphs.

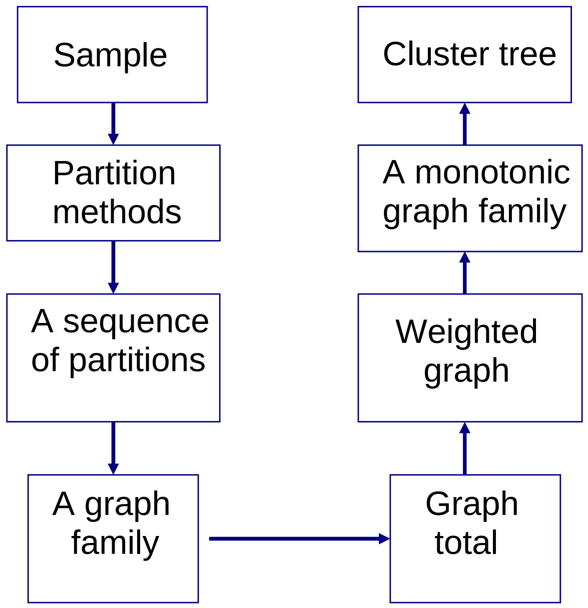


Figure 1.12: The method of clustering partition integration

Figures 1.13 and 1.14 show how a cluster tree is constructed from five partitions

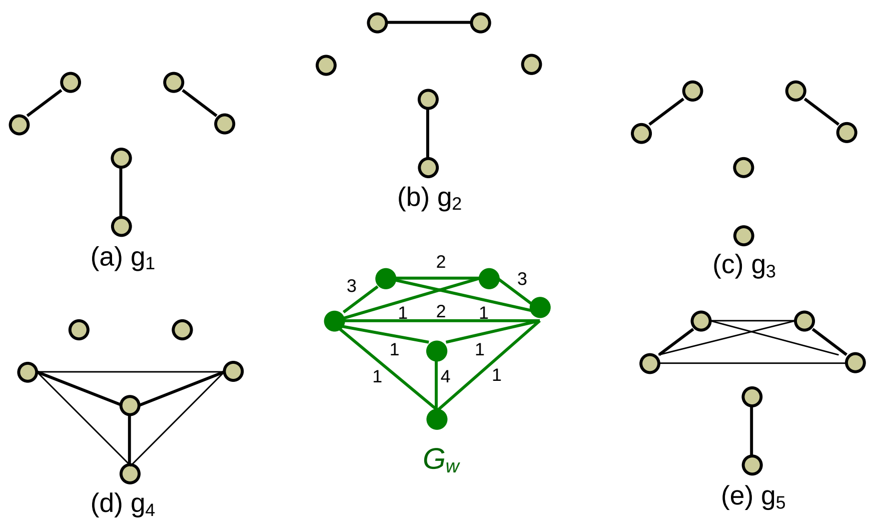


Figure 1.13: The total of a graph family

of the toy data set from Fig. 1.7. In Fig. 1.13, a weighted graph is constructed as the total of the graph family corresponding to five partitions. In Fig. 1.14, a monotonic family of graphs is generated from the total of the graph family. A cluster tree is then constructed for this monotonic family of graphs. The root of the cluster tree contains the whole data set. The left child node of the root contains points 5 and 6. The right child node of the root contains the other four points and each of its two child nodes contains two points: they are 1, 2 and 3, 4. The index of the points in the toy data set is shown in Fig. 1.9.

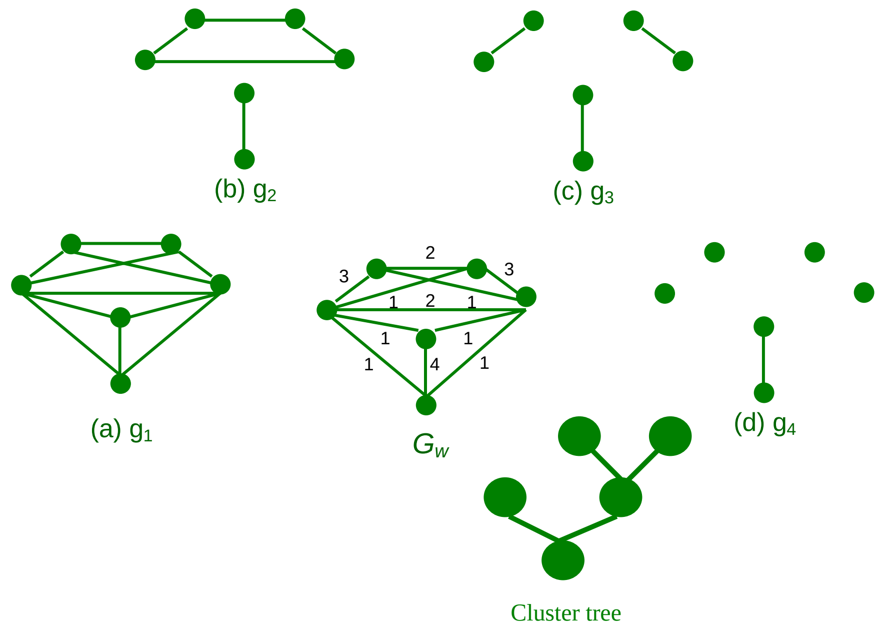


Figure 1.14: A cluster tree generated from a weighted graph

1.2.2 Cluster tree averaging

As shown in Fig. 1.15, given a sample, different clustering methods lead to a set

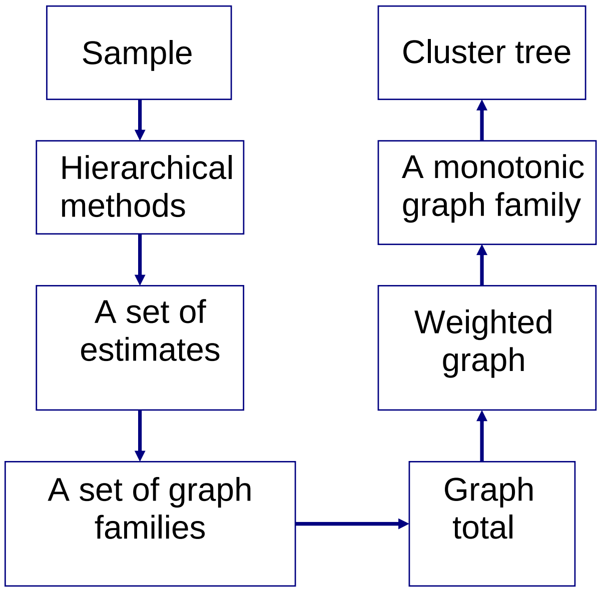


Figure 1.15: The method of cluster tree averaging

of different cluster trees. For each cluster tree, we construct a monotonic family of graphs. We obtain a larger graph family by combining these families together. The larger graph family may not be monotonic. Through total, we have a weighted graph and a single cluster tree is constructed from the monotonic family of graphs generated from the weighted graph.

1.2.3 Cluster tree bagging

This method is similar to cluster tree averaging. The difference is that bootstrapping is used for bagging. Using bootstrapping, we obtain a sequence of bootstrapped samples. For each such sample, we construct a cluster tree using a single clustering method. We therefore obtain a set of cluster trees. Using the same strategy used for the cluster tree averaging, we construct a single cluster tree. Figure 1.16 shows this

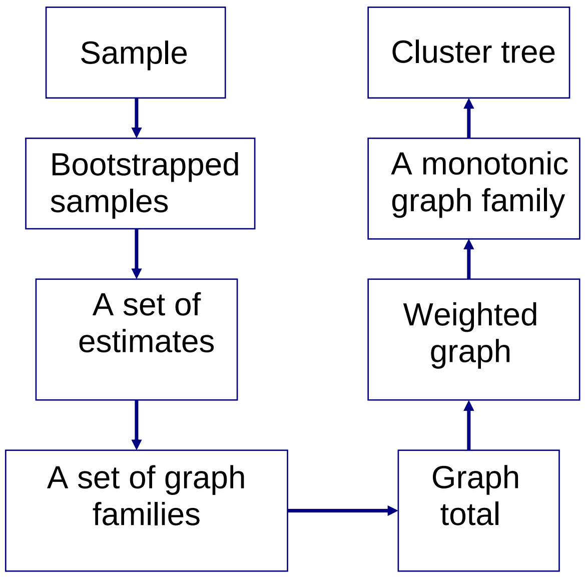


Figure 1.16: The method of cluster tree averaging

method.

In Chapter 2, we introduce methods to integrate a set of graphs into a tree and to combine different trees into a single tree. These methods are derived from the graph family framework. In Chapter 3, we consider applications of the methods discussed in this section to clustering.

1.3 Experiments and applications

To check the performance of our new clustering framework, we carried out experiments as follows:

Examination of our clustering distance measure Fowlkes and Mallows set up experiments using sampling techniques to test their clustering similarity measure [20]. We carry out their experiments using our clustering distance measure. We use many more clustering methods than they did. Our outcomes agree with their results and we have results that they did not discover. This shows the ability of our measure to compare different clusterings.

Comparison of clustering performance These experiments compare cluster trees constructed using clustering methods (estimators) and the high-density cluster tree (estimand) on a sample drawn from a distribution. The experiments use the Monte Carlo method.

Empirical examination of convergence A high-probability mass cluster tree can be constructed for a discrete distribution; a high-density cluster tree can be constructed for a continuous distribution. If a sequence of discrete random variables converges in distribution to a continuous random variable, we examine the convergence of the corresponding high-probability mass cluster trees to the corresponding high-density cluster tree. This examination is achieved via our clustering distance measure and the Monte Carlo method.

Additional applications We have experiments that examine partition integration, cluster tree averaging, and bagging.

We also apply our clustering distance measure and the additional applications to two real data sets, Enron email data and olive oil sample data.

The implementation of Fowlkes and Mallows' experiments are discussed in Chapter 3 when we examine our cluster tree distance measure. Chapter 5 discusses the Monte Carlo methodologies for evaluating clustering using our distance measure.

Chapter 6 presents and analyses the outcomes of the experiments. The applications to real data sets are discussed in Chapter 7.

1.4 Contributions

The contributions of the thesis include: the extension of a high-density cluster tree to discrete distributions; the graph family framework and its application including a new clustering distance measure; the assessment of clustering performance through this measure; the idea of forming a new clustering framework based on the concept of a statistical framework; and additional techniques derived from the graph family framework for generating new clustering methods.

In this new clustering framework, under a well-defined clustering goal, the high-density cluster tree or its extension can be regarded as a parameter of interest for the underlying distribution. A clustering method, which can produce a cluster tree, can be regarded as an estimator. From a graph family framework, which is derived directly from clustering outcomes, a clustering distance measure is defined to assess the performance of different estimators. Some further techniques such as the cluster tree bagging are derived from the graph family framework. These techniques can be used to generate new estimators. The work in this thesis provides a new and general view of clustering and has practical value.

1.5 Organization of the thesis

The thesis is organized as follows. In Chapter 2, we propose a new graph family framework based on a graph view of clustering outcomes. To construct this framework, we first define some terms including graph family, monotonic family of graphs, and the total of a graph family. We propose methodologies to generate a monotonic family of graphs from a graph family that is not monotonic and to generate a component tree from a monotonic family of graphs. These methodologies are derived based on the total of a graph family. This chapter also addresses applications of the framework such as a tree distance measure, methods to generate a tree from a sequence of graphs, and methods to produce a single tree from different trees.

Chapter 3 shows the connection of this graph family framework to clustering. The applications of the framework can also be used for clustering, and Chapter 3 addresses these applications as well. We implement Fowlkes and Mallows' experiments to evaluate our tree distance measure for clustering.

Based on a well-defined clustering goal, Chapter 4 parameterizes clustering using Hartigan's definition of a high-density level set [31] and Stuetzle's definition of a high-density cluster tree [62] for continuous distributions. We discover some properties of this parameter. Also in Chapter 4, we define the high-probability mass cluster tree and propose new methods to determine contiguity for a discrete distribution with a Euclidean support space. The algorithms developed to construct a high-probability mass cluster tree are also given in this chapter.

Chapter 5 sets up methodologies for comparing clustering performance and examining convergence properties. These methodologies are the basis for the experiments discussed in Chapter 6. The clustering distance measure and the Monte Carlo method are used in these experiments. The outcomes of these experiments are presented and analysed in Chapter 6. Chapter 7 presents the application of our methods and the clustering distance measure to two real data sets: the Enron email data set and the olive oil sample set. Chapter 8 provides concluding remarks and suggests possible future research directions.

Appendix A provides a brief review of clustering and typical clustering methods. Appendix B givess a brief review of some existing approaches to a framework for clustering. Appendix C presents some standard definitions from graph theory. These definitions are necessary when we construct a probability mass cluster tree. Appendix D gives a four-spring model. We propose this model to determine the contiguity of points in a discrete distribution. The model is natural but has limits. We therefore use a more general approach, the coupling graph method. Appendix E lists the clustering methods that we used for the experiments in this thesis.

Chapter 2

A graph family framework

2.1 Chapter summary

As explained in Chapter 1, a graph family is associated with a sequence of partitions of a sample; a monotonic family of graphs is associated with a cluster tree; the total of a graph family is defined as a weighted graph; and a monotonic family of graphs can be generated from a weighted graph. A graph family framework is constructed based on the above building blocks.

The idea underlying the graph family framework is motivated by clustering and the use of this framework in clustering becomes an important part of our clustering framework shown in Fig. 1.2 of Chapter 1. Since a graph family is defined on a vertex set, the construction of the graph family framework is independent of clustering.

In this chapter, we first define graph family, monotonic family of graphs, and graph total, and then describe in detail how a graph family framework is constructed. Applications of this framework are also given.

2.2 Monotonic family of graphs

Before defining a monotonic family of graphs, we define a graph family.

Definition 2.2.1 *A* **family of graphs** *is an indexed collection of graphs* $\mathcal{F} = \{g_k\}$

such that $\forall i < j$, $V_j \subseteq V_i$, *where* V_i *and* V_j *are the vertex sets of* g_i *and* g_j *respectively.*

Definition 2.2.2 $\mathcal{F} = \{g_k\}$ *is a* **monotonic family of graphs** *if* $\mathcal{F}$ *is a family of graphs and* $\forall i < j$, $E_j \subseteq E_i$, *where* E_i *and* E_j *are the edge sets of* g_i *and* g_j *respectively. We further call* g_i *the* **immediate predecessor** *of* g_j *in* $\mathcal{F}$ *(or* g_j *the* **immediate successor** *of* g_i *in* $\mathcal{F}$*) whenever* $i < j$ *and there exists no* $g_k \in \mathcal{F}$ *such that* $i < k < j$*. If each graph in* $\mathcal{F}$ *has the same vertex set,* $\mathcal{F}$ *is called a* **monotonic partition family of graphs***. If there do not exist any pair of graphs in* $\mathcal{F}$ *that have the same vertices and the same edge sets,* $\mathcal{F}$ *is called a* **reduced monotonic family of graphs***.*

Note that we say g_j is a subgraph of g_i, denoted $g_j \subseteq g_i$, iff $V_j \subseteq V_i$ and $E_j \subseteq E_i$. In a family of monotonic graphs, $\mathcal{F} = \{g_k\}$, $g_j \subseteq g_i$ iff $i < j$. Figure 2.1 shows an

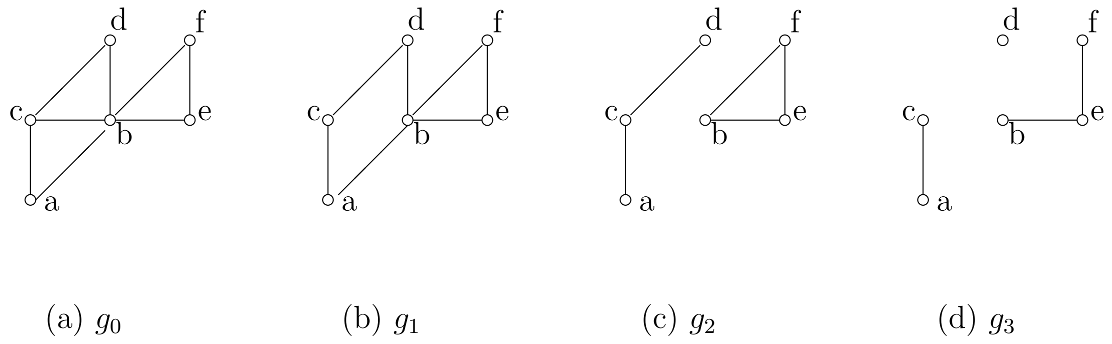


Figure 2.1: Example of a monotonic partition family of graphs.

example of a monotonic partition family of graphs with index starting from 0.

For simplicity and convenience, we do not allow a mixture of weighted and unweighted graphs in a graph family. Taking this into account, definitions 2.2.1 and 2.2.2 can be extended so that if all members of a family of graphs have weights assigned to their edges, the family will be called a family of weighted graphs. We can similarly define a family of unweighted graphs, a monotonic family of weighted graphs, and a monotonic family of unweighted graphs, etc.

For weighted graphs, graph operators can be defined as follows.

Definition 2.2.3 *The addition of two weighted graphs* $g_1 =< V_1, E_1, W_1 >$ *and* $g_2 =< V_2, E_2, W_2 >$ *is denoted by* $g_1 + g_2 =< V, E, W >$*, where* $V = V_1 \cup V_2$*,* $E = E_1 \cup E_2$*,* $W = \{w_j = w_{j,1} + w_{j,2}\}$*,* $w_{j,1}$ *is the weight on edge* e_j *from* g_1 *with* $w_{j,1} = 0$ *if* e_j *does not exist in* E_1*, and* $w_{j,2}$ *is defined similarly.*

Definition 2.2.4 *The total of a family* $\mathcal{G}$ *of weighted graphs, denoted* $Tot(\mathcal{G})$*, is the graph*

$$\sum_{\forall g \in \mathcal{G}} g.$$

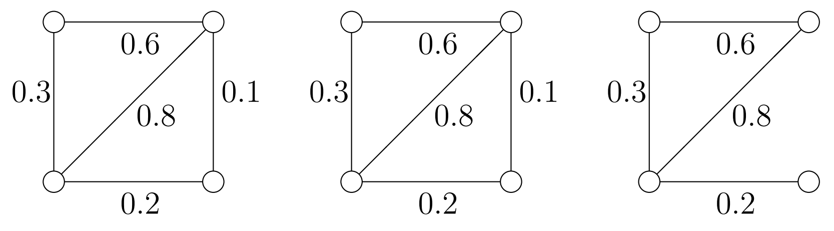
An unweighted graph $g_u =< V, E >$ can be transformed to a weighted graph $g_w =< V, E, W >$ by simply assigning positive weights to the edges in E. The standard transformation is to assign weight 1 to each edge in E. We can transform a family of unweighted graphs to be a family of unity-weighted graphs by this standard transformation. The total of a family of unweighted graphs can be calculated using the family of transformed unity-weighted graphs.

Consider how we might generate a monotonic family of weighted graphs from a single weighted graph. A natural way might be to use the edge weights to determine whether or not an edge appears. To be more specific, let $G =< V, E, W >$ be a weighted graph, where V is the set of vertices, E is the set of all the edges in G, and $W = \{w_j = w(e_j) : w(e_j)$ is the weight on edge $e_j \in E$, $w(e_j) > 0\}$. We can always construct $g_p =< V, E_p, W_p >$ where $e_j \in E_p$ iff $e_j \in E$ and $w_j \geq p$. We have $g_p \subseteq G$ and $g_p \subseteq g_q$ iff $p > q$. Therefore we can construct a monotonic family of weighted graphs, $\mathcal{G} = \{g_p : p \in \mathbb{R}^+\}$. However generating a graph family this way may have some problems which we explain as follows.

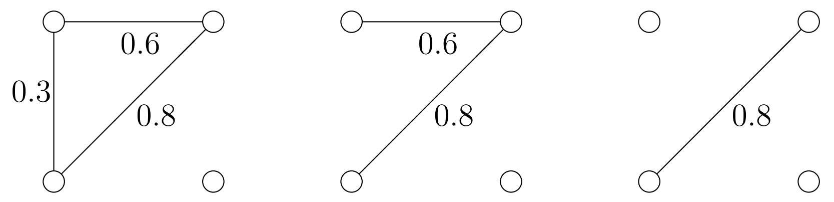
The family of weighted graphs, $\mathcal{G}$, generated above has uncountably infinite members with the weights on edges of $Tot(\mathcal{G})$ a vector of values that could all be infinite. (Note that, if we set $p \in \mathbb{Q}^+$ in the above construction, there are countably infinite members in the generated family.) Since there are uncountably (or countably) infinite repeats of graphs in $\mathcal{G}$, we further reduce $\mathcal{G}$ in the following way.

Let $W_l = \{l_1, l_2, \cdots, l_m\}$ be the set of all the unique levels of weights in W from the weighted graph G. The reduced version of $\mathcal{G}$, denoted $\mathcal{G}_R$, $\mathcal{G}_R = \{g_p : p \in W_l\}$

is a reduced monotonic family of weighted graphs. The number of members in $\mathcal{G}_R$ equals the number of unique levels of weights from W in G. We denote $W_{(l)} = \{l_{(1)}, l_{(2)}, \cdots, l_{(m)}\}$ to be the ordered set of W_l, then the set of weights on all the edges of $Tot(\mathcal{G}_R)$ equals $\{l_{(1)}, 2l_{(2)}, 3l_{(3)}, \cdots, ml_{(m)}\}$. Figure 2.2 shows an example of



(a) graph G (b) g_1 in $\mathcal{G}_R$ (c) g_2 in $\mathcal{G}_R$



(d) g_3 in $\mathcal{G}_R$ (e) g_4 in $\mathcal{G}_R$ (f) g_5 in $\mathcal{G}_R$

Figure 2.2: An example of a naturally generated family of graphs.

a reduced family of weighted graphs generated from a weighted graph G using this method.

We are interested in the condition $Tot(\mathcal{G}) = G$. The graph family $\mathcal{G}$ described above does not preserve this feature, nor does its reduced version $\mathcal{G}_R$ unless only one member exists in $\mathcal{G}_R$. The following definition ensures that the condition $Tot(\mathcal{G}) = G$ is preserved.

Definition 2.2.5 *Given a weighted graph $G =< V, E, W >$, let $W_l = \{l_1, l_2, \cdots, l_m\}$ be the set of all the unique levels of weights in W, and let $W_{(l)} = \{l_{(1)}, l_{(2)}, \cdots, l_{(m)}\}$ be* *the ordered set of W_l. The* **weight-generated family of graphs***, denoted $\mathcal{F}_w(G)$, is constructed by a sequence of graphs $g_i =< V, E_i, W_i >$, $i = 1, 2, \cdots, m$, such that $E_i = \{e_j : w_j = w(e_j) \geq l_{(i)}\}$ and W_i is a vector of dimension $|E_i|$ all of whose elements equal $l_{(i)} - l_{(i-1)}$.*

From the above definition, $\mathcal{F}_w(G)$ is a uniquely generated partition family of graphs. A simple algorithm can be derived to obtain this graph family:

1. Obtain the ordered set $W_{(l)} = \{l_{(1)}, l_{(2)}, \cdots, l_{(m)}\}$ from W;
2. For each i from 1 to m, generate $g_i =< V, E_i, W_i >$ in $\mathcal{F}_w(G)$, where $E_i = \{e_j : e_j \in E, w_j \geq l_{(i)}\}$ and $W_i = (l_{(i)} - l_{(i-1)}) \cdot \mathbf{1}$ where $\mathbf{1}$ is a vector of 1s with length $|E_i|$.

Figure 2.3 shows an example of a weight-generated family of graphs constructed from the weighted graph G shown in Fig. 2.2.

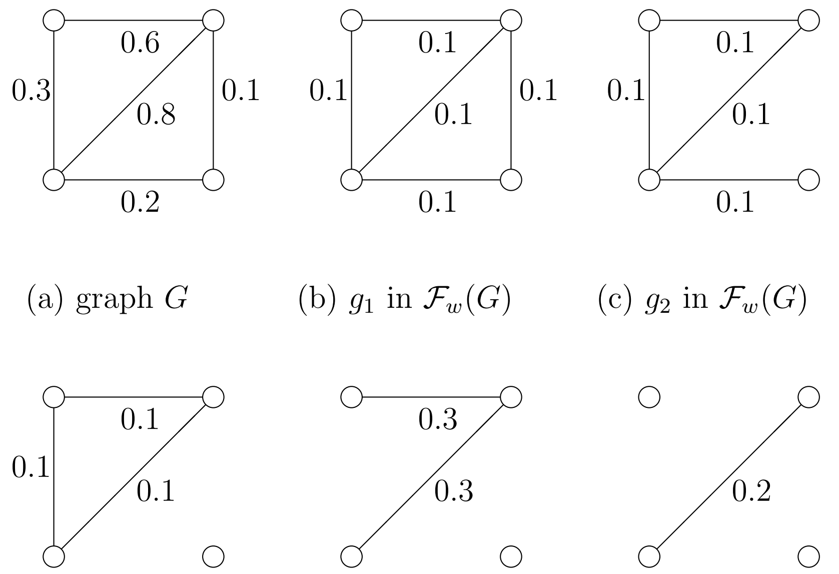
The following proposition demonstrates the properties of the weight-generated family of graphs.

Proposition 2.2.1 *Let $\mathcal{F}_w(G)$ be a weight-generated family of graphs. Then $\mathcal{F}_w(G)$ is a reduced monotonic family of weighted graphs such that $Tot(\mathcal{F}_w(G)) = G$.*

Proof: Let $\mathcal{F}_w(G) = \{g_i =< V, E_i, W_i >\}$. We have $l_{(i)} < l_{(j)}$ if $1 \leq i < j \leq m$, therefore $g_j \subset g_i$ and $g_j \neq g_i$, so by definition, $\mathcal{F}_w(G)$ is a reduced monotonic family of weighted graphs.

Let a_i be an element in W_i, then by definition we have $a_1 = l_{(1)}$, therefore when $j = 1$, we have $\sum_{t=1}^{j} a_t = l_{(j)}$. Suppose that when $j = k$, where $1 \leq j < m$, we have $\sum_{t=1}^{j} a_t = l_{(j)}$. Then for $j = k+1$, $\sum_{t=1}^{j} a_t = \sum_{t=1}^{k} a_t + a_{k+1} = l_{(k)} + (l_{(k+1)} - l_{(k)}) = l_{(k+1)} = l_{(j)}$. Thus, by mathematical induction, if $1 \leq j \leq m$, we have $\sum_{t=1}^{j} a_t = l_{(j)}$.

For all $e_j \in E$ with weight w_j, we have $w_j \in W_{(l)}$, say $w_j = l_{(k)}$. By definition, e_j appears in every graph g_i if $i \leq k$ and does not appear in any graph g_i if $i > k$. The weight of e_j in the graph $Tot(\mathcal{F}_w(G))$ equals $\sum_{t=1}^{k} a_t$ which is $l_{(k)}$. Therefore, $Tot(\mathcal{F}_w(G)) = G$. $\square$



(a) graph G (b) g_1 in $\mathcal{F}_w(G)$ (c) g_2 in $\mathcal{F}_w(G)$

(d) g_3 in $\mathcal{F}_w(G)$ (e) g_4 in $\mathcal{F}_w(G)$ (f) g_5 in $\mathcal{F}_w(G)$

Figure 2.3: An example of a weight-generated family of graphs.

We denote by $\mathcal{S}$ the set of all the reduced monotonic partition families of weighted graphs, and by $\mathcal{S}_e$ the subset of $\mathcal{S}$ such that $\forall \mathcal{F} \in \mathcal{S}_e$, every member of $\mathcal{F}$ is a graph with an identical positive weight on each edge. Note that the weights on two edges from two members in $\mathcal{S}_e$ can be different. We can produce a different graph family $\mathcal{G} \in \mathcal{S}$ by modifying the graph family generated in Fig. 2.3 so that $Tot(\mathcal{G}) = G$. There are many ways to do so: for example, reassign the weight of the only edge in g_5 to be 0.3 and change the weight of this edge in g_4 to be 0.2. This example shows that $\mathcal{F}_w(G)$ such that $Tot(\mathcal{F}_w(G)) = G$ is not unique in $\mathcal{S}$. However, Theorem 2.2.1 shows that $\mathcal{F}_w(G)$ is unique in $\mathcal{S}_e$.

Theorem 2.2.1 *$\mathcal{F}_w(G)$ such that $Tot(\mathcal{F}_w(G)) = G$ is unique in $\mathcal{S}_e$.*

Proof: Let $\mathcal{G} = \{g_i\} \in \mathcal{S}_e$ such that $Tot(\mathcal{G}) = G$. Because $Tot(\mathcal{G}) = G$, any graph in $\mathcal{G}$ can contain only the edges appearing in G. $\forall e_j \in E$, suppose the weight on e_j in G equals $l_{(k)}$ with $1 \leq k \leq m$. Because the weight of e_j is the k^{th} level in $W_{(l)}$, e_j must appear in each graph g_i with $i \leq k$. Therefore, the number of members of $\mathcal{G}$ equals m which is the same as that in $\mathcal{F}_w(G)$, and the edge set of g_i in $\mathcal{G}$ is the same as the edge set of the i^{th} graph in $\mathcal{F}_w(G)$.

Since $\mathcal{G}$ is a reduced graph family, at least one edge is removed in g_i compared with g_{i-1}, where $1 < i \leq m$. Because every member of $\mathcal{G}$ is a graph with an identical positive weight on each edge, the edges that appear in g_{i-1} but not in g_i have the same weight in $Tot(\mathcal{G})$ which is the $(i-1)^{th}$ level in $W_{(l)}$.

Denote a_i to be the identical weight of each edge in $g_i \in \mathcal{G}$. Let e_1 be an edge appearing only in g_1; its weight in G must be $l_{(1)}$, so to have $Tot(\mathcal{G}) = G$, a_1 must be $l_{(1)}$. Suppose a_i equals $l_{(i)} - l_{(i-1)}$, where $1 \leq i < m$, $l_{(i)} = 0$ if $i = 0$, and let e_{i+1} be an edge appearing in g_{i+1} but not in g_{i+2} (if $i+1 < m$). Then the weight on e_{i+1} in G must be $l_{(i+1)}$, and therefore a_{i+1} must be $l_{(i+1)} - \sum_{t=1}^{i} a_t = l_{(i+1)} - l_{(i)}$ (by the result in the proof of the above proposition).

Because $Tot(\mathcal{G}) = G$ and either $\mathcal{G}$ or $\mathcal{F}_w(G)$ is a partition family of graphs, any graph from $\mathcal{G}$ has the same vertex set as any graph from $\mathcal{F}_w(G)$.

From the above discussion, we have $\mathcal{G} = \mathcal{F}_w(G)$. $\square$

Theorem 2.2.1 is important because it implies that for any weighted graph G there exists and only exists one graph family in $\mathcal{S}_e$ such that its total is G, and $\mathcal{F}_w(G)$ is that one.

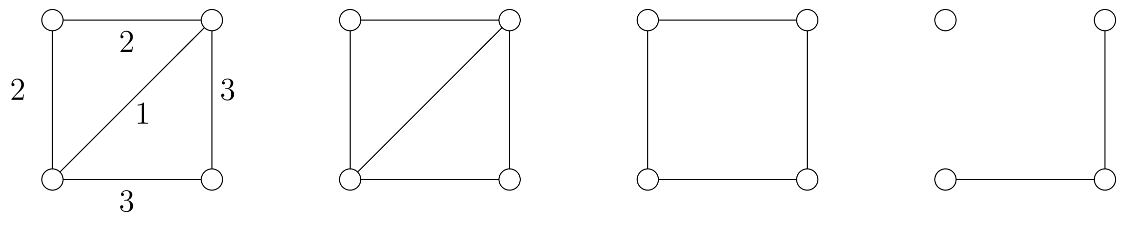
If the weight of each edge in the given weighted graph G is an element of a set of positive integers, and the maximum common factor of all the weights is 1, we can generate a family of unity-weighted graphs.

Definition 2.2.6 *Given a weighted graph $G =< V, E, W >$ with W a vector of positive integers, the* **unity-weight-generated family of graphs***, denoted $\mathcal{F}_u(G)$, is constructed by a sequence of graphs $g_i =< V, E_i, W_i >$, with $i = 1, 2, \cdots, m$, where m is the maximum weight in G, $E_i = \{e_j : w_j = w(e_j) \geq i\}$, and W_i is a vector of dimension $|E_i|$ all of whose elements equal 1.*

$\mathcal{F}_u(G)$ is a partition family of graphs. A simple algorithm can be derived to obtain this graph family:

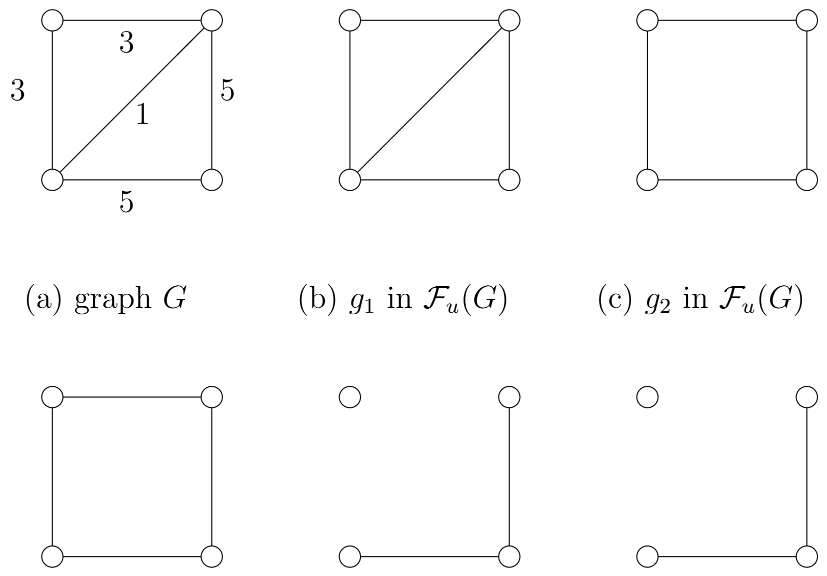
1. Set $m = max_{\forall e_j \in E}(w_j)$;
2. For each i from 1 to m, generate $g_i =< V, E_i, W_i >$ in $\mathcal{F}_u(G)$, where $E_i = \{e_j : w_j = w(e_j) \geq i\}$ and W_i is a vector of 1s with length $|E_i|$.

Figures 2.4 and 2.5 show two examples of unity-weight-generated families of graphs;



(a) graph G (b) g_1 in $\mathcal{F}_u(G)$ (c) g_2 in $\mathcal{F}_u(G)$ (d) g_3 in $\mathcal{F}_u(G)$

Figure 2.4: A reduced unity-weight-generated family of graphs.



(a) graph G (b) g_1 in $\mathcal{F}_u(G)$ (c) g_2 in $\mathcal{F}_u(G)$

(d) g_3 in $\mathcal{F}_u(G)$ (e) g_4 in $\mathcal{F}_u(G)$ (f) g_5 in $\mathcal{F}_u(G)$

Figure 2.5: A non-reduced unity-weight-generated family of graphs.

the graph family in Fig. 2.4 is reduced, but that in Fig. 2.5 is not.

From the definition, $\mathcal{F}_u(G)$ is a monotonic family of graphs, and it is reduced if the weights on edges of G form a vector of consecutive integers starting from 1. We denote $\mathcal{S}_u$ to be the set of all the monotonic partition families of unity-weighted graphs. Theorem 2.2.2 shows that $\mathcal{F}_u(G)$ such that $Tot(\mathcal{F}_u(G)) = G$ is unique in $\mathcal{S}_u$.

Theorem 2.2.2 *$\mathcal{F}_u(G)$ such that $Tot(\mathcal{F}_u(G)) = G$ is unique in $\mathcal{S}_u$.*

Proof: Since each edge e_j in G with weight say n_j by definition appears in the first n_j graphs of $\mathcal{F}_u(G)$ with weight 1, and only in those graphs, it follows that the weight on e_j in $Tot(\mathcal{F}_u(G))$ equals n_j and then $Tot(\mathcal{F}_u(G)) = G$.

Suppose there exists a graph family $\mathcal{G} = \{g_i\} \in \mathcal{S}_u$ such that $Tot(\mathcal{G}) = G$ and $\mathcal{G} \neq \mathcal{F}_u(G)$. Since $\mathcal{G}$ is a partition graph family with $Tot(\mathcal{G}) = G$, the vertex set of each graph in $\mathcal{G}$ is V, and no edge outside E can appear in any graph in $\mathcal{G}$. Let g_k be the first graph in $\mathcal{G}$ such that g_k is different from the k^{th} graph in $\mathcal{F}_u(G)$. Then there exists at least one edge e_j that appears in either the k^{th} graph of $\mathcal{G}$ or $\mathcal{F}_u(G)$ but not both. Suppose e_j appears in g_k of $\mathcal{G}$, then e_j does not appear in any graph of $\mathcal{F}_u(G)$ with index no less than k, therefore e_j has more weight in $Tot(\mathcal{G})$ than in $Tot(\mathcal{F}_u(G))$, but this is not possible. A similar conflict can be found if e_j appears in the k^{th} graph of $\mathcal{F}_u(G)$ but not in g_k of $\mathcal{G}$.

By the above discussion, $\mathcal{G}$ and $\mathcal{F}_u(G)$ must be the same, so the unity-weight-generated family of graphs is unique in the space $\mathcal{S}_u$. $\square$

Similar to Theorem 2.2.1, Theorem 2.2.2 is important because of the uniqueness it proves.

Considering the given weighted graph $G =< V, E, W >$, suppose W is a vector of real values and if there exists at least one positive real value such that a vector of positive integers remains when that real value is factored out from each element in W. Let w be the maximum such positive real value. We can construct a unity-weight-generated family of graphs by first factoring w out from the weight of each edge in G, i.e. $W = w \cdot W_I$ and $G_I =< V, E, W_I >$, then constructing the unity-weight-generated family $\mathcal{F}(G_I)_u$ and finally replacing the weight of each edge in each member of $\mathcal{F}(G_I)_u$ by w.

In this section, we have discussed generating a monotonic family of graphs from a weighted graph; on the other hand, a weighted graph can be generated from a graph family by obtaining the graph total. The following section considers how to construct a tree structure from a monotonic family of graphs and vice versa.

2.3 The component tree

Every monotonic family of graphs produces a hierarchy of graph components that in turn can be connected in what we call a component tree. To formalize this idea, the following proposition is useful.

Proposition 2.3.1 *Given two elements, g_i and g_j with $i < j$, of a monotonic family of graphs, for any connected component $c(g_j)$, there exists a unique component $c^*(g_i)$ such that $c(g_j) \subseteq c^*(g_i)$.*

Proof: By definition, $g_j \subseteq g_i$, so $c(g_j) \subseteq g_j \subseteq g_i$ and $c(g_j)$ is a subgraph of g_i. Because $c(g_j)$ is a connected component of g_j, it is a connected subgraph of g_i. There must exist a connected component $c^*(g_i)$ such that $c(g_j) \subseteq c^*(g_i)$. For any other connected component $c^{'}(g_i)$, $c^*(g_i) \cap c^{'}(g_i) = \phi$, and therefore $c^*(g_i)$ is unique. $\square$

Proposition 2.3.1 suggests a basis for organizing the connected components of a monotonic family of graphs into a tree where each node corresponds to a connected component of a graph in $\mathcal{F}$. If g_i is the immediate predecessor of g_j in $\mathcal{F}$, then an arc is drawn from every connected component $c(g_i)$ to every connected component $c(g_j)$ that is also a subgraph of the connected component $c(g_i)$.

In general, this construction might more appropriately be described as a forest of trees, in that there might be multiple root nodes, each corresponding to a connected component in the first graph of the monotonic family. To avoid this situation, we assume that the first graph has a single connected component; if this is not the case, then for the purpose of tree construction, we simply push the undirected complete graph on all vertices into the first position in the family. Similarly, for simplicity and without loss of generality, we can take the first index of the monotonic family to be 0 so that the first graph can be referred to as g_0. Such a tree can now be defined more formally as follows:

Definition 2.3.1 *Given a monotonic family of graphs $\mathcal{F}$, with first element g_0 having a single component, the* **component tree** *of $\mathcal{F}$ is the tree with nodes $N_{i,u} = c_u(g_i)$ corresponding to the u^{th} component of the graph g_i in $\mathcal{F}$, where $N_{j,v}$ is a child of $N_{i,u}$ if, and only if, g_i is the immediate predecessor of g_j in $\mathcal{F}$ and $c_v(g_j) \subseteq c_u(g_i)$.*

Definition 2.3.2 *If $\mathcal{F}$ is a monotonic partition family of graphs, the component tree of $\mathcal{F}$ is called a* **partition tree**.

Figure 2.6 shows the component tree constructed for the monotonic graph family

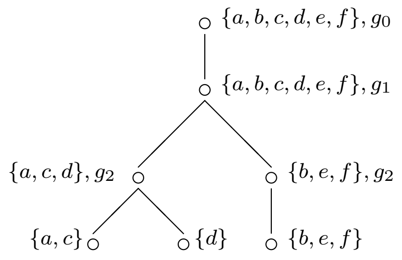


Figure 2.6: The component tree for the graph family in Fig. 2.1.

in Fig. 2.1. For simplicity every non-leaf node in the tree is represented by the vertex set of the corresponding component and the graph from which this component comes. In this example, for the leaf nodes from g_3 we show only their corresponding vertex sets. Since the graph family shown in Fig. 2.1 is a partition family of graphs, the tree in Fig. 2.6 is also a partition tree.

Some components contain a single vertex that may have little interest. These are defined formally as follows.

Definition 2.3.3 *A connected component that consists entirely of a single vertex will be called a* **trivial component**. *Similarly, connected components having more than one vertex will be said to be* **non-trivial**.

Non-trivial component trees and non-trivial partition trees are component trees and partition trees whose trivial components have been removed. Figure 2.7 shows

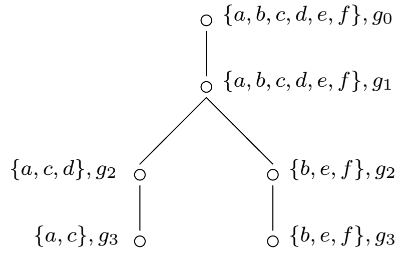


Figure 2.7: Non-trivial component tree for the graph family in Fig. 2.1.

the non-trivial component tree of the monotonic family of graphs shown in Fig. 2.1.

The tree structure defined so far allows nodes to have a single child. Sometimes, these single child nodes are of no interest and can be removed to produce a reduced tree as follows. We construct a **reduced component tree** where each node represents a component of a graph in the monotonic family of graphs, $\mathcal{F}$. Associated with each node, N, is a connected component $c_{\lambda,N}$ and a level $\lambda(N)$ being the index level where the component $c_{\lambda,N}$ of $g_{\lambda(N)}$ first appears. Descendant branches appear for the component at the node, N, at the smallest level λ^* large enough that the corresponding graph g_{λ^*} in the family has two or more connected components that are subgraphs of $c_{\lambda,N}$.

This structure is probably more easily understood with the following recursive description, as in Stuetzle's paper [62]. The root node represents the entire graph g_0 and it is associated with the index, $\lambda(N) = 0$, of the family $\mathcal{F}$. To determine the descendants of a node N having index level λ and component $c_{\lambda,N}$, find the next lowest index level $\lambda^* > \lambda$ for which g_{λ^*} has two or more connected components that are subgraphs of $c_{\lambda,N}$. Each of these k (typically $k = 2$ but it could be larger) connected components now forms a new branch in the tree from N to a descendant node D_i, for $i = 1, 2, \ldots, k$. Associated with the i^{th} descendant node D_i is the corresponding connected component c_{λ^*,D_i} and level λ^*. The procedure is applied recursively to each descendant node. If $\mathcal{F}$ is a monotonic partition family of graphs, the reduced component tree of $\mathcal{F}$ is also called a reduced partition tree. Note that two different monotonic families of graphs may yield the same reduced component tree.

Figure 2.8 shows the reduced component tree of the monotonic family of graphs

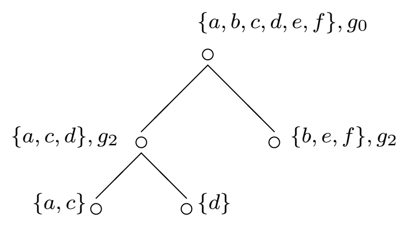


Figure 2.8: The reduced component tree for the graph family in Fig. 2.1.

shown in Fig. 2.1.

Non-trivial reduced component trees and non-trivial reduced partition trees are reduced component trees and reduced partition trees whose trivial components have been removed. Figure 2.9 shows the non-trivial reduced component tree of the mono-

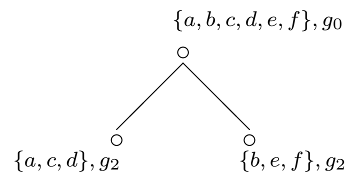


Figure 2.9: The non-trivial reduced component tree for the graph family in Fig. 2.1.

tonic family of graphs in Fig. 2.1. Note that component trees are unique by construction. However, the uniqueness of their reduced counterparts may not be so obvious.

Proposition 2.3.2 *The reduced component tree of a monotonic family of graphs is unique.*

Proof: Suppose T_1 and T_2 are two reduced component trees of the monotonic family of graphs $\mathcal{F} = \{g_k\}$. Let $N^{(1)}$ be the node in T_1 that corresponds to the component $c(g_i)$, and suppose that g_j is the first successor of g_i in $\mathcal{F}$ having more than one component, say $c_1(g_j), \cdots, c_p(g_j)$, $p \geq 2$, that are also subgraphs of $c(g_i)$. Suppose there exists a node $N^{(2)}$ in T_2 that also corresponds to the component $c(g_i)$; its child nodes will also correspond to $c_1(g_j), \cdots, c_p(g_j)$ by construction.

Now the root nodes of T_1 and T_2 are identical, corresponding to the graph g_0 in $\mathcal{F}$. Therefore, T_1 and T_2 are identical by induction. $\square$

In the construction of the reduced component tree of a monotonic family of graphs, each node of the tree is associated with a component in the family. The associated component plays an important role in the tree construction, and will be called a branch component. This is defined formally as follows.

Definition 2.3.4 *Let $\mathcal{F} = \{g_k\}$ be a monotonic family of graphs. Let g_i be the immediate predecessor of g_j in $\mathcal{F}$. Components $c_t(g_j)$ and $c_l(g_j)$ will be called* **branch components** *if there exists a component $c_h(g_i)$ such that $c_t(g_j) \cup c_l(g_j) \subseteq c_h(g_i)$.*

Note that in the above definition there could be more than two branch components as subsets of $c_h(g_i)$.

For convenience, we call the component in g_0 the root component. Any branch component including the root component in a monotonic family of graphs $\mathcal{F}$ represents a unique node in the reduced component tree of $\mathcal{F}$. Any node in the reduced component tree of a monotonic family of graphs $\mathcal{F}$ corresponds to a unique branch component in $\mathcal{F}$.

The discussion so far in this section shows how to construct a tree structure from a monotonic family of graphs. We are also interested in generating a monotonic family of graphs from a tree structure.

Definition 2.3.5 *An* **inheritance path** *of a tree is a sequence of nodes such that the immediate successor of a node in this path is its direct child node in the tree. A*

layer of a node *N of a tree is the number of nodes in the inheritance path of the tree with the root being the first node and N being the last node in this path. The* **layer of a tree** *is defined to be the maximum layer of all nodes in the tree.*

Figure 2.10 shows a tree with 8 nodes including the root which is node 0. A path

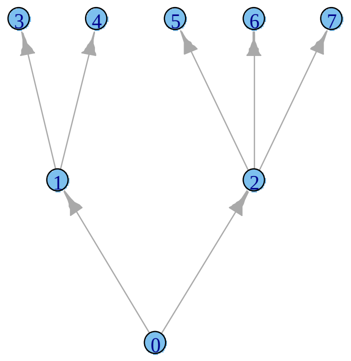


Figure 2.10: A tree structure

from node 0 to node 7 through node 2 is an inheritance path such that the layer of node 2 is 2 and the layer of node 7 is 3. Since the maximum layer of all the nodes in this tree is 3, the layer of the tree is 3.

Definition 2.3.6 *Let T be a component tree with layer m, V_{ij} the vertex set in the j^{th} node with layer i in T, and g_{ij} the complete unity-weighted graph with vertex set V_{ij}. The* **component-generated family of graphs** *for the component tree T is denoted by $\mathcal{F}(T)$ such that $\mathcal{F}(T) = \{g_i\}$, where $i = 1, 2, \cdots m$ and*

$$g_i = \bigcup_{j=1}^{n_i} g_{ij}$$

where n_i is the number of nodes with layer i in T.

A unique monotonic family of graphs can be generated by the above definition. Moreover, if T is a reduced component tree, its component-generated family of graphs is reduced as well.

2.4 Overview of the framework

In the previous sections of the chapter we have completed all the building blocks to construct a new framework of graph families. This framework is shown in Fig. 2.11. In this framework, $\mathcal{G}$ denotes any family of graphs including non-monotonic families;

$$\begin{tikzcd}
\mathcal{G} \arrow[r, "Tot(\mathcal{G})"] & G_w \arrow[r, shift left, "\text{generated}"] & \mathcal{G}_m \arrow[l, shift left, "Tot(\mathcal{G}_m)"] \arrow[r, shift left, "\text{component tree}"] & T_{com}(\mathcal{G}_m) \arrow[l, shift left, "\text{component-generated}"]
\end{tikzcd}$$

Figure 2.11: The framework of a monotonic family of graphs

G_w denotes a weighted graph; $\mathcal{G}_m$ denotes a monotonic family of graphs; $T_{com}(\mathcal{G}_m)$ denotes a component tree. Each arrow in Fig. 2.11 represents a transformation. For example, a weighted graph G_w can be derived from a graph family $\mathcal{G}$ by the total; a monotonic graph family $\mathcal{G}_m$ can be constructed from a weighted graph G_w; a component tree $T_{com}(\mathcal{G}_m)$ can be generated from a monotonic graph family $\mathcal{G}_m$. The framework can be run in reverse, from a component tree to a monotonic family of graphs and a weighted graph. All the above conceptions and transformations have been introduced in the chapter.

Although motivated by clustering, the above framework is independent of clustering. When associating the framework with clustering, $\mathcal{G}$ represents any sequence of partitions on a finite sample and $\mathcal{G}_m$ represents a sequence of nested partitions on a finite sample. If $T_{com}(\mathcal{G}_m)$ is a reduced component tree, it represents a single cluster tree. If $T_{com}(\mathcal{G}_m)$ is not reduced, we can reduce it to produce a cluster tree. Chapter 3 describes in detail the connection of the graph family framework to clustering.

2.5 Applications

From the graph family framework, we derive two methods, graph integration and tree averaging, as well as a tree distance measure.

2.5.1 Graph integration

If a sequence of graphs can form a graph family, we can construct a component tree using graph integration. As shown in Fig. 2.12, the method of graph integration

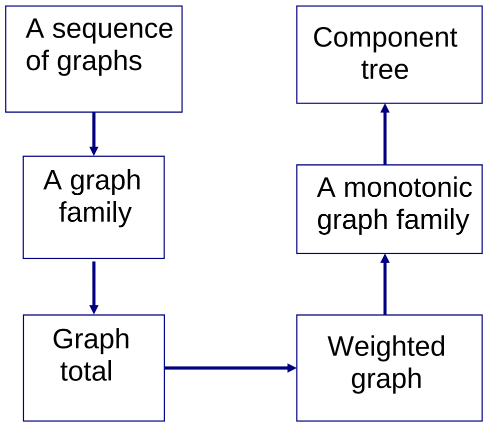


Figure 2.12: Method of graph integration

first constructs a weighted graph from the total of the graph family, then generates a monotonic family of graphs through the methodologies introduced in Section 2.2, and finally produces a single component tree.

Figure 2.13 shows a sequence of graphs with a common vertex set. These graphs

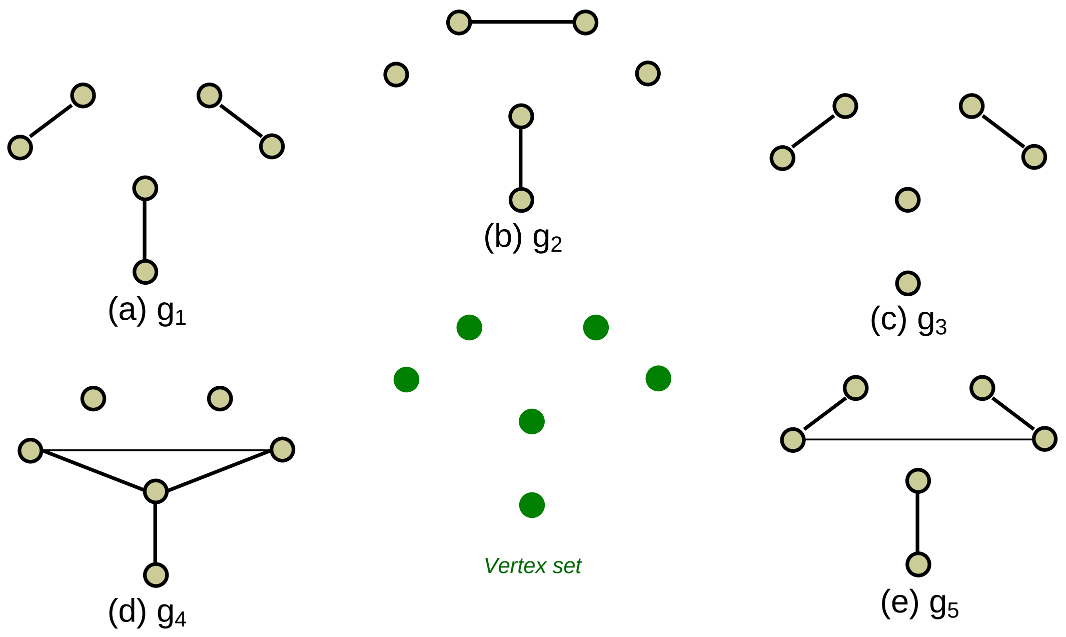


Figure 2.13: A sequence of graphs

form a graph family. Note that some of the connected subgraphs in graphs (d) and (e) in Fig. 2.13 are not complete. This makes the graph family different from that in Fig. 1.8. The weighted graph G_W in Fig. 2.14 is the total of the graph family shown in Fig. 2.13. The monotonic family of graphs formed by graphs (a) to (d) in Fig. 2.14 is the unity-weight-generated family of graphs from G_W. The tree at the bottom right of Fig. 2.14 represents the non-trivial component tree generated from the above monotonic family of graphs.

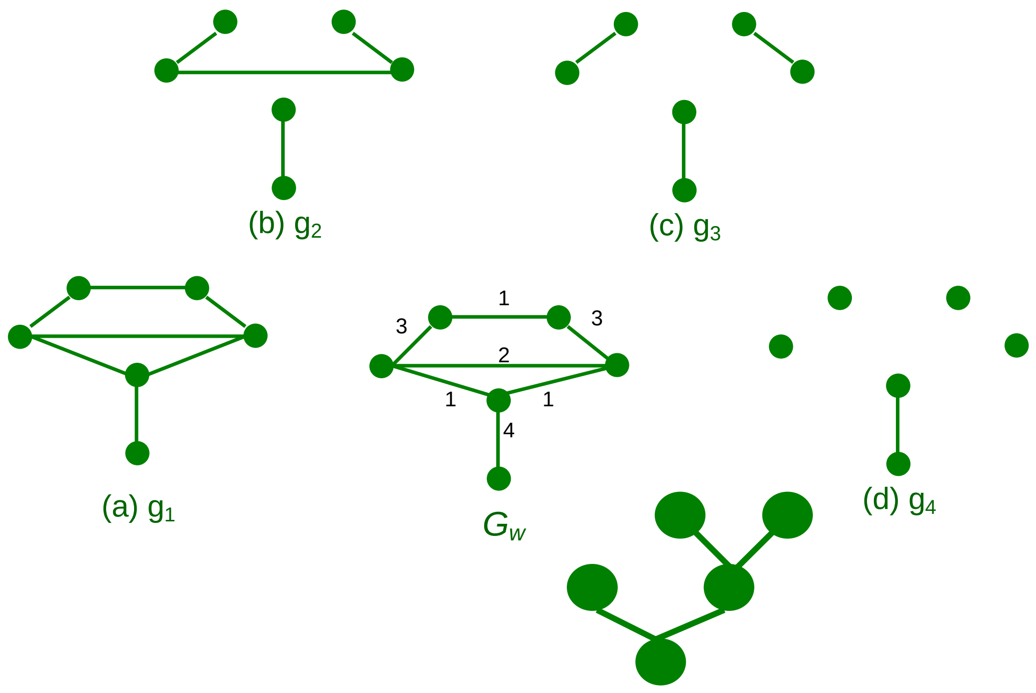


Figure 2.14: Example of graph integration

2.5.2 Tree averaging

If we have a set of component trees such that they have the same root content, as shown in Fig. 2.15, a set of graph families can be constructed. We construct a

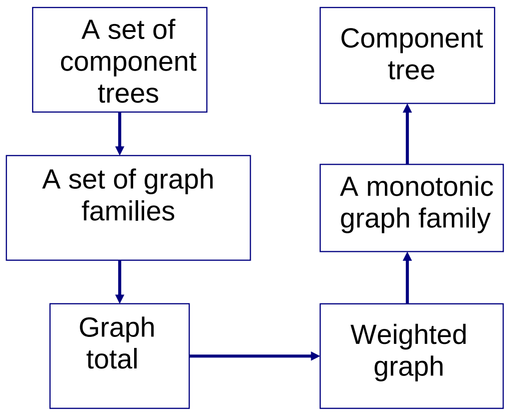


Figure 2.15: Method of tree averaging

weighted graph from the total of each graph family. These weighted graphs form a new graph family and the total of this graph family is a single weighted graph, say G_W. We can also divide the weight on each edge from G_W by the number of graph families. A single component tree can be constructed from the monotonic family of graphs generated from G_W. Note that the tree averaging may not produce a simpler tree structure.

Figures 2.16 and 2.17 show two component trees with the same root content. The roots of both trees contain four points. These four points are shown as a vertex set in the two figures. The roots of both trees have two child nodes. The two child nodes in the first tree contain points 1, 2 and 3, 4 respectively. The two child nodes in the second tree contain points 1, 2, 3, and 4 respectively. Graphs (a) and (b) in each figure represent a monotonic family of graphs associated with each tree.

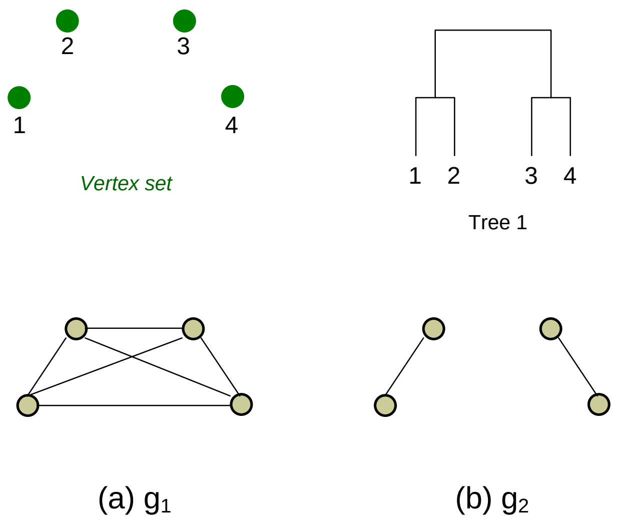
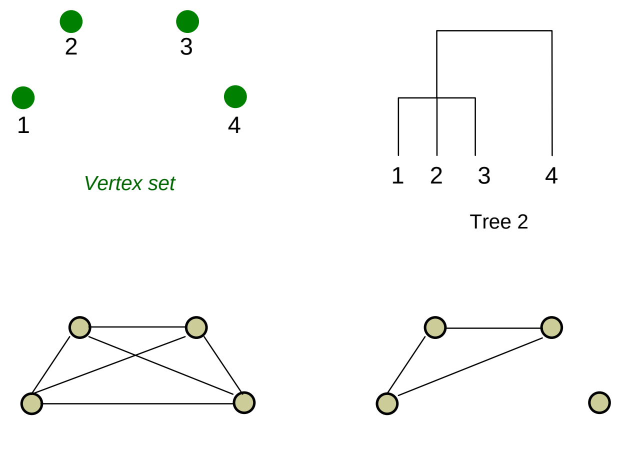


Figure 2.16: A tree and generated graph family

Figure 2.18 shows a component tree generated by tree averaging from the two trees shown in Figs. 2.16 and 2.17. $Gw1$ and $Gw2$ represent the total of the monotonic family of graphs corresponding to each tree. $Gw1$ and $Gw2$ also form a new graph family. Let Gw' be the total of this new graph family. Gw is constructed from Gw' by dividing the weight on each edge by 2. From the definition of a weight-generated family of graphs, a monotonic family of weighted graphs is constructed. This graph family is shown as graphs (a), (b), and (c) in Fig. 2.18. For simplicity, weights are omitted from these graphs. A reduced component tree is generated from this monotonic family of graphs. The first child node of the root in the tree contains two points; the other two child nodes each contain only one point.



(a) g_1 (b) g_2

Figure 2.17: A tree and generated graph family

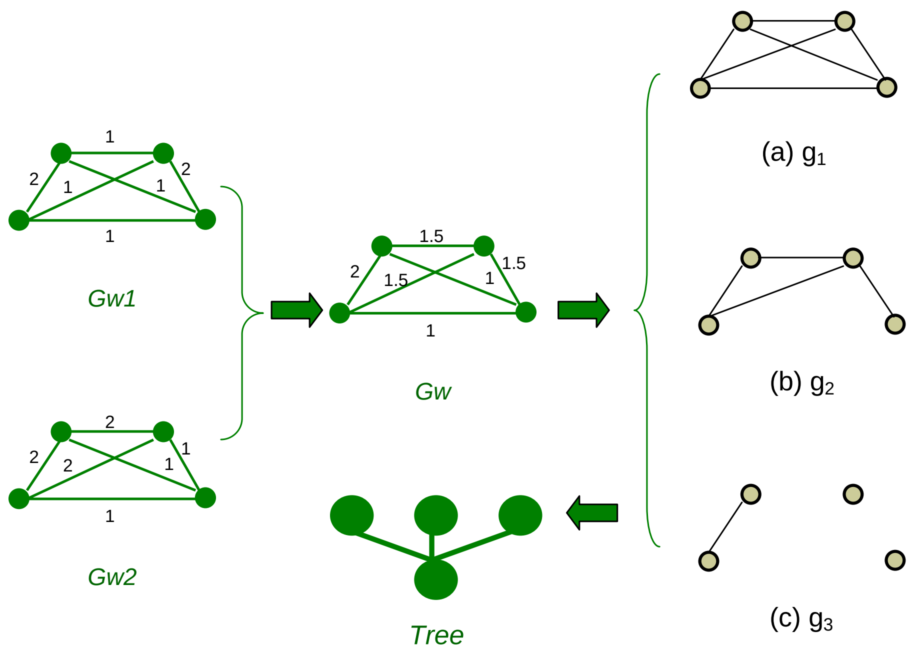


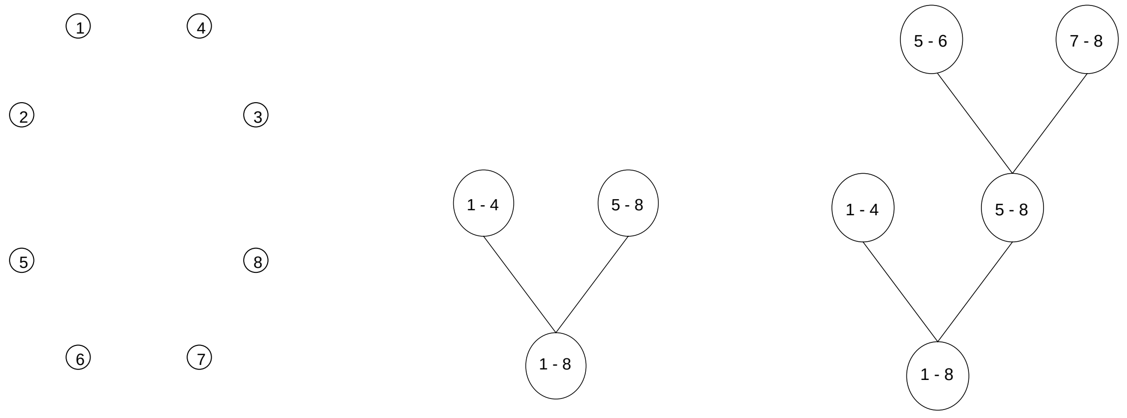
Figure 2.18: Example of tree averaging

2.5.3 A tree distance measure

Based on the graph family framework, a tree distance measure can be defined as follows. Let T_1 and T_2 be two component trees such that their roots have the same content, say D. Their component-generated families of graphs are $\mathcal{F}_1$ and $\mathcal{F}_2$ respectively. Both $\mathcal{F}_1$ and $\mathcal{F}_2$ contain g_0, which is the complete graph of D. Let $W(T_1)$ and $W(T_2)$ be the weight vectors of $Tot(\mathcal{F}_1 - g_0)$ and $Tot(\mathcal{F}_2 - g_0)$ respectively such that the i^{th} elements in both $W(T_1)$ and $W(T_2)$ are the weights on the i^{th} edge in the complete graph with vertex set D respectively. Let $W_{norm}(T_1) = W(T_1)/||W(T_1)||$ and $W_{norm}(T_2) = W(T_2)/||W(T_2)||$, where $||W(T_i)||$ denotes the Euclidean length of $W(T_i)$ and $i = 1, 2$. The distance of T_1 and T_2 can be defined as

$$d(T_1, T_2) = ||W_{norm}(T_1) - W_{norm}(T_2)||.$$

For example, Fig. 2.19 (b) and (c) shows two component trees say T_1 and T_2.

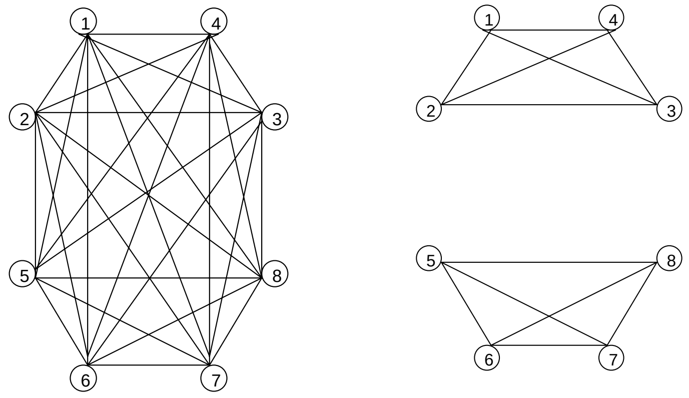


(a) Root content (b) The first tree (c) The second tree

Figure 2.19: Two component trees with the same root content

They have the same root content which is shown in Fig. 2.19 (a). The index value for each point from the root content is also shown in Fig. 2.19 (a). The content of each node in both trees is also shown in Fig. 2.19 (“1-8” denotes points 1 to 8, etc).

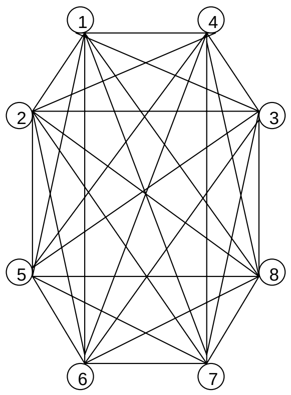
Figure 2.20 and Fig. 2.21 shows the component-generated family of graphs for T_1 and

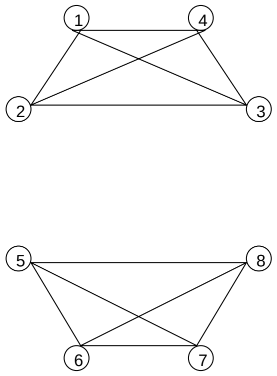


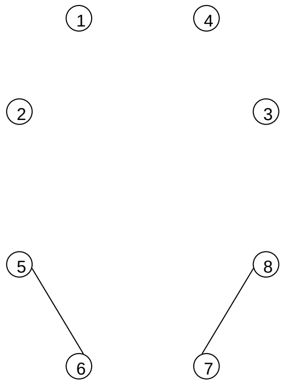
(a) g_0 (b) g_1

Figure 2.20: The component-generated family of graphs, $\mathcal{F}_1$

T_2 respectively. We denote these two graph families $\mathcal{F}_1$ and $\mathcal{F}_2$ respectively. Figure 2.22 shows two weighted graphs $Tot(\mathcal{F}_1 - g_0)$ and $Tot(\mathcal{F}_2 - g_0)$. We denote the weight vectors of these two graph families $W(T_1)$ and $W(T_2)$ respectively. In the example, both vectors contain $\binom{8}{2}$ elements. Since there are twelve edges in $Tot(\mathcal{F}_1 - g_0)$ all with weight 1, $W(T_1)$ contains twelve 1s and sixteen 0s. Similarly, $W(T_2)$ contains two 2s, ten 1s and sixteen 0s. The only difference between $W(T_1)$ and $W(T_2)$ in this example is that the values of the two elements corresponding to points 5,6 and points 7,8 are 2 in $W(T_2)$ but 1 in $W(T_1)$. It is clear, the distance between $W(T_1)/||W(T_1)||$ and $W(T_2)/||W(T_2)||$ is 0.308. Therefore, $d(T_1, T_2) = 0.308$. Note that the distance measure does not depend on the coordinates of each point in the root content.

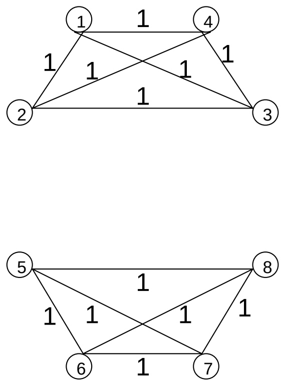


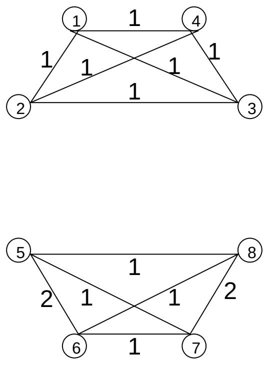




(a) g_0 (b) g_1 (c) g_2

Figure 2.21: The component-generated family of graphs, $\mathcal{F}_2$





(a) $Tot(\mathcal{F}_1 - g_0)$ (b) $Tot(\mathcal{F}_2 - g_0)$

Figure 2.22: Two weighted graphs

In our tree distance measure, we remove g_0 so that the common root of trees T_1 and T_2 does not dominate the difference at higher layer nodes from these two trees.

Our tree distance measure based on two normalized weight vectors depends on the angle between these two vectors. Let $d = d(T_1, T_2)$ be the tree distance from T_1 to T_2. Let $W_1 = W_{norm}(T_1)$ and $W_2 = W_{norm}(T_2)$ be the normalized vectors of the weights constructed from T_1 and T_2 respectively, and α the angle between W_1 and W_2. We have $cos(\alpha) = 1 - \frac{d^2}{2}$. It is common in the literature to use such a $cos(\alpha)$ to remove dependence on vector length. Since $cos(\alpha) = \frac{W_1^T W_2}{||W_1|| \cdot ||W_2||} \geq 0$, we have $0 \leq \alpha \leq \frac{\pi}{2}$. Therefore $0 \leq d \leq \sqrt{2}$, so the maximum possible distance between two trees by our measure is $\sqrt{2}$.

It is clear that, for any trees T_1, T_2, and T_3 with the same finite root content, we have $d(T_1, T_2) \geq 0$, $d(T_1, T_2) = d(T_2, T_1)$, and $d(T_1, T_2) + d(T_2, T_3) \geq d(T_1, T_3)$. The following proposition leads to the result that the measure $d(\cdot)$ is metric for the space of non-trivial trees with the same finite root content.

Proposition 2.5.1 *Suppose T_1 and T_2 are two non-trivial trees with the same root content. Then $d(T_1, T_2) = 0$ if and only if $T_1 = T_2$.*

Proof: It is clear that if $T_1 = T_2$ then $d(T_1, T_2) = 0$. If $d(T_1, T_2) = 0$, we have $W_{norm}(T_1) = W_{norm}(T_2)$. Therefore $W(T_1)$ is proportional to $W(T_2)$ or say $W(T_1) = k \cdot W(T_2)$, where k is a positive constant.

As a tree, any node say N_2 in T_1 except the root has a parent node say N_1 such that $B = Content(N_1) - Content(N_2) \neq \phi$. If the layer of N_1 is i, the weight of every pair of points formed by one point from N_2 and the other from B is i in the weight vector $W(T_1)$. Therefore the distinct elements of $W(T_1)$ are consecutive integers starting from 0. This is also true for $W(T_2)$. So we have $k = 1$ and therefore $W(T_1) = W(T_2)$.

If every node in T_1 appears in T_2 and T_2 does not have any node not in T_1, then T_1 is identical to T_2 because the parent node of a node say N in T_1 must be the parent node of N in T_2. Assume T_1 and T_2 are different with $W(T_1) = W(T_2)$. Let j be the lowest layer such that a node in T_2 at layer j is different from every node in T_1 at layer j. (Note that the layer of the root in a tree is 1 and j must be larger than 1 because T_1 and T_2 have the same root content.) Since every node in T_1 and T_2 contains at least two points, there must exist at least a pair of points say (p, q) such that either (p, q) appears in a node of layer j from T_1 but not in any node at layer j from T_2, or (p, q) appears in a node of layer j from T_2 but not in that from T_1. Either way makes $W(T_1) \neq W(T_2)$, and this conflicts with the above assumption. Therefore, $T_1 = T_2$ if $W(T_1) = W(T_2)$.

Because $W(T_1) = W(T_2)$ if $d(T_1, T_2) = 0$, we have $T_1 = T_2$ if $d(T_1, T_2) = 0$. $\square$

From Proposition 2.5.1 and its proof, it is clear that the above measure without normalization is still metric for the space of non-trivial trees with the same finite root content.

Note that in our tree distance measure, an edge with a large weight means that the edge is preserved for a long time in the tree structure. The edges with large weights represent "fine scale structures". A natural question to ask would be, "have you thought about basing the measure on the inverse of the weights to pick up large scale structures?" Since our major purpose of the tree distance measure is to compare cluster trees, such a "fine scale structure" in a cluster tree represents a possible mode in the density of the underlying distribution, we use large weights on the "fine scale structures".

The methodology for constructing the tree distance measure is illustrated in Fig. 2.23.

An alternative distance between two graphs appears in the literature. It is the graph edit distance [3], which counts the costs of transforming one graph to another by a set of edit operations such as insertion, deletion, substitution, splitting, and merging. Since the sequence of these transformation operations is not unique, the minimum cost from all possible sequences is defined to be the graph edit distance. This makes the algorithm complicated and computationally expensive. Moreover, the strategy that determines the cost of each operation is ad hoc. Note that the graph edit distance is designed for general graphs but not specifially for a tree structure.

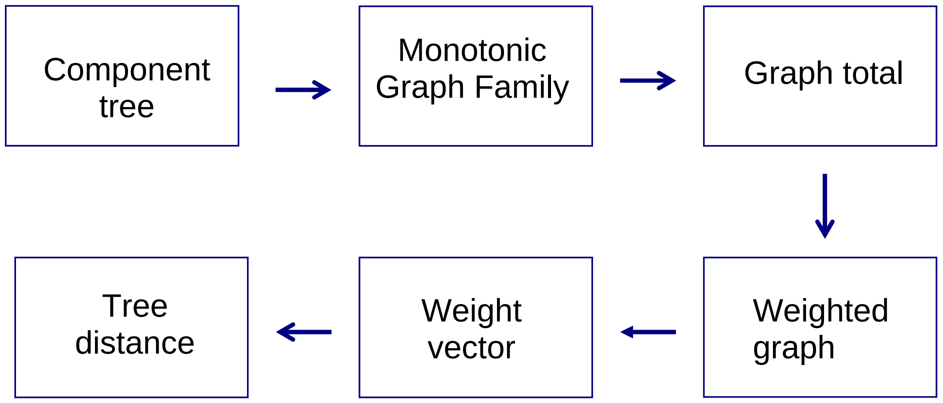


Figure 2.23: A tree distance measure

2.6 Testing the tree distance measure

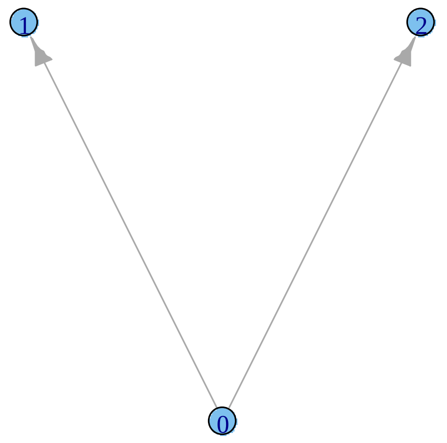
We now explore the properties of the new tree distance and get a sense of its magnitude.

2.6.1 Normalization

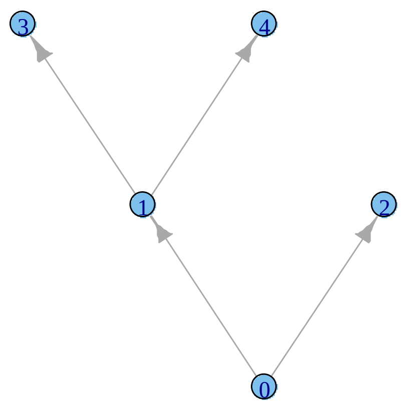
The following two test cases indicate why normalization is useful in the tree distance measure.

Case 1

We construct two trees T_1 and T_2 with the same root. T_1 has only the root and two child nodes with each child node containing half of the content of the root. T_2 has the same first two layers as T_1 and a third layer that contains two nodes, both children of a node in T_2's second layer. Each node in T_2's third layer contains half of the content from its parent node in the second layer. Figure 2.24 shows these two



(a) Structure of T_1



(b) Structure of T_2

Figure 2.24: The structure of T_1 and T_2 for Case 1

trees. We increase the size of their roots and calculate $d(T_1, T_2)$ with different root sizes.

Let $n = |Content(root)|$ be the size of the root. Table 2.1 shows the distance from T_2 to T_1 with and without normalization, with n increasing in increments of 100 from 100 to 1000. The distance without normalization depends on n. The distance with normalization does not depend on n; it converges to a constant. The distance measure with normalization is more reasonable when we compare two distances: one

Distance from T_2 to T_1 with and without normalization										
n	100	200	300	400	500	600	700	800	900	1000
without	24.4	49.4	74.5	99.5	124.5	149.5	174.5	199.5	224.5	249.5
with	0.331	0.332	0.332	0.332	0.332	0.332	0.332	0.332	0.332	0.332

Table 2.1: Distance from T_2 to T_1 in case 1

is calculated for a pair of trees with root size n_1 and the other is calculated for a pair of trees with root size n_2, where $n_1 \neq n_2$. Actually, the scale of the distance measure with normalization does not depend on the root size in each tree but instead on how different the two trees are in their structure.

It is straightforward to calculate the distance between T_1 and T_2 mathematically. Let m be the size of a child node in T_1. Node 2 of T_2 has m points and hence has $\binom{m}{2}$ pairs of points. Similarly, node 1 of T_2 has m points and so $\binom{m}{2}$ pairs of points. Node 1 is further split into two nodes each contains $m/2$ points and hence $\binom{m/2}{2}$ pairs. So there are $\binom{m}{2}$ - 2 $\binom{m/2}{2} = (m/2)^2$ pairs in node 1 that do not appear in either nodes 3 or 4. Therefore in T_2, there are $\binom{m}{2} + (m/2)^2$ pairs having $weight = 1$ and 2 $\binom{m/2}{2}$ pairs having $weight = 2$. Let $d = d(T_1, T_2)$.

Without normalization,

$$d^2 = 2\binom{\frac{m}{2}}{2}(2-1)^2 = \frac{m^2}{4} - \frac{m}{2}.$$

When m is large, $d \approx m/2$. Since $m = n/2$, $d \approx n/4$; d is a linear function of n. With normalization, let $||W_1||$ and $||W_2||$ be the norm of the weight vectors from T_1 and T_2 respectively. We have

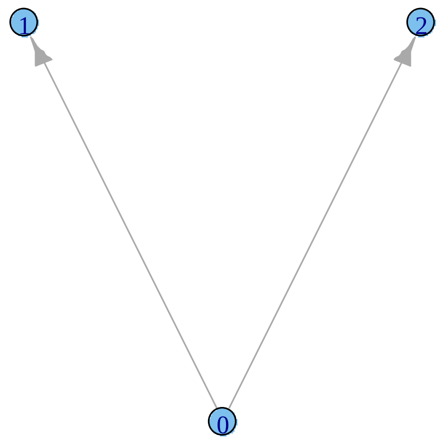
$$d^2 = (\binom{m}{2} + (m/2)^2)\big(\frac{1}{||W_1||} - \frac{1}{||W_2||}\big)^2 + 2\binom{m/2}{2}\big(\frac{1}{||W_1||} - \frac{2}{||W_2||}\big)^2.$$

When m is large, by simple algebra, $d^2 \approx 0.11$. We have $d \approx 0.3319$, which is a constant.

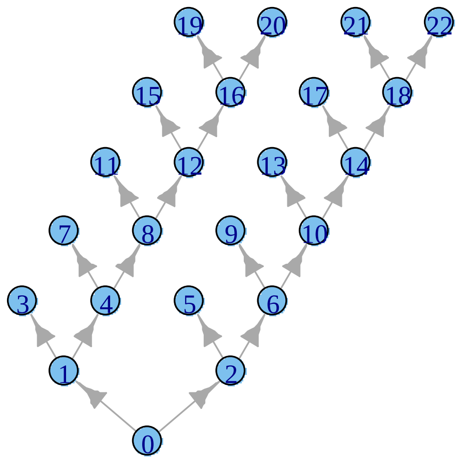
Case 2

We construct a tree T_1 and a sequence of trees $T_2^{(k)}$, $k = 1, 2, \cdots, K$. T_1 and each $T_2^{(k)}$ have the same root content. T_1 has only two child nodes for its root and each child node contains half of the content of the root. $T_2^{(1)}$ has the same first two layers as T_1 and a third layer with four nodes. In the third layer, the first two nodes are formed from the first node in the second layer: one contains only two elements from its parent node and the other contains all the remaining elements from its parent node.

The other two nodes in the third layer are constructed similarly from the second node in the second layer. $T_2^{(2)}$ has four layers with its first three layers identical to $T_2^{(1)}$ and its fourth layer constructed similarly to the third layer of $T_2^{(1)}$. We construct $T_2^{(3)}$, $T_2^{(4)}$, $T_2^{(5)}$, and so on by the same method. Figure 2.25 shows the structure of



(a) Structure of T_1



(b) Structure of $T_2^{(5)}$

Figure 2.25: The structure of T_1 and T_2 for Case 2

T_1 and $T_2^{(5)}$.

Let $n = |Content(root)|$ be the size of the root. We increase n in increments of 100 from 100 to 800 and calculate $d(T_1, T_2^{(i)})$ for each value of n, where $i = 1, 2, 3, 4, 5$. Table 2.2 shows the distance from $T_2^{(i)}$ to T_1 with and without normalization. Similarly to the results from Case 1, the distance without normalization depends on n and i. For each i, the distance with normalization approaches zero as n increases. For each n, the distance with normalization slightly increases as i increases, but this trend is decreasing. The distance measure with normalization is more reasonable than that without normalization if we care more about the common content of nodes from each tree than the complexity of a tree. This is useful when we apply the tree

Distance from $T_2^{(i)}$ to T_1 with and without normalization									
$T_2^{(i)}$	norm	n=100	n=200	n=300	n=400	n=500	n=600	n=700	n=800
$i=1$	without	47.51	97.50	147.50	197.50	247.50	297.50	347.50	397.50
$i=1$	with	0.138	0.099	0.081	0.070	0.063	0.057	0.053	0.049
$i=2$	without	92.05	192.02	292.01	392.01	492.01	592.00	692.00	792.00
$i=2$	with	0.205	0.147	0.120	0.104	0.093	0.085	0.079	0.074
$i=3$	without	133.95	283.89	433.87	583.86	733.85	883.85	1033.8	1183.8
$i=3$	with	0.256	0.184	0.151	0.131	0.117	0.107	0.099	0.093
$i=4$	without	173.23	373.11	573.07	773.05	973.04	1173.0	1373.0	1573.0
$i=4$	with	0.299	0.215	0.176	0.153	0.137	0.125	0.116	0.109
$i=5$	without	209.92	459.69	709.62	959.59	1209.5	1459.5	1709.5	1959.5
$i=5$	with	0.335	0.242	0.199	0.173	0.155	0.141	0.131	0.123

Table 2.2: Distance from $T_2^{(i)}$ to T_1 in Case 2

distance measure in clustering.

Let $||W_1||$ and $||W_2^{(i)}||$ be the norm of the weight vectors from T_1 and $T_2^{(i)}$ respectively. Let $t = ||W_1||$. Without normalization, when n is large and $i << n$, $||W_2^{(i)}|| \approx (i+1)t$. We have $d^2 = d(T_1, T_2^{(i)})^2 \approx t(i+1-1)^2$. Since $t = (n/2)^2 - n/2$, we have $d \approx in/2$, which is a function of both n and i. With normalization, when n is large and $i << n$,

$$\begin{aligned} d^2 &\approx \Big(\frac{i+1}{||W_2^{(i)}||} - \frac{1}{||W_1||}\Big)^2 ||W_1||^2 \\ &\approx \Big(\frac{i+1}{(i+1)t} - \frac{1}{t}\Big)^2 t^2 \\ &\approx 0. \end{aligned}$$

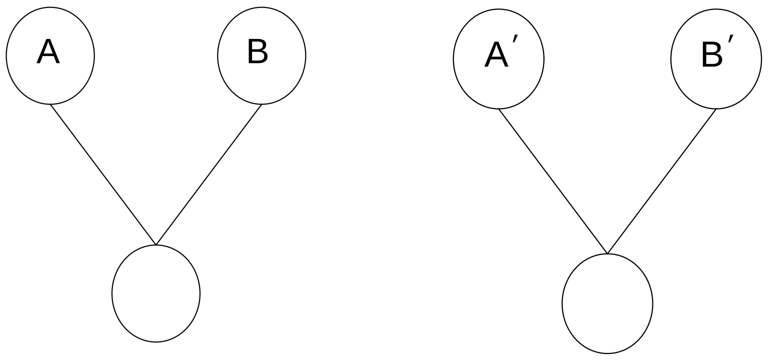
So $d \to 0$ when $n \to \infty$ and $i << n$.

2.6.2 Magnitude

To get a sense of the magnitude of our tree distance measure, we design some test cases.

Case 1

In this case, T_1 is a tree with two layers and its root has just two child nodes. If these two child nodes have the same size, we define the left node to be node A and the right to be node B. If they do not have the same size, we define the child node with less content to be A and the other to be B. T_2 has the same root as T_1 and the same tree structure as T_1: two layers with a root and two children. We define the two child nodes in T_2 to be A' and B'. A' and B' are constructed by randomly assigning each point contained in $Content(A)$ to A' and B'. This assignment follows a Bernoulli distribution with probability p. We then divide points in B into two parts and assign the first part to A', the second part to B' such that $|Content(A')| = |Content(A)|$ and $|Content(B')| = |Content(B)|$, where $|Content(N)|$ is the size of node N. Figure 2.26 shows the structure of these two trees.



(a) Structure of T_1 (b) Structure of T_2

Figure 2.26: The structure of T_1 and T_2 for Case 1

Distance to T_1					
$p \setminus \beta$	0.5	0.4	0.3	0.2	0.1
0.5	1.0010015	0.9616941	0.8516975	0.6864993	0.4688072
0.4	0.9807772	0.9157159	0.7937580	0.6291870	0.4239474
0.3	0.9174330	0.8383855	0.7140340	0.5577147	0.3711185
0.2	0.8008012	0.7196659	0.6039593	0.4656065	0.3062236
0.1	0.6006009	0.5325621	0.4413827	0.3363146	0.2187767

Table 2.3: Distance from T_2 to T_1 in Case 1

Let $m = |Content(A)| + |Content(B)|$, then we have $m = |Content(A')| + |Content(B')|$ as well. Let $\beta = |Content(A)|/m$, then we have $\beta = |Content(A')|/m$ as well. Let K be the number of points assigned to B' in the random assignment process. K follows a Binomial distribution, $Bin(m\beta, p)$. Let d be the distance between T_1 and T_2 calculated by our tree distance measure.

By our tree distance measure, we have

$$\begin{aligned} d^2 &= \frac{2(m\beta - K)K + 2(m(1-\beta) - K)K}{\binom{m\beta}{2} + \binom{m(1-\beta)}{2}} \\ &= \frac{4Q}{m^2 - m - 2m^2\beta(1-\beta)} \end{aligned}$$

where $Q = K(m - 2K)$. When m is large, by simple algebra, we get $d \approx 1$ if $\beta = 0.5$, $p = 0.5$, and $K = E(K) = m/4$.

Table 2.3 shows the distance between T_1 and T_2 for different values of p and β with $m = 1000$. The distance decreases when p and/or β decrease. This is because when p and/or β decrease there are fewer changes from T_1 to T_2.

In Table 2.3, we use the expected value of K to calculate the distance between T_1 and T_2. This distance is not exactly the expectation of the tree distance. However, Table 2.4 indicates that each value in Table 2.3 is close to the true expectation of the tree distance. Table 2.4 shows the quantiles of d when $m = 1000$, $\beta = 0.5$, and $p = 0.5$. These quantiles imply that the distribution of d is not symmetric and the

0.05	0.10	0.25	0.50	0.75	0.90	0.95
0.9970	0.9982	0.9996	1.0005	1.0009	1.00098	1.00099

Table 2.4: Quantiles of $d = d(T_2, T_1)$

variation in d is small compared with its median. We have similar results for other settings of β and p when m is not too small.

Case 2

In this case, we calculate the distance from T_1 to a group of canonical trees. Figure 2.27 shows the tree T_1 with its root denoted by M and the two child nodes denoted

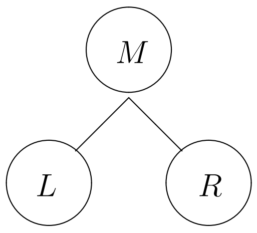
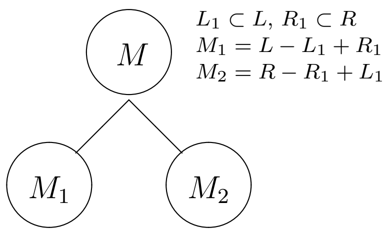


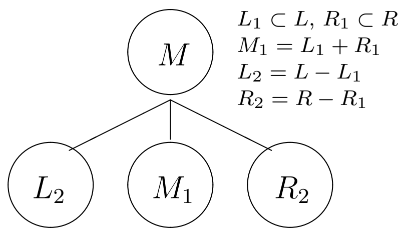
Figure 2.27: The tree T_1

by L and R respectively. Figure 2.28 shows six canonical trees denoted T_2 to T_7. Since our distance measure compares trees with the same root content, we assume that these trees have the same root content.

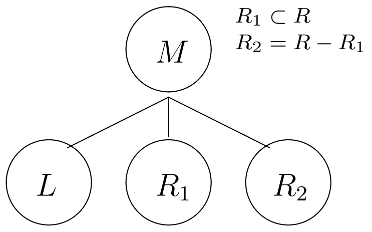
We define the set $M = L + R$ to be the root content for each tree, where L and R are the contents of the left and right child nodes of the root in T_1. In Fig. 2.28, T_2 can be constructed by taking sets L_1 and R_1 from the contents of the left and right child nodes of the root respectively and merging them respectively into the remaining contents of the right and left child nodes of the root. The other canonical trees can be constructed similarly.



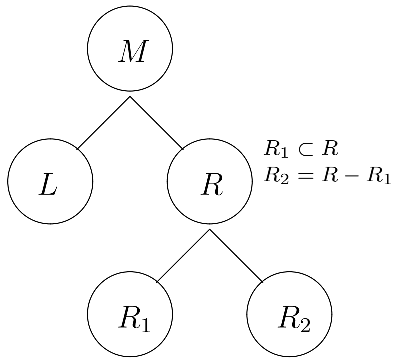
(a) T_2



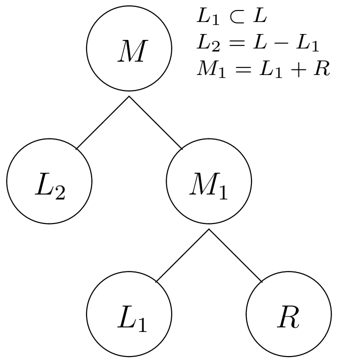
(b) T_3



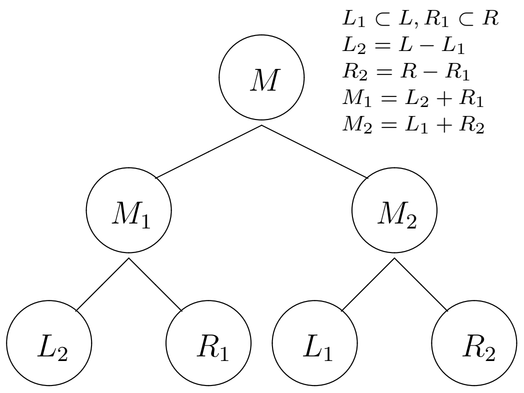
(c) T_4



(d) T_5



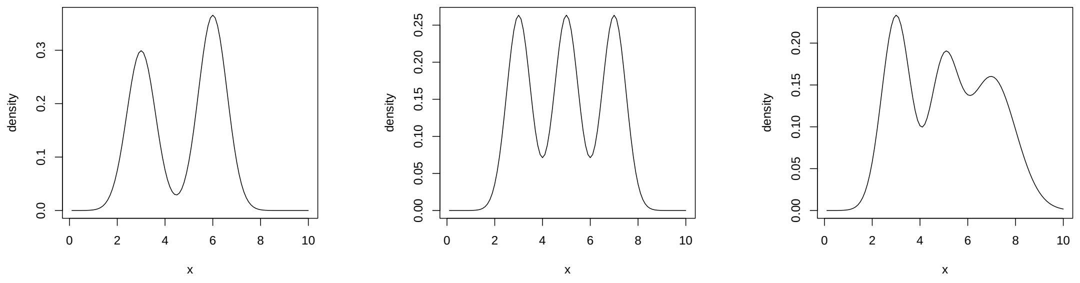
(e) T_6



(f) T_7

Figure 2.28: Six canonical trees

First, the comparison of T_4 and T_5 deserves special consideration. Figure 2.29 shows possible underlying densities for T_1, T_4, and T_5 if each of these trees represents



(a) Density for T_1 (b) Density for T_4 (c) Density for T_5

Figure 2.29: Some underlying densities

a high-density cluster tree. Intuitively speaking, T_5 should be closer to T_1 than T_4. This is because below the density level, say λ_2, at which R_1 and R_2 separate in T_5, it represents a clustering similar to that of T_1. However, T_4, above the density level, say $\lambda_1 < \lambda_2$, at which three modes are found, represents a different clustering from T_1.

To get the exact distance from T_1 to the other trees we need to know the size of each node in these trees. Table 2.5 shows the distances of the canonical trees to T_1 under nine different parameter settings shown in Table 2.6, where $|R|$ denotes the size of set R. In Table 2.5, T_4 and T_5 are closer to T_1 than the others and T_5 is closer than T_4. In most of the cases, T_2 has the largest distance because R and L are mixed well in this tree and no part of R or L has been assigned to an individual node. T_3 is also far from T_1 because it has three child nodes and no R or L has been correctly identified. T_6 is closer to T_1 than T_7 because T_7 has a more complex structure and T_6 has correctly identified R at its third layer. These results agree with our intuition.

From the magnitude tests, we have a sense of the scale of our distance measure. If $d > 1$, the two trees are quite different; if $d < 1$, they are not too far away; if $d < 0.8$ they are close; and if $d < 0.6$ they are very close.

Distance to T_1						
From tree:	T_2	T_3	T_4	T_5	T_6	T_7
Case 1	0.804	0.907	0.411	0.338	0.603	0.667
Case 2	0.937	0.907	0.411	0.338	0.670	0.782
Case 3	0.917	0.722	0.411	0.338	0.562	0.728
Case 4	0.937	0.907	0.520	0.332	0.603	0.782
Case 5	1.005	0.925	0.520	0.332	0.670	0.863
Case 6	0.937	0.766	0.520	0.332	0.562	0.782
Case 7	0.917	0.722	0.411	0.338	0.603	0.728
Case 8	0.937	0.766	0.411	0.338	0.670	0.782
Case 9	0.804	0.634	0.411	0.338	0.562	0.667

Table 2.5: Distance comparison of trees

Setting of parameters	
Case 1	$\lvert L\rvert = 100, \lvert L\rvert/\lvert R\rvert = 1, \lvert R_1\rvert/\lvert R\rvert = 0.8, \lvert L_1\rvert/\lvert L\rvert = 0.8$
Case 2	$\lvert L\rvert = 100, \lvert L\rvert/\lvert R\rvert = 1, \lvert R_1\rvert/\lvert R\rvert = 0.8, \lvert L_1\rvert/\lvert L\rvert = 0.5$
Case 3	$\lvert L\rvert = 100, \lvert L\rvert/\lvert R\rvert = 1, \lvert R_1\rvert/\lvert R\rvert = 0.8, \lvert L_1\rvert/\lvert L\rvert = 0.2$
Case 4	$\lvert L\rvert = 100, \lvert L\rvert/\lvert R\rvert = 1, \lvert R_1\rvert/\lvert R\rvert = 0.5, \lvert L_1\rvert/\lvert L\rvert = 0.8$
Case 5	$\lvert L\rvert = 100, \lvert L\rvert/\lvert R\rvert = 1, \lvert R_1\rvert/\lvert R\rvert = 0.5, \lvert L_1\rvert/\lvert L\rvert = 0.5$
Case 6	$\lvert L\rvert = 100, \lvert L\rvert/\lvert R\rvert = 1, \lvert R_1\rvert/\lvert R\rvert = 0.5, \lvert L_1\rvert/\lvert L\rvert = 0.2$
Case 7	$\lvert L\rvert = 100, \lvert L\rvert/\lvert R\rvert = 1, \lvert R_1\rvert/\lvert R\rvert = 0.2, \lvert L_1\rvert/\lvert L\rvert = 0.8$
Case 8	$\lvert L\rvert = 100, \lvert L\rvert/\lvert R\rvert = 1, \lvert R_1\rvert/\lvert R\rvert = 0.2, \lvert L_1\rvert/\lvert L\rvert = 0.5$
Case 9	$\lvert L\rvert = 100, \lvert L\rvert/\lvert R\rvert = 1, \lvert R_1\rvert/\lvert R\rvert = 0.2, \lvert L_1\rvert/\lvert L\rvert = 0.2$

Table 2.6: Parameter settings for Table 2.5

Chapter 3

Connection of the framework to clustering

3.1 Chapter summary

In Chapter 2, we constructed a new graph family framework. Although this framework is defined and can work independently of clustering, in this thesis both the motivation and purpose of the framework are connected to clustering. As explained in Chapter 1, it plays an important role in our clustering framework. The applications of the graph family framework can also be used for clustering. In Chapter 1, we introduced the connection of a graph family to clustering outcomes. We provided methodologies for and examples of constructing a graph family from a sequence of partitions and a monotonic family of graphs from a cluster tree.

The connection of a graph family to clustering makes it straightforward to use the graph family framework for clustering. A cluster-tree distance measure is derived directly from the tree distance measure introduced in Chapter 2. Applications such as partition integration, cluster tree averaging, and bagging are also available for clustering.

3.2 Partition integration

Section 2.5.1 introduced the method for generating a single component tree from a graph family. If a sequence of partitions for a sample S is derived by one or more clustering methods, we can construct a graph family from these partitions and therefore a single component tree can be constructed. The content of the root of this tree is the sample S. A cluster tree is just the reduced component tree. In Chapter 1, Fig. 1.12 shows the mechanism of this method. We provided an example of partition integration in Chapter 1 using a toy data set. We give more examples in this section.

3.2.1 Example 1

Figure 3.1 shows a sample generated from an artificial continuous distribution. This

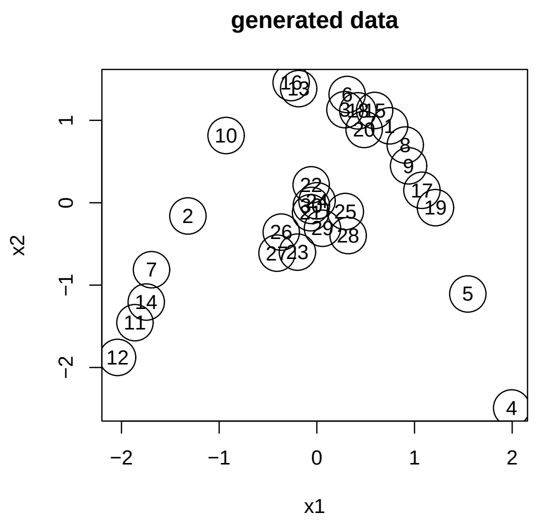


Figure 3.1: A sample for example 1

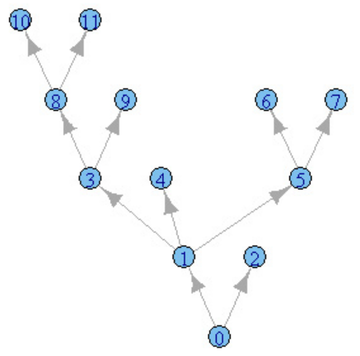
distribution is constructed from a mixture of two sub-distributions.

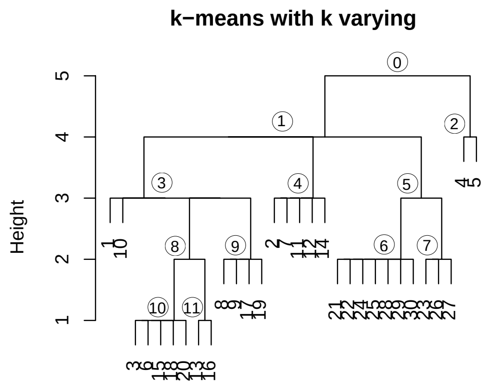
A partition found by k-means cannot capture the features of this distribution. The k-means algorithm finds different partitions by varying the value of k; and in some cases with k fixed, it finds different partitions via different random starts. Either approach generates a family of unnested graphs $\mathcal{G}$. Using partition integration, a single cluster tree can be constructed from these partitions.

Figure 3.2 shows the cluster trees generated by the method of partition integration. The cluster tree shown in (a) is obtained by k-means with k varying from 3 to 8, and (b) is a dendrogram-like plot of this tree. For convenience, we add an index to each node in Fig. 3.2(b) such that the node id is associated with that in Fig. 3.2(a). A dendrogram-like structure is similar to but not the same as a dendrogram. In a dendrogram-like plot, a horizontal bar represents a node. If a sample point joins a horizontal bar by a short vertical bar, it belongs to the node, say N, represented by that horizontal bar and no longer goes to any child node of N. For example, in Fig. 3.2(b) points 1 and 10 are contained in node 3 and its parent node (node 0), and they are not contained in any child node of node 3. Figure 3.2(c) shows another cluster tree found by k-means with k fixed at 5 and (d) is a dendrogram-like plot of the tree in (c).

Both cluster trees in Fig. 3.2 tend to group each dense part of the observations from the rainbow-shaped region into clusters and to group the observations from the dense centre-based region under the rainbow into another cluster. It is better than a partition constructed by running k-means just once with a specific value of k such as $k = 2$. As shown in Fig. 3.3, the partition from k-means with $k = 2$ mixes observations from the two regions into a large cluster and also generates a small cluster with only five points (points 2, 7, 11, 12, and 14). However, the main point of this example is to show how a cluster tree is constructed from partitions.

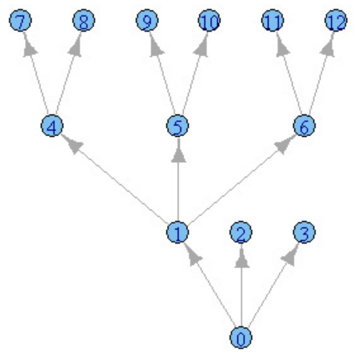
Figure 3.4 shows another cluster tree for the sample in Fig. 3.1. This tree is generated through the method of partition integration by multiple runs of DBSCAN with $Minpts$ fixed at 4 and ϵ varying. As discussed in Appendix A, running DBSCAN in this way generates a sequence of nested partitions that forms a monotonic family of graphs. The cluster tree in Fig. 3.4 is just the reduced component tree generated from this monotonic family of graphs. From Fig. 3.4, we see intuitively that DBSCAN is inherently better at discovering patterns in this sample.

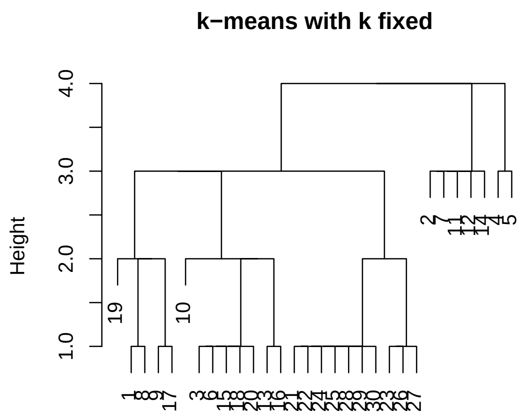




(a) Cluster tree

(b) Dendrogram-like plot





(c) Cluster tree

(d) Dendrogram-like plot

Figure 3.2: Generation of a cluster tree through k-means

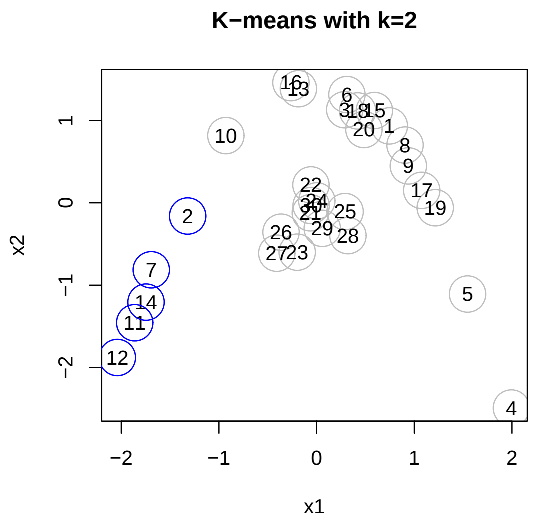
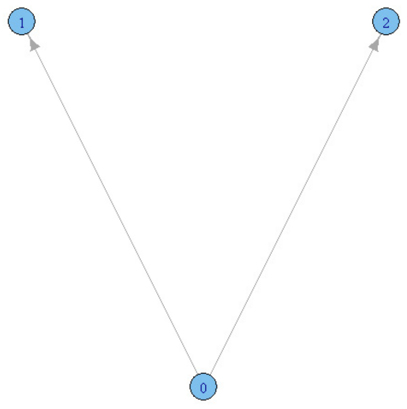
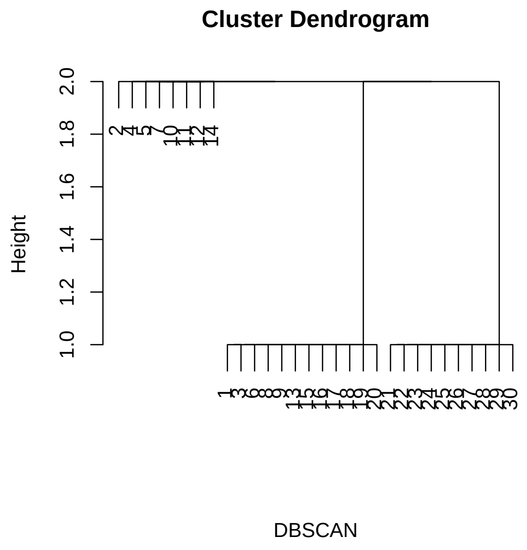


Figure 3.3: Partition using k-means with k=2



(a) Cluster tree



(b) Dendrogram-like plot

Figure 3.4: A cluster tree from DBSCAN

3.2.2 Example 2

Figure 3.5 shows a sample generated from another artificial continuous distribution.

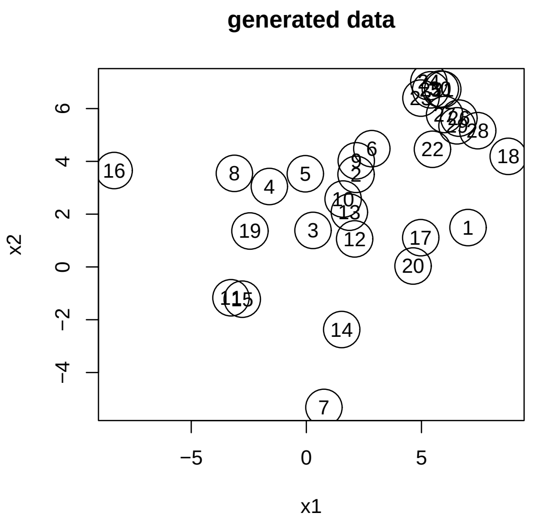
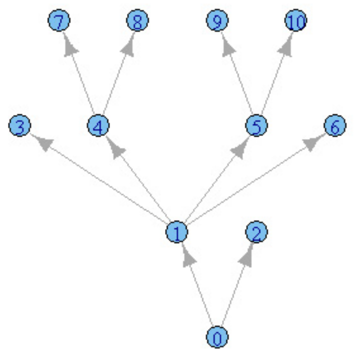
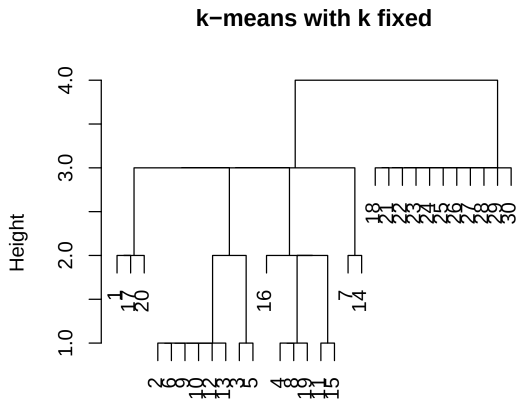


Figure 3.5: A sample for example 2

As in the previous example, this distribution is constructed from a mixture of two sub-distributions. Figure 3.6 shows a cluster tree generated by k-means with k fixed at 5 then random starts 6 times; it tends to group the observations from the two different sub-distributions. This result is better than the partition from k-means with $k = 2$ which mixes up points from the two sub-distributions.





(a) Cluster tree (b) Dendrogram-like plot

Figure 3.6: Generation of a cluster tree using k-means

3.2.3 Comparison to existing methods

There exist some approaches to generate cluster trees from non-nested partitions. The method proposed by Carroll et al. [13] turns a sequence of non-nested partitions into a tree or multiple trees. The method first generates a graph with each vertex relating to a cluster from the input partitions and each edge corresponding to the relation of two clusters in terms of disjoint. A similar graph is generated in terms of inclusion. The union of the above two graphs is called a nesting graph. A maximal complete subgraph, i.e. a clique, in the nesting graph represents a cluster tree. The method then finds an optimal set of cliques in the nesting graph.

Carroll's method takes partitions as the input. The number of vertices in the nesting graph depends on the number of different clusters from the input partitions. If there are too many different clusters, the vertex set is large and the search for optimal cliques is computationally expensive. If a cluster with a certain point appears multiple times from the input partitions, there is only one vertex in the graphs corresponding to this cluster. It makes no difference if a cluster appears just once or multiple times. Furthermore, the method produces multiple trees if the optimal set of cliques contains more than one element. This means that in some cases there is no way to produce a single cluster tree for the whole sample.

The multiple-clustering method proposed by Quan et al. [56] generates a new partition from a sequence of input partitions calculated using different clustering methods for a sample of size n. The method first constructs a vector for each sample point. Each vector has m elements, where m is the number of input partitions. The i^{th} element of the vector for the sample point j is the cluster number for this point in the i^{th} input partition. The multiple-clustering method then uses these vectors as an embedded sample and uses a partitioning method such as k-means to find a new partition from the embedded sample. This multiple-clustering method is different from our method of partition integration. Quan's method relies on an arbitrarily chosen partitioning method to obtain an integrated partition from the embedded sample.

Ashlock et al. [5] propose another multiple-clustering method. They calculate the number of times that each pair of sample points appears in the same cluster in the partitions derived from multiple runs of k-means on the sample. They then construct a graph, say G_a, by adding an edge to connect each pair of points if the number of times they appear together in a cluster is greater than some threshold value.

In our graph family framework, we can construct a weighted graph, say G_W, from the graph total of all the graphs constructed from partitions by multiple runs of k-means. If the weight on each edge of this graph is 1, then by our definitions, both the weight-generated and the unity-weight-generated family of graphs from G_W contain the graph G_a constructed by Ashlock's method. This implies that the partition found by Ashlock's method is contained in the cluster tree generated by our method. Furthermore, the outcome from our method is a cluster tree rather than another partition.

Other approaches accumulate information from multiple runs of k-means. Evidence accumulation clustering [24] is one such approach. The method considers the number of times that a pair of points appears in the same cluster to be a similarity between them, and runs a single linkage on the dissimilarity matrix constructed from the similarities between every pair of points.

All the above approaches focus on employing features from a specific clustering method whereas our method of partition integration is more general and can take as input partitions from one or more clustering methods. The graph family framework makes our method more general.

3.3 Combining cluster trees

The method of partition integration is derived from the method of graph integration, and similarly we propose cluster tree averaging directly from the method of component tree averaging. We also propose a cluster tree bagging using bootstrapped samples.

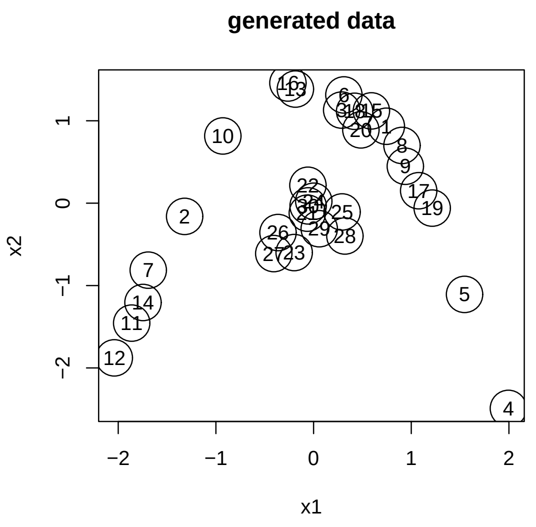
3.3.1 Cluster tree averaging

Model averaging is a common application in model inference; it is often used for supervised learning. We are interested in such an application for clustering.

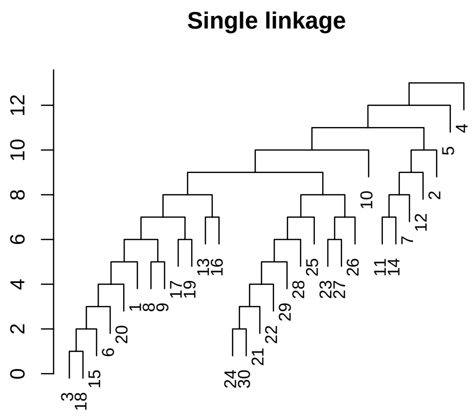
Section 2.5.2 introduces a method for generating a single component tree from multiple component trees. Since a set of component trees can be obtained from a set of cluster trees if they are constructed for a fixed finite sample, a single cluster tree can be obtained in the end. Figure 1.15 in Chapter 1 shows the mechanism of cluster tree averaging.

As a committee method, cluster tree averaging gives an equal weight to each tree. The usual way to obtain weights for model averaging in supervised learning is through the control of squared-error loss. However, in clustering, we do not have a training set with class labels, so there is no obvious way to set up weights for our averaging.

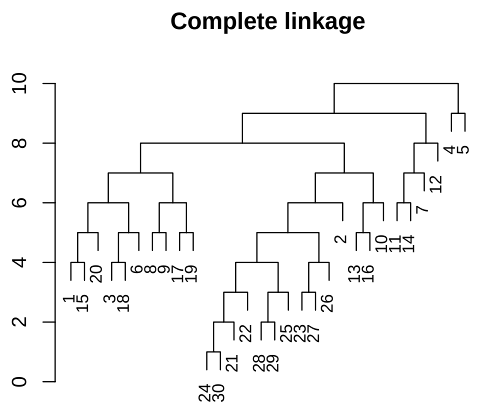
Figure 3.7 shows an example of cluster tree averaging. Figure 3.7(a) shows the sample from Fig. 3.1. We index all the points in the sample. Figures 3.7(b) and (c) show the dendrogram-like cluster trees constructed by single linkage and complete linkage respectively. Figure 3.7(d) is the cluster tree averaging of these two cluster trees. The tree averaging in this case reduces the layers. Note that cluster tree averaging may not always produce a simpler tree structure.



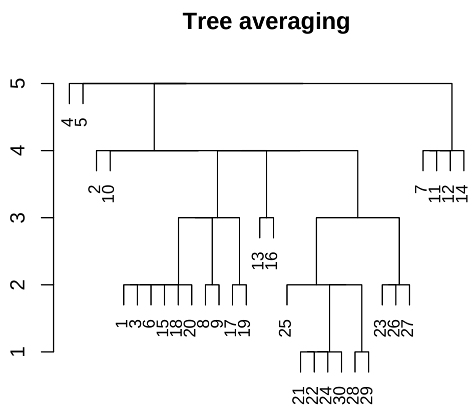
(a) Sample with index



(b) Single linkage



(c) Complete linkage



(d) Cluster tree averaging

Figure 3.7: Example of cluster tree averaging

3.3.2 Cluster tree bagging

Bagging was introduced to reduce the variation of an estimator [11]. Combining cluster tree averaging with the idea of bagging produces a new method, cluster tree bagging.

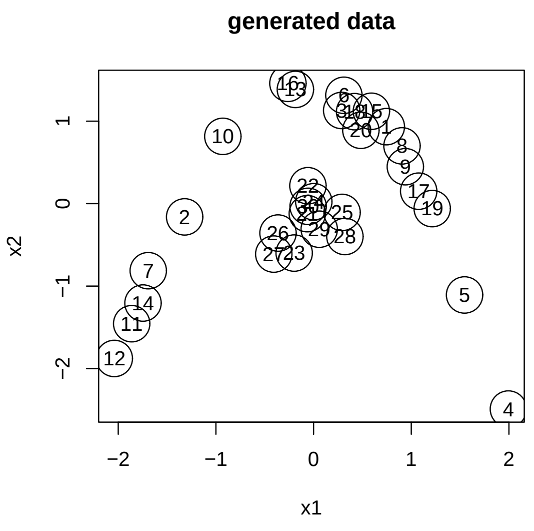
For a fixed finite sample, we use bootstrapping to obtain a sequence of bootstrapped samples. For each bootstrapped sample, we construct a cluster tree using a single clustering method. We therefore have a set of cluster trees. Using the same strategy used for cluster tree averaging, we construct a single cluster tree in the end. Figure 1.16 of Chapter 1 shows the mechanism of this method. We discuss cluster tree averaging and bagging in more detail in Chapter 6.

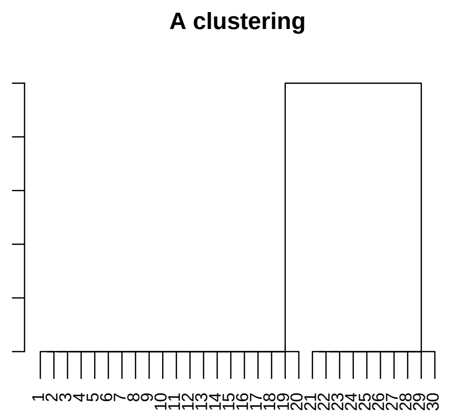
3.4 Cluster-tree distance measure

Since a cluster tree is also a reduced component tree in our graph family framework, the tree distance introduced in Section 2.5.3 can be used to form a cluster-tree distance measure. The mechanism of this measure is shown in Fig. 1.11 of Chapter 1.

If a clustering separates the points from the two regions in the sample shown in Fig. 3.8(a), we can construct a cluster tree with two layers from this clustering and denote this tree T_0. The dendrogram-like cluster tree of T_0 is shown in Fig. 3.8(b). We denote the cluster tree shown in Fig. 3.2(b) T_1 and the 2-layer cluster tree corresponding to the partition shown in Fig. 3.3 T_2. Both figures are from Section 3.2.1. Since T_0, T_1, and T_2 are constructed from the same sample, we can calculate the distances among them using our cluster-tree distance measure. The distance from T_2 to T_0 is 0.9024; the distance from T_1 to T_0 is 0.6761. From our tests on tree distance magnitude in Section 2.6.2, T_1 is much closer to T_0 than T_2. This result makes sense intuitively.

The distance measure can be written as an inner product or a kernel between any two trees T_X, T_Y if they have the same root content. Let $\mathcal{F}_X$ and $\mathcal{F}_Y$ respectively be





(a) Sample with index (b) T_0

Figure 3.8: A clustering with two clusters

their component-generated families of graphs. Let

$$K_{norm}(T_X, T_Y) =< \psi_{norm}(T_X), \psi_{norm}(T_Y) >,$$

where $\psi_{norm}(T_X) = \psi(T_X)/||\psi(T_X)||$, $\psi_{norm}(T_Y) = \psi(T_Y)/||\psi(T_Y)||$, and $\psi(T_X)$ and $\psi(T_Y)$ are the weight vectors of $Tot(\mathcal{F}_X - g_0)$ and $Tot(\mathcal{F}_Y - g_0)$ respectively. We have

$$\begin{aligned} d(T_X, T_Y)^2 &= K_{norm}(T_X, T_X) + K_{norm}(T_Y, T_Y) - 2K_{norm}(T_X, T_Y) \\ &= 2 - 2K_{norm}(T_X, T_Y). \end{aligned}$$

Note that, the inner product we introduced in this section is constructed based on the weight vectors corresponding to trees, this does not imply a vector space on trees.

There are existing methods to measure the distance or similarity between two trees or two clusterings.

3.4.1 Comparison to other approaches

As we have already shown, we can write our cluster-tree distance measure using an inner product or kernel. There exist similar approaches. Taylor and Cristianini [64] show a way to compare trees based on subtree kernels. In their method: $\phi_s(T) = 1$ if s is a subtree of T, $\phi_s(T) = 0$ otherwise, and the kernel is $k(T_1, T_2) =< \phi_s(T_1), \phi_s(T_2) >= \sum_{s \in \tau} \phi_s(T_1)\phi_s(T_2)$, where τ is the set of all trees containing at least two layers. This subtree kernel is actually the total number of subtrees contained in both T_1 and T_2.

This approach is similar to ours as follows:

- Both methods can be written in an inner product or kernel form.
- Both kernels perform an embedding map from the space of all finite trees to a vector space.

The two approaches are different in that:

- The two kernels are different.
- Their kernel embedding contains all finite subtrees, whereas our embedding is fixed from a tree to a vector in $\mathcal{R}^m$, where m is n choose 2 and n is the size of the root.
- In their kernel, a subtree can contribute 1 only if it is contained exactly in both T_1 and T_2, which means that if a subtree from T_1 is slightly different from a subtree of T_2, the two subtrees can not contribute to the kernel no matter how similar they are. In our kernel, we count the contribution from the edge between each pair of data objects, such that two similar nodes in the tree can make a contribution if they contain many common edges.

From the above comparison, Taylor and Cristianini's kernel is designed for symbol strings or sequences of symbols, such that the edges in the tree are normally labelled by symbols, and each node in the tree represents a string or sequence of symbols.

Contributions to their kernel rely on identical nodes and therefore identical subtrees from two different trees.

Our kernel is simpler with a fixed size in the embedding vector space. Moreover, our kernel looks at the details inside each node and is more suitable for comparing cluster trees. In many cases, the likelihood of the same node being contained in two cluster trees constructed by different clustering methods is low, especially when the node is large. A kernel constructed from common edges rather than common subtrees is more reasonable in these cases.

The Rand Index measure is an early approach for measuring two partitions [57]. However, it can not measure cluster trees. Ben-Hur et al. [9] proposed a dot product or kernel to measure the similarity between two partitions of the same sample. This kernel is a special case of our kernel. It compares two trees built on just two partitions. Such a tree can be constructed with a root and several child nodes, each as a maximally disjoint subset in a partition. Ben-Hur's approach is a special case of ours. However, our method is derived from the graph family framework and therefore can measure cluster trees and not just partitions. Ben-Hur's approach does not have such a framework.

Fowlkes and Mallows [20] developed a method to calculate the similarity between two hierarchical clusterings. Their method cuts a cluster tree into a sequence of partitions with different numbers of clusters. Two cluster trees are compared using two sequences of partitions with the same set of values for the number of clusters for each partition in the sequence. They calculate the similarity between each pair of partitions from two trees with the same number of clusters in both partitions. The comparison is done by a plot of the similarity values versus the number of clusters in each pair of partitions.

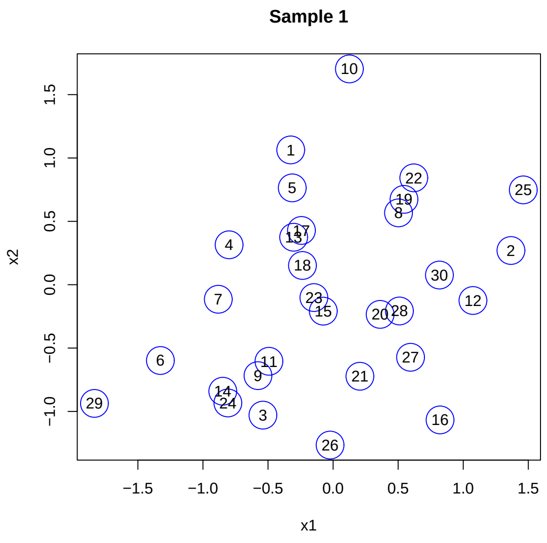
Fowlkes and Mallows's method does not give an outcome for the distance or similarity measure between two cluster trees. The method gives only a sequence of similarities between pairs of partitions cut from two cluster trees. This affects the use of their measure in more general applications.

3.4.2 An examination

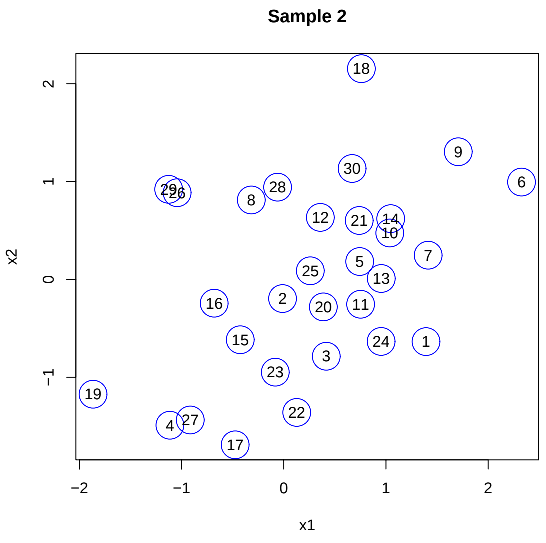
To confirm the performance of our tree distance measure, we did Monte Carlo sampling experiments similar to those carried out by Fowlkes and Mallows. They used only two clustering methods: single linkage and complete linkage. We used more methods. Our experiments used the configurations described by Fowlkes and Mallows.

Irrelevant sampling

These experiments compare the distance or similarity between two clusterings constructed from two irrelevant samples with an index of the sample points. Figure 3.9 shows a pair of irrelevant samples with an index of the sample points. Here "irrele-



(a) Sample 1



(b) Sample 2

Figure 3.9: Example of two independent samples

vant" indicates that a pair of points with the same index value from two independent samples is unrelated. In Fig. 3.9(a), the location of point 1 is quite different from the location of point 1 in Fig. 3.9(b). The same is true for most of the other points. If we use a method such as single linkage to construct a cluster tree for each sample in Fig. 3.9 and use the index value to represent each point in a sample, we can calculate the distance between the two cluster trees using our clustering distance measure. Clearly this distance should be large.

[1. One mode case]

We draw 20 pairs of independent random samples of size 100 from a bivariate Normal F_0, with $\mu = [0, 0]^T$ and covariance matrix

$$V = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$$

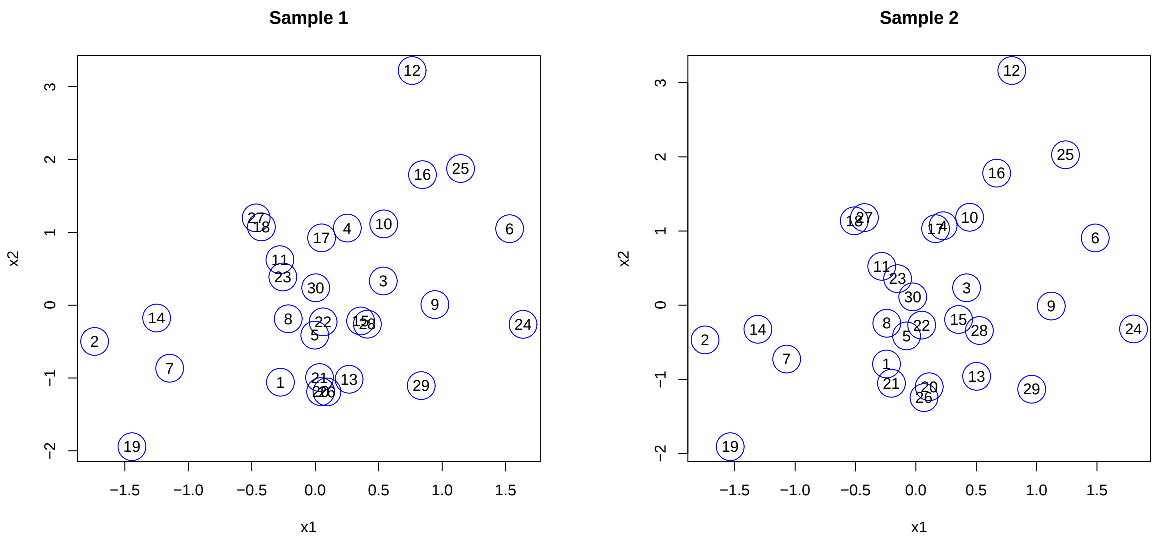
For each pair of samples, we construct a pair of cluster trees using different clustering methods. Note that a brief description of every clustering method used for these experiments is given in Appendix E. We calculate the distance between each pair of cluster trees using our tree distance measure.

[2. Two modes case]

We draw 20 pairs of independent random samples of size 100 from a mixture of two bivariate Normal distributions. We denote this mixture $F = 1/2F_1 + 1/2F_2$, where F_1 is the same as F_0 and F_2 is the same as F_1 except its mean vector is $[3, 3]^T$. We calculate the distance between each pair of cluster trees as for the one mode case.

Perturbation sampling

In perturbation sampling, we first draw a random sample, say S, from a distribution and assign an index value to each point in S, then obtain a sample S' such that the point in S' with index value i is obtained by adding a small random deviate to the point in S whose index value is i. Each pair of points from S and S' with the same index value is similar and the similarity is determined by the scale of the random deviates. Figure 3.10 shows a pair of two samples with perturbation. The location of each point in Fig. 3.10(a) is close to the location of the point with the same index value in Fig. 3.10(b). If we use a clustering method to construct a cluster tree for each sample in Fig. 3.10 and use the index value to represent each point



(a) Sample 1 (b) Sample 2

Figure 3.10: Example of two samples with perturbation

in a sample, we can calculate the distance between the two cluster trees using our clustering distance measure. Clearly this distance should be small. Fowlkes and Mallows briefly noted that the stability of clustering can be studied by controlling the scale of the perturbation, i.e. the value of the deviates.

[1. One mode case]

We first draw 20 individual samples from the same bivariate Normal used for the irrelevant sampling, then perform the following perturbation. For each sample S_1, let (x_i, y_i) be the i^{th} object in the sample. We draw two sample points from a Normal with mean zero and standard deviation $\delta_\epsilon = 0.03$, then add these two points to x_i and y_i respectively to produce another sample S_2. S_1 and S_2 form one pair and we construct 20 pairs of such samples. We then calculate the distance between each pair of cluster trees. We then repeat this perturbation experiment with a larger perturbation, i.e. $\delta_\epsilon = 0.09$.

[2. Two modes case]

The experiments are similar to the above one mode case, but the original 20 samples are drawn from the same mixture of two bivariate Normal distributions used in the irrelevant sampling experiments.

The results

For each clustering method and each experiment setting, we get the distances between 20 pairs of cluster trees constructed from 20 pairs of samples.

For the one mode cases, for each clustering method, we draw 3 boxplots for 3 different experiment settings. Each boxplot is constructed from the 20 distances obtained from the corresponding experiment setting. Figure 3.11 shows these boxplots. Sampling method "1" is irrelevant sampling; method "2" is perturbation sampling with $\delta_\epsilon = 0.09$; and method "3" is perturbation sampling with $\delta_\epsilon = 0.03$.

For each method except the Gaussian partition, the median of the distances from irrelevant sampling is the largest, and the median of the distances from perturbation sampling with $\delta_\epsilon = 0.03$ is the smallest. This result is reasonable because the pair of samples from irrelevant sampling is the most different and the pair of samples from perturbation sampling with smaller δ_ϵ is the most similar. The Gaussian partition method finds just one cluster almost every time because the samples are drawn from a one-mode distribution. The distances from single linkage are small compared with the others. This is because single linkage can make small clusters joined with large clusters and therefore the differences between large clusters from the two trees constructed by each pair of samples are small, no matter how the samples are drawn. This indicates that single linkage is a stable method. However, single linkage is not always suitable for forming clusters from samples of mixture Gaussians. We demonstrate this in our clustering performance comparisons in Chapter 6.

The variations in the distances between trees constructed for each pair of samples reveal some interesting patterns. For most methods, the distances from the irrelevant sampling have the smallest variation. These distances, except those from the Gaussian partition and single linkage, are all close to or above 1, which indicates "far away". Since they are all far away, they have small variations. This is the expected result.

For the two-mode cases, we draw similar boxplots for each clustering method. Figure 3.12 shows these boxplots. The same pattern of three boxplots occurs for each method including the Gaussian partition. In contrast to the one-mode case, for most of the methods in the two-mode case, the variation from the larger perturbation is greater than that from the smaller perturbation. This result is reasonable because the samples from two-mode cases have a more “clusterable” pattern than those from just one mode. The methods that are designed for discovering cluster “patterns” will capture this change in the number of modes in the samples. In the one-mode case, the small perturbation can make a pair of trees much closer than the large perturbation does, but the lack of “pattern” can also make the distances for some of the pairs of trees from the small perturbation not much different from those from the large perturbation. This makes the variation in the distances larger for some methods in the one-mode case.

The above experiments show that our cluster-tree distance measure performs well and makes sense. As mentioned by Fowlkes and Mallows, perturbation sampling can be used to study the stability of clustering. The above experiments verify the capability of our measure in this respect.

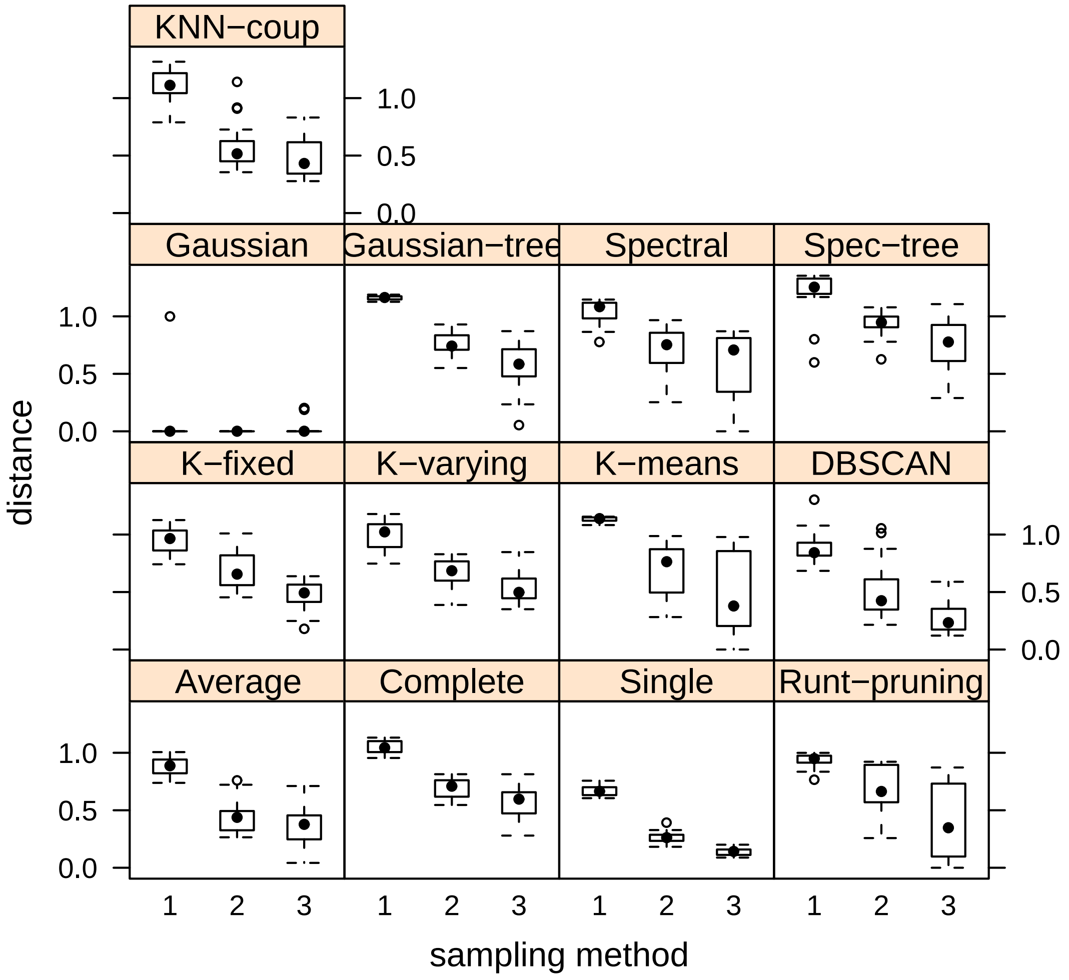


Figure 3.11: Stability test on samples from one-mode distribution

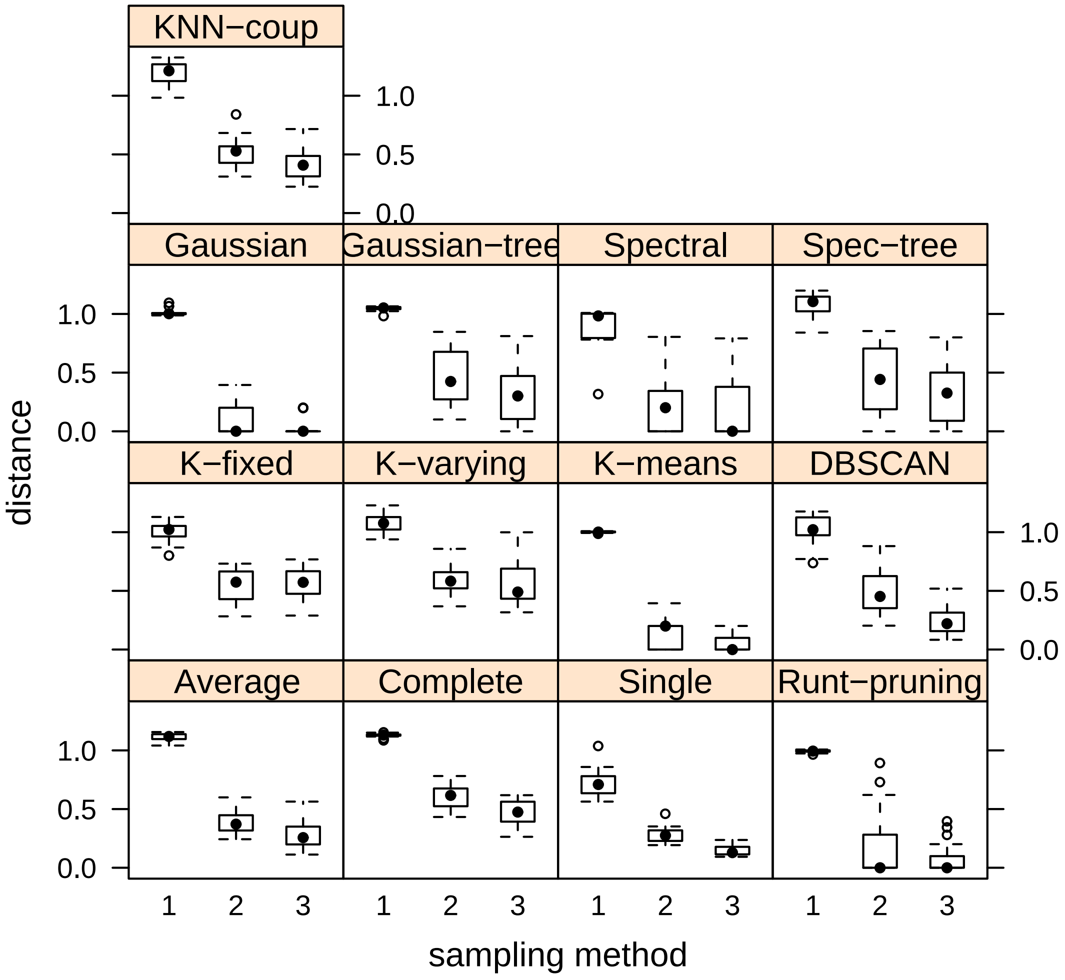


Figure 3.12: Stability test on samples from two-mode distribution

Chapter 4

A cluster tree parameter

4.1 Chapter summary

A high-density cluster tree [31, 62] describes a characteristic of a continuous distribution. We use this tree as a starting point to develop a meaningful clustering parameter of interest for the underlying distribution. We define this tree formally in this chapter.

The high-density cluster tree is a feature of the underlying distribution if it is continuous. Although it can not determine a distribution, it can be regarded as a parameter of interest. In this chapter, we give some properties of a high-density cluster tree.

The high-density cluster tree is a population parameter to be estimated from a sample. Several recent methods in the literature can be understood and studied as estimators of this tree. This approach has so far required the underlying distribution to be continuous. Building a cluster tree for a discrete distribution is an open problem and presents some interesting conceptual and computational challenges.

We define a high-probability mass cluster tree for a discrete distribution. We do it only for discrete distributions whose support is a subset of $\mathcal{R}^d$ and for which a distance measure exists. For simplicity, we will not mention this again in the thesis. The definition of a high-probability mass cluster tree makes our clustering parameterization more general.

To define a high-probability mass cluster tree, we need methodologies to determine contiguity among points with a positive mass. Without further exploration, Hartigan proposed a way to determine the connectivity of points in a discrete distribution whose support is the vertex set of a regular lattice [32]. His idea is that two points with a high-probability mass are connected if they share a common edge in the lattice. Hartigan's approach does not work for arbitrary discrete distributions.

To solve the problem of contiguity, we propose two methods. In the first method, we use a four-spring model to determine the contiguity of two points. This model is natural and easy to explain. However, it is suitable only for discrete distributions in a 2-d space and not cheap in computation. In the second method, we use a coupling graph to determine the contiguity among points. We further discuss some properties of the coupling graph method via propositions and theorems. We find that Hartigan's approach is a special case of our coupling graph method.

The coupling graph method is more general and cheap in computation. It can construct a cluster tree on a distance matrix. We also discuss some convergence properties for the coupling graph method in this chapter.

4.2 High-density cluster tree

Given the density function f of a continuous distribution, the density level set defined by Hartigan [31, Section 11] is a good way to construct a tree structure from f. For convenience, we repeat Hartigan's definition:

Definition 4.2.1 *Define the* **level set** $L(\lambda; f)$ *of a density* f *at level* λ *to be a subset of the feature space such that the density for each object in this subset exceeds* λ. $L(\lambda; f) = \{x | f(x) > \lambda\}$.

Based on Hartigan's definition, we further define a continuously contiguous set:

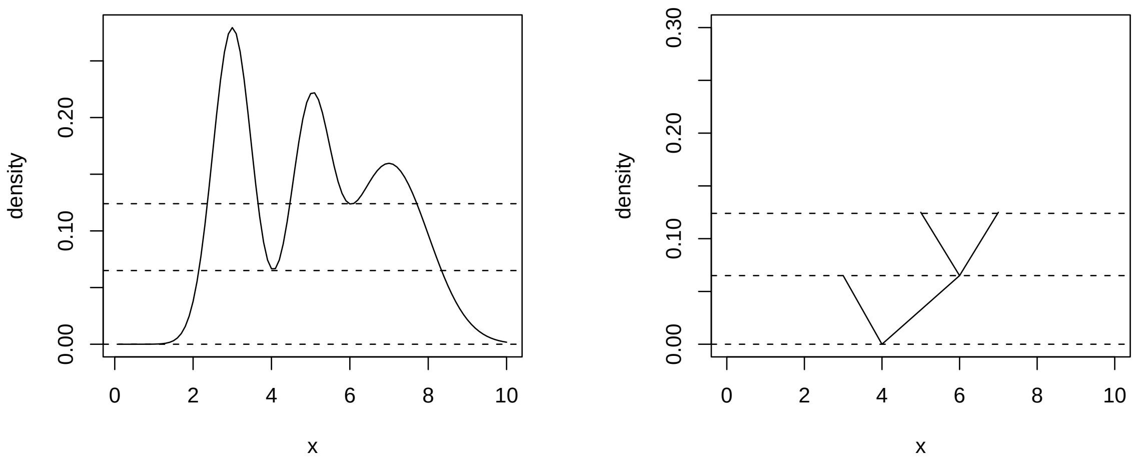
Definition 4.2.2 *Consider any two points* p_1, p_2 *in a set* $S \subset \mathcal{R}^d$ *and denote by* $a_1 = p_1, a_2, \cdots, a_{m-1}, a_m = p_2$ *a sequence of* m *points* $a_1, \cdots, a_m$ *in* S. *If* $\forall \epsilon > 0$, *there exist $m > 0$ and a sequence $a_1, \cdots, a_m$ in S such that $||a_i - a_{i+1}|| < \epsilon$, where $||a_i - a_{i+1}||$ is the Euclidean norm of the vector $a_i - a_{i+1}$, the set S will be called a* **continuously contiguous set***. We further define a set A to be a* **maximally and continuously contiguous subset** *of $D \subset \mathcal{R}^d$ if A is a continuously contiguous subset of D and A is not part of any larger continuously contiguous subset in D.*

A density level set $L(\lambda; f)$ can be a single continuously contiguous subset or a set of disjoint continuously contiguous subsets of the feature space. For example, let X be a random variable with Gaussian density $f(X)$, then for any $0 < \lambda < max_{x \in X} f(x)$, $L(\lambda; f)$ is a single continuously contiguous set; whereas if f is a density of a mixture Gaussian, $L(\lambda; f)$ can be a continuously contiguous set or disjoint continuously contiguous subsets. Hartigan uses this disjointness to set up high-density clusters.

Definition 4.2.3 *If A is a maximally and continuously contiguous subset in $L(\lambda; f)$, then A is a* **high-density cluster** *for the distribution with density f at level λ.*

Since there are only three possibilities for any two high-density clusters A and B found by the same or different density levels, $A \subset B$, $B \subset A$, or $A \cap B = \phi$, it is easy to derive a hierarchical structure such as a cluster tree when constructing the high-density clusters.

For the readers' convenience, we recall the example from Chapter 1. This example shows how a high-density clustering is constructed. Figure 4.1(a) is the p.d.f. of a univariate continuous distribution X. If we set $\lambda_1 = 0.05$, $L(\lambda_1; p) \approx (2, 8.6)$ is a continuously contiguous subset. However, if we set λ_2 to be slightly larger than 0.05, the level set splits into two, forming a union of two disjoint continuously contiguous subsets: $L(\lambda_2; p) \approx (2.2, 3.9) \cup (4.3, 8.1)$. Note that $L(\lambda_2; p) \subset L(\lambda_1; p)$. Similarly, at λ_3 slightly larger than 0.125, another split makes the level set a union of three disjoint continuously contiguous subsets. Figure 4.1(b) gives Hartigan and Stuetzle's cluster tree for this distribution. The root node of the tree is the set $(0, 10)$, which is also the total support of this distribution X. The root has two children: the child



(a) P.d.f. of a continuous distribution (b) Hartigan's cluster tree

Figure 4.1: Hartigan's density level set and cluster tree

on the left is the set $(2.1, 4)$; the child on the right is the set $(4, 8.4)$, and its two children are the sets $(4.3, 6)$ and $(6, 8)$.

Given a continuous density $f(x)$, similarly to Stuetzle's definition [62], we formally define a high-density cluster tree of $f(x)$.

Definition 4.2.4 *First let $D(X) = \{x|f(x) > 0\}$ denote the support of the random variable X with density f. Each node N of the* **high-density cluster tree** *of $f(x)$ is determined by a density level, denoted by $\lambda(N)$, and its content, denoted by $C(N)$ or $Content(N)$. Every node except the root node has a parent node; all nodes except the leaf nodes have at least two child nodes. The tree is built recursively. The root is determined by $\lambda(N_{root}) = 0$, $C(N_{root}) = D(X)$. The child nodes of any node N are constructed as follows: find the lowest $\lambda_c > \lambda(N)$ such that the set $L(\lambda_c; p) \cap C(N)$ has at least two disjoint maximally and continuously contiguous subsets, denoted $C_1, C_2, \cdots, C_k$, then add k child nodes to N, each having density level λ_c and content $C_1, C_2, \cdots, C_k$ respectively. If no such λ_c exists, then N is a leaf node. This tree will* *be denoted by $CT(F)$, where F is the c.d.f. of the distribution with density f.*

Note that to make the above definition work, we need the maximum value of $f(x)$ to be finite. Although the high-density cluster tree, $CT(F)$, does not determine the continuous distribution F, it does capture its inherent modal structure. This characteristic of F, $CT(F)$, represents the true nested clustering of F. It can be regarded as a parameter of F that we would like to learn.

In practice, we observe samples from unknown distributions. Given a sample $X = \{x_1, x_2, \cdots, x_n\}$ drawn from F, the cluster tree built by a hierarchical clustering method such as linkage clustering or runt pruning clustering is an estimate of $CT(F)$. We denote such an estimate as $\widehat{CT}(F)$. A clustering method that produces an estimate of $CT(F)$ is an estimator of $CT(F)$.

Partitioning methods such as k-means or DBSCAN thus produce estimates that have a root and many branches one layer deep from the root. These methods are not originally good estimators of a high-density cluster tree with more layers.

4.3 Properties of high-density cluster tree

Some properties are derived directly from the definition and we call them the nested properties.

Let N_{root} be the root node, N_2 and N_3 two sibling nodes of $CT(F)$, and N_1 their parent node. The high-density cluster tree has the following properties:

1) $\lambda(N_{root}) = 0$

2) $\lambda(N_1) < \lambda(N_2)$

3) $\lambda(N_2) = \lambda(N_3)$

4) $C(N_2) \subset C(N_1)$

5) $C(N_3) \subset C(N_1)$

6) $C(N_2) \cap C(N_3) = \phi$

The first three properties are satisfied by definition. Properties 4) and 5) are obtained because N_2 and N_3 are child nodes of N_1 and their contents are disjoint subsets of N_1's content. Since the contents of N_2 and N_3 are disjoint, property 6) is satisfied.

The one-to-one linear transform of a probability density implies an equivariance property of a high-density cluster tree.

4.3.1 Equivariance

We are interested in the relation of the high-density cluster tree of X and AX, where X is a vector of random variables with density function $f(X)$ and A is a full rank matrix used to define a linear transformation of X. Here AX represents a family of distributions such that each is a linear transformation of any of the others. The following propositions are useful.

Proposition 4.3.1 *A continuously contiguous set is still continuously contiguous under any one-to-one linear transformation.*

Proof: Let S be the original contiguous set in $\mathcal{R}^n$. Define the linear transformation by a full rank transformation matrix A. $\forall p_1, p_2 \in S$, let x_1 be the coordinate of p_1, x_2 the coordinate of p_2, and denote $x_d = x_1 - x_2$ by the linear transformation with transformation matrix A. Let $y_1 = Ax_1$ and $y_2 = Ax_2$ be the coordinates of the transformed points of p_1 and p_2 respectively. We have

$$y_1 - y_2 = Ax_1 - Ax_2 = A(x_1 - x_2) = Ax_d$$

$$||y_1 - y_2|| = ||Ax_d||,$$

where $||Ax_d||$ is the Euclidean norm of the vector Ax_d. Denote

$$x_d = x_1 - x_2 = [d_1, d_2, \cdots, d_n]^T$$

$$A = [\alpha_1, \alpha_2, \cdots, \alpha_n]^T$$

$$\alpha_i = [\alpha_{i_1}, \alpha_{i_2}, ..., \alpha_{i_n}]$$

$$t = max_{\forall i}(\sum_{j=1}^{n} |\alpha_{ij}|)$$

We know $t < \infty$, because $|\alpha_{ij}| < \infty$ and $n < \infty$. Thus,

$$Ax_d = [\alpha_1 x_d, \alpha_2 x_d, ..., \alpha_n x_d]^T$$

$$||Ax_d||^2 = \sum_{i=1}^{n}(\alpha_i x_d)^2 = \sum_{i=1}^{n}(\sum_{j=1}^{n}\alpha_{ij}d_j)^2 \le \sum_{i=1}^{n}(\sum_{j=1}^{n}|\alpha_{ij}||d_j|)^2$$

$$\le \sum_{i=1}^{n}(\sum_{j=1}^{n}|\alpha_{ij}||x_d|)^2 = |x_d|^2\sum_{i=1}^{n}(\sum_{j=1}^{n}|\alpha_{ij}|)^2 \le |x_d|^2 n t^2$$

Denote $b = (n^{1/2}t)$, $b < \infty$. We have

$$||y_1 - y_2|| \le b||x_1 - x_2||.$$

Let S' be the transformed set of S. For any two points p'_1, p'_2 in S', $\forall \epsilon' > 0$, let $\epsilon = \epsilon'/b$. Let p_1, p_2 be the original points in S corresponding to p'_1, p'_2.

Since S is a continuously contiguous set, by definition there is a positive integer $m < \infty$ such that there exists a series of m points in S, $a_1, a_2, \cdots, a_{m-1}, a_m$, for which $||a_i - a_{i+1}|| < \epsilon$, where $p_1 = a_1$, $p_2 = a_m$. By the linear transformation, let a'_i be the transformed point of a_i, then

$$||a'_i - a'_{i+1}|| \le b||a_i - a_{i+1}|| < b\epsilon = \epsilon'.$$

Therefore, S' is a continuously contiguous set by definition. □

Proposition 4.3.2 *Let S'_1, S'_2 be the transformed sets of continuously contiguous sets S_1 and S_2 respectively by the transformation matrix A. If $S_1 \cup S_2$ is not continuously contiguous, $S'_1 \cup S'_2$ is not continuously contiguous either.*

Proof: By proposition 4.3.1, S'_1 and S'_2 are continuously contiguous sets. If $S'_1 \cap S'_2 \neq \phi$, then $S'_1 \cup S'_2$ is a continuously contiguous set, therefore $S_1 \cup S_2$ is a continuously contiguous set as well, but this conflicts with the initial condition that $S_1 \cup S_2$ is not continuously contiguous. Therefore $S'_1 \cap S'_2 = \phi$ and $S'_1 \cup S'_2$ is not continuously contiguous. $\square$

Note that if S_1 and S_2 are continuous but $S_1 \cup S_2$ is not, then $S_1 \cap S_2 = \phi$. This is true for their transformed sets as well.

Proposition 4.3.3 *The set generated by a linear transformation of a density level set $L(\lambda; f(\mathbf{x}))$ is the density level set determined by $L(|A|^{-1}\lambda; g(\mathbf{y}))$, where $\mathbf{y} = A\mathbf{x}$, A is the invertible linear transformation matrix, and $g(\cdot)$ is the density function of $\mathbf{y}$.*

Proof: By the one-to-one transform of a density, we have

$$g(\mathbf{y}) = g(A\mathbf{x}) = f(A^{-1}\mathbf{y})|J|$$

where J is the Jacobian matrix with determinant $|J| = |A|^{-1}$. Then

$$L(\lambda; f(\mathbf{x})) = \{\mathbf{x}|f(\mathbf{x}) > \lambda\}$$

and for this set, we have

$$g(\mathbf{y}) = g(A\mathbf{x}) = f(A^{-1}\mathbf{y})|A|^{-1} = f(\mathbf{x})|A|^{-1} > \lambda|A|^{-1}$$

Now we have a new density level set for Y which is $L(|A|^{-1}\lambda; g(\mathbf{y}))$. $\square$

Let $T(X)$ and $T(Y)$ be the high-density cluster trees of the original and the transformed distribution respectively. From the above propositions, we assert that for every node N in $T(X)$ there is a unique node N' in $T(Y)$ with $C(N') = AC(N)$ and $\lambda(N') = |A|^{-1}\lambda(N)$, where A is the transformation matrix. Therefore we say that the high-density cluster tree of a continuous distribution has the equivariance property under any linear transformation determined by a full rank transformation matrix. Note that it is not equivariant to non-linear transformations.

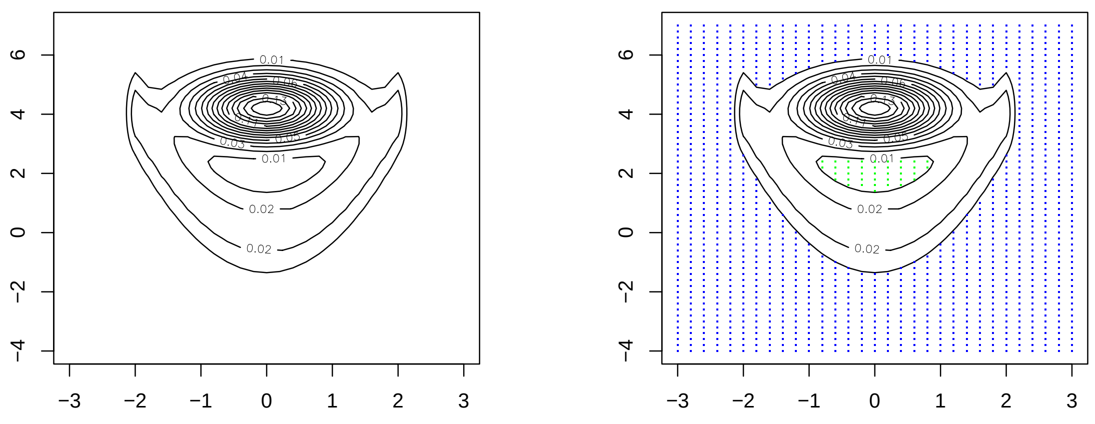
4.3.2 Low-density cluster tree

The high-density cluster tree focuses on finding clusters in higher density regions, and lower density regions are missed.

Let $L(\lambda; f(x))$ be a density level set and $D(X) - L(\lambda; f(x))$ the set of all the points with density no more than λ. We call this set the low-density level set and denote it by $L_{low}(\lambda; f(x))$, then $L(\lambda; f(x))$ and $L_{low}(\lambda; f(x))$ are complementary sets.

We can build the tree for low-density regions in a similar way to the high-density cluster tree. The difference is that in a low-density cluster tree, the density level of the root is $max_{\forall x}(f(x))$, and when searching for child nodes, instead of increasing the density level we decrease the density level.

Two different distributions can have the same high-density cluster tree structure yet a different low-density tree structure. Figure 4.2(a) shows a density contour of a



(a) A contour with low-density regions (b) Illustration of low-density regions

Figure 4.2: An example of low-density regions

distribution F_1 with two low-density regions. The first is located in the centre of the contour; the second is located in the border of the contour. The two dotted areas shown in Fig. 4.2(b) illustrate these low-density regions.

The high-density cluster tree of F_1 has only a root. The low-density cluster tree has a root with two child nodes. If we have a distribution F_2 with a density contour similar to that of F_1 but without the low-density region in the centre, both the high-density and the low-density cluster tree will have just a root. This happens because a lower density hole exists in the contour of the density of F_1 but not in F_2. The low-density cluster tree can disclose such information but the high-density cluster tree does not. Therefore, in certain circumstances, we are interested in both trees to get more complete information.

There is interest in low-density regions. For example, dense groups of stars are called "clusters" in astronomy. Astronomers are also interested in large regions with few stars especially when they are hidden inside a dense area. Some algorithms that discover interesting holes in the data are proposed in the literature [46].

In this chapter we have defined $CT(F)$ with F the c.d.f. of a continuous distribution. We further regard $CT(F)$ as a parameter of F. An immediate question is how to define a similar cluster tree for a discrete distribution.

4.4 "Contiguity" in a discrete distribution

In a discrete distribution points in its support can have non-zero probability mass and the contiguity property determined by continuity no longer holds. It is not obvious how to determine a connected subset in a discrete distribution. In Section 1.1.2 we gave examples of the difficulties in determining "contiguity" for a discrete distribution.

Perhaps density-based clustering methods such as single linkage can be applied to a discrete distribution. However, this idea may not work. Consider a discrete distribution with all the data points located on the vertices of a two-dimensional grid. The distances of any two points to their closest neighbours are identical. The minimal spanning tree and the KNN graph do not have a unique solution. Therefore the density-based clustering methods discussed previously can not be used for this discrete distribution. If we directly apply these methods, we ignore the information on the probability mass of each point.

Similarly to Hartigan's density level set, we now define a mass level set for a discrete distribution. Given a multivariate discrete random variable X in $\mathcal{R}^d$, the probability of $X = \mathbf{x}$, i.e. the probability mass of $\mathbf{X}$ at $\mathbf{x}$, will be denoted $m(\mathbf{x})$. The support of $\mathbf{X}$ is denoted $D_{\mathbf{X}} = \{\mathbf{x} | m(\mathbf{x}) > 0\}$.

Definition 4.4.1 *Define the* **mass level set** $L(\lambda; m)$ *of a discrete distribution at level* λ *to be a subset of* $D_{\mathbf{X}}$ *such that the mass for each object in this subset is not less than* λ. $L(\lambda; m) = \{\mathbf{x} | \mathbf{x} \in D_{\mathbf{X}}, m(\mathbf{x}) \geq \lambda\}$.

Given any λ and a discrete distribution with the mass function $m(\cdot)$, to determine the connected subsets in $L(\lambda; m)$ we have to find a way to define the "contiguity" of any two points in $L(\lambda; m)$.

4.5 Coupling graph method

We say that two points are contiguous at λ only if they are in $L(\lambda; m)$. Before we define the contiguity of two discrete points, we can identify the following two desirable properties that any definition of contiguity should possess.

1. Contiguity is symmetric. If point a is contiguous to point b at λ, then b is contiguous to a at λ.

2. If two points are contiguous at λ_2, they must be contiguous at λ_1 if $\lambda_1 < \lambda_2$.

In addition to the above properties, it is reasonable that whether or not two points in $L(\lambda; m)$ are contiguous is influenced by the distance between them and the distances to other points especially the points outside $L(\lambda; m)$.

As described in Chapter 1, Hartigan proposed a way to determine contiguity. However, his approach does not work for arbitrary discrete distributions. We need a way to determine contiguity for arbitrary distributions. We first propose the four-spring model discussed in Appendix D. The four-spring model is natural but it is computationally expensive and suitable only for two-dimensional cases. For a more general and efficient approach, we develop a coupling graph method. The general idea of this method is that for any point $\mathbf{p}$ in $L(\lambda; m)$ its contiguous points are determined by its nearest neighbour among those points not in $L(\lambda; m)$.

Consider any directed or mixed graph G, where undirected edges are interpreted as two directed edges, one in each direction. We are interested in two undirected graphs derived from such a graph. We introduce the following definitions.

Definition 4.5.1 *For any such graph G, the graph that retains only the undirected edges of G will be called the* **strong coupling graph** *of G.*

Definition 4.5.2 *For any such graph G, the graph that replaces all edges, directed or undirected, in G by undirected edges will be called the* **weak coupling graph** *of G.*

Suppose we have a set D of n points, $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$, and a measure of distance $\delta(\mathbf{x}_i, \mathbf{x}_j)$ between points $\mathbf{x}_i$ and $\mathbf{x}_j$. Associated with each point $\mathbf{x}_i$ is a value of a mass weight function $m(\mathbf{x}_i) \in \mathbb{R}^+$. This information will be denoted by the triple $< D, \delta, m >$.

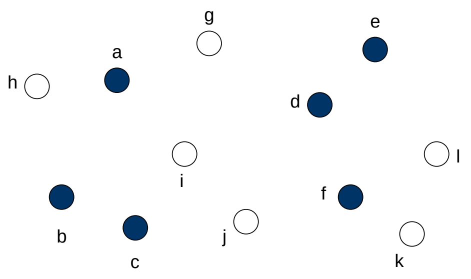
Definition 4.5.3 *Define a subset of $S \subseteq D$ to be the λ* **level set***, denoted by $L_{\lambda,S}$, if $L_{\lambda,S} = \{\mathbf{x} | \mathbf{x} \in S, m(\mathbf{x}) \geq \lambda\}$. For convenience, when $S = D$, we will write $L_{\lambda,D}$ as L_λ.*

Note that, in the above definition the mass function $m(x)$ need not in general be restricted to be a probability. Here we take $m(x)$ to be any bounded real-valued function.

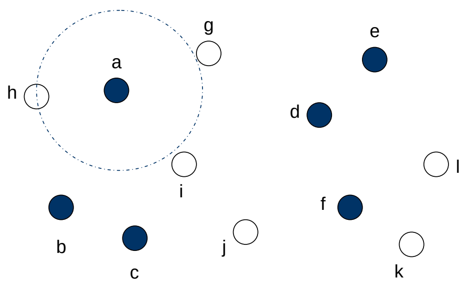
Now consider the triple $< D, \delta, m >$ and level λ. For each point $\mathbf{x_i}$ in the λ level set we draw a "hyper-sphere" with radius being the distance from $\mathbf{x_i}$ to the closest point not in the λ level set. This is defined formally as follows.

Definition 4.5.4 *Given a triple $< D, \delta, m >$, we take a set of points $V \subseteq D$ and a subset $L_{\lambda,V}$ of V as a vertex set, such that $L_{\lambda,V}$ is a λ level set of V. Draw a directed edge from $\mathbf{x}$ to $\mathbf{y}$ if $\mathbf{x}, \mathbf{y} \in L_{\lambda,V}$ and $\delta(\mathbf{x}, \mathbf{y}) \leq \delta(\mathbf{x}, \mathbf{z}), \forall \mathbf{z} \in V - L_{\lambda,V}$. We call the resulting graph the λ* **similarity coupling graph** *of $L_{\lambda,V}$, and denote this by $G_{\lambda,sim}(V)$.*

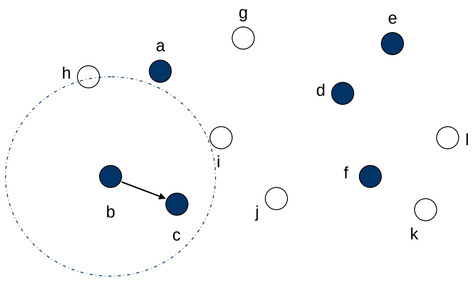
Figure 4.3 shows an example of how a λ similarity coupling graph is constructed. In Fig. 4.3 (a), each point represented by a solid circle is inside a λ level set in $\mathcal{R}^2$, each point represented by a hollow circle is outside the level set. For point a, point h is its nearest neighbour among the points outside the level set. A neighbourhood of point a can be constructed by a circle centred at a with radius the distance from a to h. This neighbourhood of a is shown in Fig. 4.3 (b). Since there is no other point in the level set locating inside the neighbourhood of a, by Definition 4.5.4, no edge can be drawn to connect a to other points. The similar neighbourhood of b is shown in Fig. 4.3 (c). Since point c is in the neighbourhood of b, by Definition 4.5.4, we draw a directed edge from b to c. Similarly in Fig. 4.3 (d), b is in the neighbourhood of c, we can also draw a directed edge from c to b. Therefore, we use an undirected edge to connect b and c. Figure 4.3 (e) shows the λ similarity coupling graph for this λ level set.



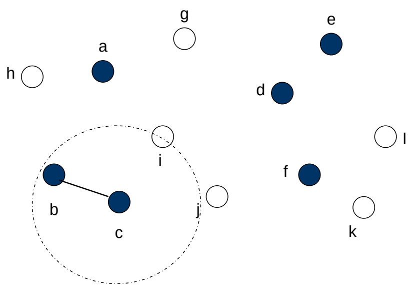
(a) λ level set



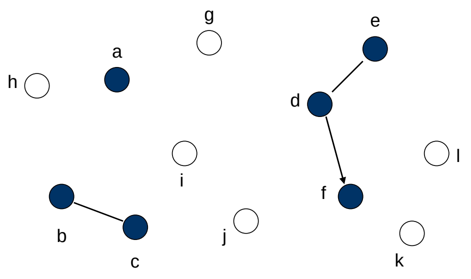
(b) Point a



(c) Point b



(d) Point c



(e) λ similarity coupling graph

Figure 4.3: Example of a λ similarity coupling graph

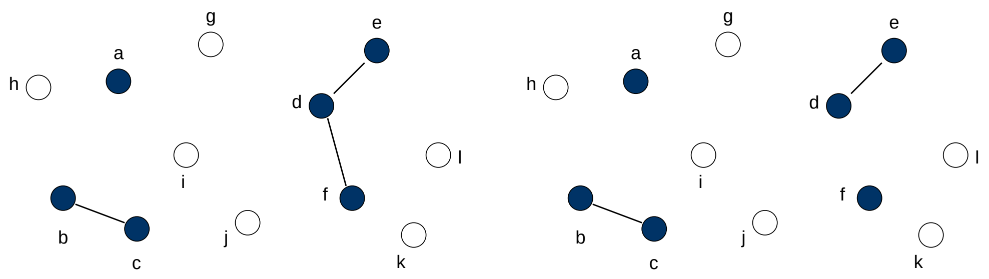
The following two definitions show the ways to obtain an undirected graph from a λ similarity coupling graph.

Definition 4.5.5 *The weak coupling graph of a λ similarity coupling graph $G_{\lambda,sim}(V)$ is called a λ* **weak coupling graph***. We denote this graph $G_{\lambda,weak}(V)$.*

Definition 4.5.6 *The strong coupling graph of a λ similarity coupling graph $G_{\lambda,sim}(V)$ is called a λ* **strong coupling graph***. We denote this graph $G_{\lambda,strong}(V)$.*

When the context makes it clear, we will refer to a λ weak coupling graph or a λ strong coupling graph as a λ-coupling graph and denote this generically as $G_\lambda(V)$.

Figure 4.4 shows the λ weak and the λ strong coupling graphs constructed from



(a) A λ weak coupling graph (b) A λ strong coupling graph

Figure 4.4: Example of λ-coupling graphs

the λ similarity coupling graph shown in Fig. 4.3 (e).

The following two definitions show the ways to construct a maximally connected subset of a λ-coupling graph.

Definition 4.5.7 *A λ* **weak component** *of a set V is a connected component in $G_{\lambda,weak}(V)$. We denote this by $c_{\lambda,i}(G_{\lambda,weak}(V))$, $i = 1, 2, ..., k$, where k is the number of such components.*

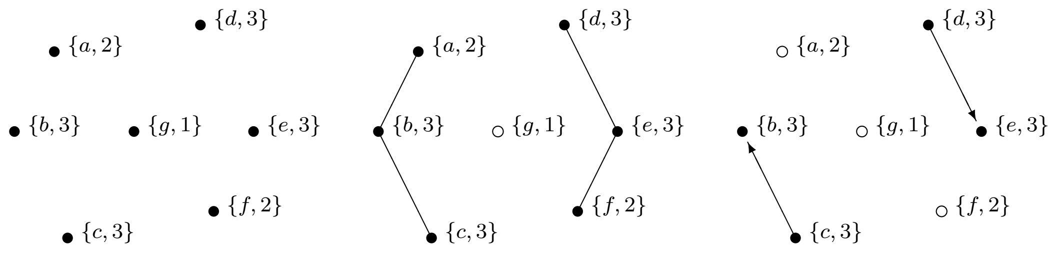
Definition 4.5.8 *A λ* **strong component** *of a set V is a connected component in $G_{\lambda,strong}(V)$. We denote this by $c_{\lambda,i}(G_{\lambda,strong}(V))$, $i = 1, 2, ..., k$, where k is the number of such components.*

Since either a weak coupling graph or a strong coupling graph is an undirected graph, any connected component in such a graph is a strongly connected component as well.

When the context makes it clear, we will refer to a λ weak component or a λ strong component as a λ component and denote this generically as $c_{\lambda,i}(G_\lambda(V))$.

To obtain a connected component and for the convenience of further discussion, we recall a few standard definitions from graph theory such as a path, a connected graph, and a component. These definitions are given in Appendix C.

We show an example to construct similarity coupling graphs at different levels. Consider a triple $< D, \delta, m >$, where $D = \{a, b, c, d, e, f, g\}$ is shown in Fig. 4.5(a), δ is the Euclidean distance between points in D, and the mass weight function m is



(a) a weighted data set (b) $\lambda = 2$ (c) $\lambda = 3$

Figure 4.5: Example of a triple and its similarity coupling graphs.

defined as follows: $m(a) = 2$, $m(b) = 3$, $m(c) = 3$, $m(d) = 3$, $m(e) = 3$, $m(f) = 2$, $m(g) = 1$. The values from m are shown in Fig. 4.5(a). Figures 4.5(b) and (c) show the similarity coupling graphs of this triple at level 2 and 3 respectively. The similarity coupling graph at $\lambda = 1$ is just the complete graph with vertex set D.

Based on the above definitions, we have several propositions that give properties of our coupling graphs. These are the basic building blocks of a tree.

Proposition 4.5.1 *The λ weak coupling graph $G_{\lambda,weak}(V)$ of the triple $< D, \delta, m >$ is unique.*

Proof: Suppose there exist two λ weak coupling graphs of the triple represented by their edge sets E_1 and E_2. Their vertex sets are both $V \subseteq D$, and they are derived at the same level λ. Let $e \in E_1$ and x,y be the elements of V connected by e. Then there exists either a directed edge from x to y or from y to x, and therefore $\forall z \in D - L_{\lambda,V}$, either $\delta(x,y) \leq \delta(x,z)$ or $\delta(x,y) \leq \delta(y,z)$. But if either holds, then $e \in E_2$ for E_2 to be the edge set of a λ weak coupling graph. Similarly, if $e \in E_2$ then $e \in E_1$. Therefore, $E_1 = E_2$ and the λ weak coupling graph is unique. $\square$

Proposition 4.5.2 *The λ strong coupling graph $G_{\lambda,strong}(V)$ of the triple $< D, \delta, m >$ is unique.*

Proof: Analogous to proof of Proposition 4.5.1. $\square$

Proposition 4.5.3 *A higher λ level set is a subset of a lower λ level set.*

Proof: Suppose $\lambda_1 < \lambda_2$, then by definition $L_{\lambda_2} = \{\mathbf{x} : m(\mathbf{x}) \geq \lambda_2, \mathbf{x} \in D\}$. Now $m(\mathbf{x}) \geq \lambda_2 > \lambda_1$, which implies that if $\mathbf{x} \in L_{\lambda_2}$ then $\mathbf{x} \in L_{\lambda_1}$. So $L_{\lambda_2} \subseteq L_{\lambda_1}$. $\square$

Proposition 4.5.4 *Suppose $0 \leq \lambda_1 < \lambda_2$ and let $G_{\lambda_i,weak}(V)$ be a λ_i weak coupling graph of a triple $< D, \delta, m >$ with its set of edges denoted by E_i for $i = 1, 2$. Then $E_2 \subseteq E_1$.*

Proof: By definition, every edge $e \in E_2$ connects two vertices $\mathbf{x}$ and $\mathbf{y}$ if and only if either $\delta(x,y) \leq \delta(x,z)$ or $\delta(x,y) \leq \delta(y,z)$ $\forall z \in D - L_{\lambda_2}$. Suppose that the first of these holds. By Proposition 4.5.3, $L_{\lambda_2} \subseteq L_{\lambda_1}$, so $D - L_{\lambda_1} \subseteq D - L_{\lambda_2}$. Thus, if $\delta(x,y) \leq \delta(x,z)$ $\forall z \in D - L_{\lambda_2}$, then $\delta(x,y) \leq \delta(x,z)$ $\forall z \in D - L_{\lambda_1}$, a subset of $D - L_{\lambda_2}$. Therefore, $e \in E_1$. Similarly, if the second of these holds, $e \in E_1$. So $E_2 \subseteq E_1$. $\square$

Proposition 4.5.5 *Suppose $0 \leq \lambda_1 < \lambda_2$ and let $G_{\lambda_i,strong}(V)$ be a λ_i strong coupling graph of a triple $< D, \delta, m >$ with its set of edges denoted by E_i for $i = 1, 2$. Then $E_2 \subseteq E_1$.*

Proof: Essentially the same as the proof of Proposition 4.5.4, following from Proposition 4.5.3. □

From Propositions 4.5.1 to 4.5.5 we conclude that $G_\lambda(V)$ is unique for each λ and $G_{\lambda_2}(V) \subseteq G_{\lambda_1}(V)$ whenever $\lambda_1 < \lambda_2$. For every λ, $G_\lambda(V)$ (weak or strong) provides either a connected graph or a graph with two or more connected components. For $\lambda_1 < \lambda_2$ either $G_{\lambda_1}(V) = G_{\lambda_2}(V)$ or $G_{\lambda_2}(V)$ is a proper subgraph of $G_{\lambda_1}(V)$ in that the edge set E_2 of $G_{\lambda_2}(V)$ is a proper subset of the edge set E_1 of $G_{\lambda_1}(V)$. This property gives our coupling graphs a nice hierarchical feature when increasing λ.

In Chapter 1, we mentioned Hartigan's approach for discrete distributions [32]. We find that his approach is a special case of our coupling graph method. We prove this in Theorem 4.10.1 of Section 4.10. Theorem 4.10.1 implies that Hartigan's approach is identical to our coupling graph method if the support of the distribution is the vertex set of a regular square mesh lattice. However, our coupling graph approach is more general and not limited to the support being a regular lattice. Theorem 4.10.1 also implies that the vertex sets of the components from both the weak and the strong coupling graphs at λ are identical to those from the graph constructed by Hartigan's approach at the same level. Theorem 4.10.2 of Section 4.10 addresses this result more clearly.

We use the coupling graph method to determine the contiguity among points of a discrete distribution. To be more specific, for a given level λ, two points from the support of a discrete distribution are contiguous at λ if there is an edge connecting them in the λ-coupling graph constructed for the distribution. The next section shows how to define a cluster tree for a discrete distribution using the coupling graph method.

4.6 High-probability mass cluster tree

In the previous section, we propose a coupling graph to determine if two points from the support of a discrete distribution are contiguous or not at some level λ.

To determine a maximally connected subset in a mass level set, we define a contiguity graph as follows.

Definition 4.6.1 A contiguity graph *at level λ of a discrete distribution with mass function $m(\cdot)$ is constructed by connecting each pair of points in $L(\lambda; m)$ with an undirected edge if they are contiguous at λ.*

A contiguity graph is an undirected graph because "contiguity" is symmetric. Note that a λ-coupling graph is also a contiguity graph.

Similarly to the high-density clusters, for a discrete distribution, we define a high-mass cluster.

Definition 4.6.2 A high-mass cluster *at λ for a discrete distribution is a connected component in the contiguity graph at level λ for this distribution.*

Points **a** and **b** are contiguous at level λ if and only if there exists an edge connecting them in the contiguity graph at λ. In this case we say that **a** and **b** are directly connected at λ as well. If there exists a path in the contiguity graph at λ between points **a** and **b** but no edge connects them directly, we say that **a** and **b** are indirectly connected at λ. We say **a** and **b** are connected at λ if they are either directly or indirectly connected.

Similarly to the high-density cluster tree, we define the high-probability mass cluster tree of a mass function $m(X)$ by simply replacing probability density level by probability mass level. Note that the meaning of *connected set* in these two definitions is also different.

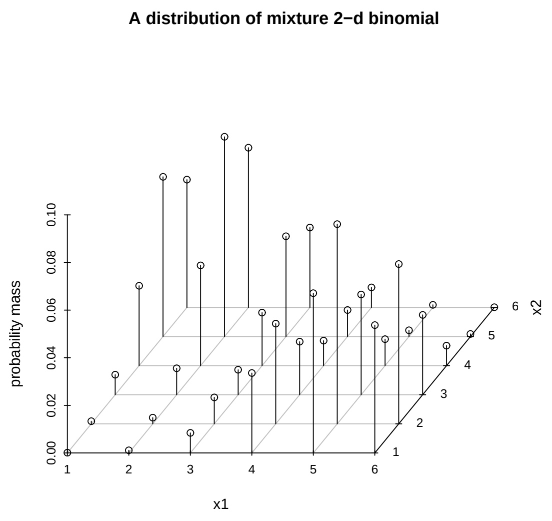
Definition 4.6.3 *Each node N of the* **high-probability mass cluster tree** *of $m(x)$ is determined by a probability mass level, denoted $\lambda(N)$, and its content, denoted $C(N)$ or $Content(N)$. Every node except the root node has a parent node; all nodes except the leaf nodes have at least two child nodes. The tree is built recursively. The root is determined by $\lambda(N_{root}) = 0$, $C(N_{root}) = D_X$. The child nodes of any node N are constructed as follows: find the lowest $\lambda_c > \lambda(N)$ such that the set $L(\lambda_c; m) \cap C(N)$ has at least two disjoint maximally connected subsets, denoted $C_1, C_2, \cdots, C_k$, then add K child nodes to N each with probability mass level λ_c and content $C_1, C_2, \cdots, C_k$ respectively. If there is no such λ_c, the node N is a leaf node.*

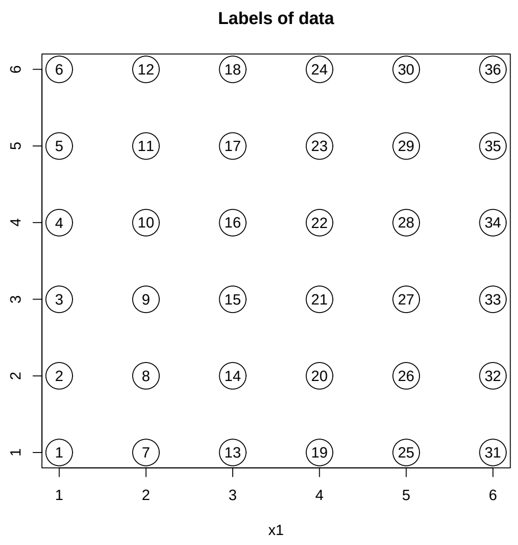
Note that the disjoint maximally connected subsets in the definition are associated with components in the contiguity graph of $m(x)$ at level λ_c respectively. If we use the coupling graph method to construct the contiguity graph, then since we can construct both the weak and strong λ components, we can define two high-probability mass cluster trees for a discrete distribution. We generically call them high-probability mass cluster trees constructed by the coupling graph method. If the support of the distribution is a vertex set of a regular lattice, then by Theorem 4.10.2, the high-probability mass cluster tree constructed by the coupling graph method is unique.

4.7 Examples

We now generate two discrete distributions and build high-probability mass cluster trees of them.

Figure 4.6(a) shows an artificial mixture of two 2-d binomials with the labels of all



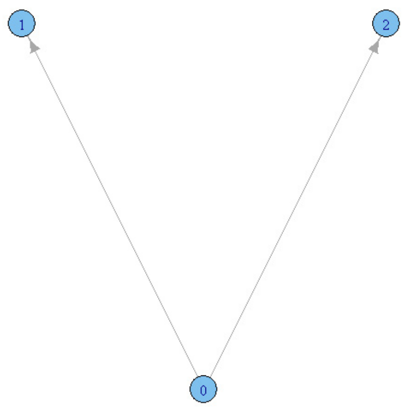


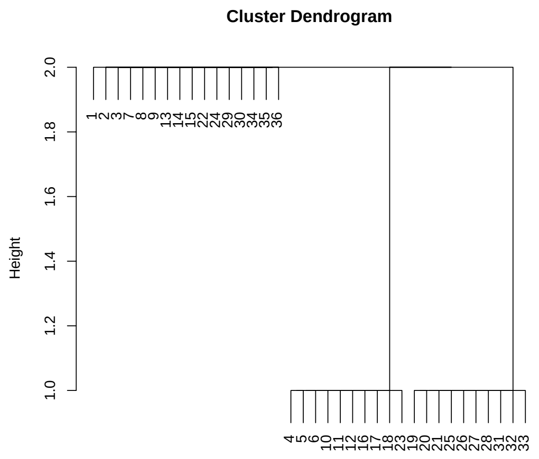
(a) 2-d discrete distribution (b) Labels of positive mass points

Figure 4.6: A distribution of a mixture of two 2-d binomials

the points with positive mass displayed in Fig. 4.6(b). The labels are used to identify these points when they are assigned to different nodes in a cluster tree. Intuitively speaking, in this distribution, there are two modes with peaks at locations $(2, 5)$ and $(5, 2)$ respectively separated by a region with lower probability mass points.

We build a high-probability mass cluster tree for the distribution in Fig. 4.6 using our coupling graph method. The structure of the tree is shown in Fig. 4.7(a). Figure 4.7(b) shows a dendrogram-like plot of the tree. In this plot, the top layer with height two represents the root node that contains all the points with positive mass. The labels of points with lower probability mass levels are shown in the top layer meaning that they are not assigned to the two modes represented by the two





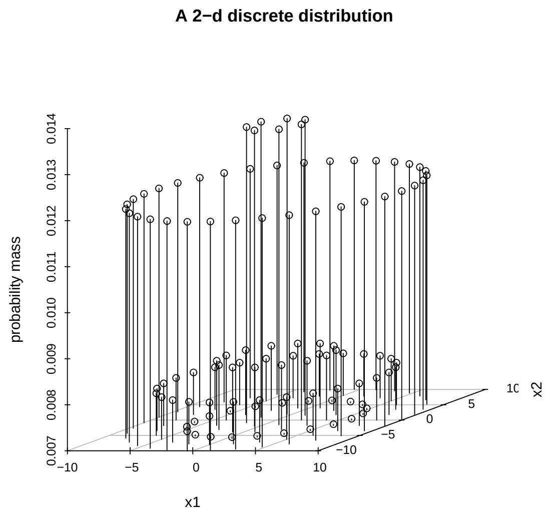
(a) Cluster tree structure (b) Dendrogram-like tree

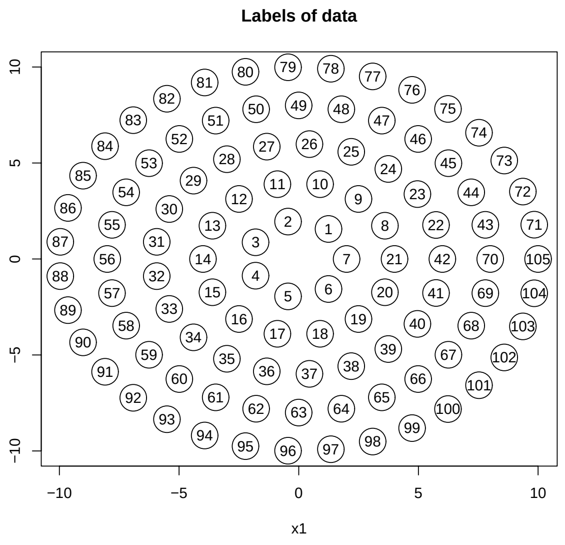
Figure 4.7: Example of a high-probability mass cluster tree

child nodes of the root in the bottom layer with height one. This dendrogram-like plot is different from the usual dendrogram where any point in a non-leaf node would be forced (typically) to one of its child nodes. In Fig. 4.7(b), some points in the root are not assigned to either child node, because these points have a relatively lower probability mass and are outside the level set determined by the probability mass level of the two child nodes.

Figure 4.8(a) shows another artificial discrete distribution. The positive mass points in this distribution form five circles. The probability masses of the points in the inner and outer circles are higher than those of the points in the middle. Intuitively, there are two modes with two circles in each, separated by the circle in the middle. For this distribution, the coupling graph method chooses the first two inner circles and the first two outer circles as two clusters. In other words, the middle circle separates the other four circles.

The next section gives a detailed algorithm for building a high-probability mass





(a) 2-d discrete distribution (b) Labels of positive mass points

Figure 4.8: An artificial discrete distribution

cluster tree.

4.8 An algorithm

To construct the high-probability mass cluster tree, we have to find:

- the connectivity between two points at a probability mass level λ,
- the maximally connected subsets of $L(\lambda; m)$.

We find the connectivity via the coupling graph method. For the subsets, we find connected components in the graph constructed.

We give a general algorithm for constructing a high-probability mass cluster tree. The data structure of each node N includes a content set $Content(N)$ that is the set of all the points assigned to this node and a corresponding probability mass level

$\lambda(N)$. By definition, for the root node, we have $Content(root) = D$, where D is the set of all points in the distribution, and $\lambda(root) = 0$.

After building the set S_λ containing all the different probability mass levels in the distribution, we build the tree recursively using Algorithm 1. Algorithm 2 is invoked

Algorithm 1 Algorithm to build a high-probability mass cluster tree

```
Require: A node $N$ in the tree.
 1: Find Content(N) and set $\lambda_{current} = \lambda(N)$.
 2: Find the next probability mass level: $\lambda_{next} = min_{\forall \lambda_i \in S_\lambda}(\lambda_i | \lambda_i > \lambda_{current})$.
 3: Set $childNode = 0$.
 4: while $\lambda_{next} \neq \phi$ and $childNode = 0$ do
 5:     With $\lambda_{next}$ call Algorithm 2 to construct the set of maximally connected subsets
        in Content(N) $\cap L(\lambda_{next}, m)$, denoting this set $maxSubsets$.
 6:     if at least two components are found in $maxSubsets$ then
 7:         Let $c$ be the total number of components in $maxSubsets$.
 8:         For each component $C_i$, construct a new node $N_i$, where $i = 1, 2, \cdots, c$, add
            $N_i$ to the tree as the child node of $N$ and recursively run Algorithm 1 on
            node $N_i$ to build a subtree from $N_i$.
 9:         Set $childNode = 1$.
10:     else
11:         Set $\lambda_{current} = \lambda_{next}$ and find the new $\lambda_{next}$.
12:     end if
13: end while
14: Return the tree whose root is $N$.
```

from Algorithm 1, and the detailed algorithm is as follows. (Note: we demonstrate this algorithm using the coupling graph method to determine contiguity; if another method such as the four-spring model is used, relevant changes are necessary).

Algorithm 2 Algorithm to obtain maximally connected subsets for the content of a node

```
Require: The content of a node, denoted $nodeD$; A mass level, denoted $\lambda$.
 1: Set up a two-dimensional set $maxSubsets$ to store all the maximally connected
    subsets found in the algorithm.
 2: Set up a 1-dimensional set $currentMaxSubset$.
 3: Set up a set of seeds $seedSet$ for storing seeds, used to find a maximally connected
    subset.
 4: while not all the points in the set $S_D = nodeD \cap L(\lambda, m)$ have been processed
    do
 5:   Initialize $seedSet$ to be empty.
 6:   Take a point say $\mathbf{p}$ from $S_D$ that is not processed and add $\mathbf{p}$ to $seedSet$;
      initialize the set $currentMaxSubset$ to be empty; set the status of $\mathbf{p}$ to be
      processed.
 7:   while $seedSet$ is not empty do
 8:     Remove the first point say $\mathbf{p}$ from $seedSet$, add $\mathbf{p}$ to $currentMaxSubset$.
 9:     For each point say $\mathbf{q}$ in the set $S_D$, if $\mathbf{q}$ is not processed and there exists
        an edge connecting $\mathbf{p}$ and $\mathbf{q}$ in the $\lambda$-coupling graph $G_\lambda(nodeD)$, add $\mathbf{q}$ to
        $seedSet$ and set the status of $\mathbf{q}$ to be processed.
10:   end while
11:   Store $currentMaxSubset$ in $maxSubsets$.
12: end while
13: Return $maxSubsets$.
```

The computation complexity of Algorithm 1 is $O(nlogn)$. The algorithm builds a high-probability mass cluster tree that is a parameter for a discrete distribution. We denote this parameter by $CT(F)$ as well but note that this F is the c.d.f. of a discrete distribution. Since different methods can be applied to determine contiguity among points, this parameter is not unique; we write $CT_i(F)$ to indicate that this parameter comes from the i^{th} method. This is a major difference from the high-density cluster tree for a continuous distribution.

If we are given a sample from a discrete distribution, applying the above algorithm to that sample using the i^{th} method to determine contiguity gives us an estimator of $CT_i(F)$.

4.9 A convergence discussion

We explore the convergence properties of a high-density/mass cluster tree.

4.9.1 Some general results

As described early in this chapter, a high-density cluster tree is uniquely determined by a density function of a continuous distribution; a high-probability mass cluster tree can be uniquely constructed from a discrete distribution by our coupling graph method if its support is located on a regular lattice.

Given a sequence of random variables $\{X_n\}$ and a random variable X, let F_n be the c.d.f. of X_n and F be the c.d.f. of X. Suppose $lim_{n\to\infty}F_n(x) = F(x)$ for all x. Let $T(F_n)$ and $T(F)$ be the corresponding high-density/mass cluster tree on F_n and F respectively. Let $dist(T_1, T_2)$ be any measure of distance between two trees T_1 and T_2. To understand the property of convergence between a sequence of trees $\{T(F_n)\}$ and a tree $T(F)$, we define that $\{T_n\}$ converges to T with respect to a tree distance measure $dist(\cdot)$ if $lim_{n\to\infty}dist(T_n, T) = 0$.

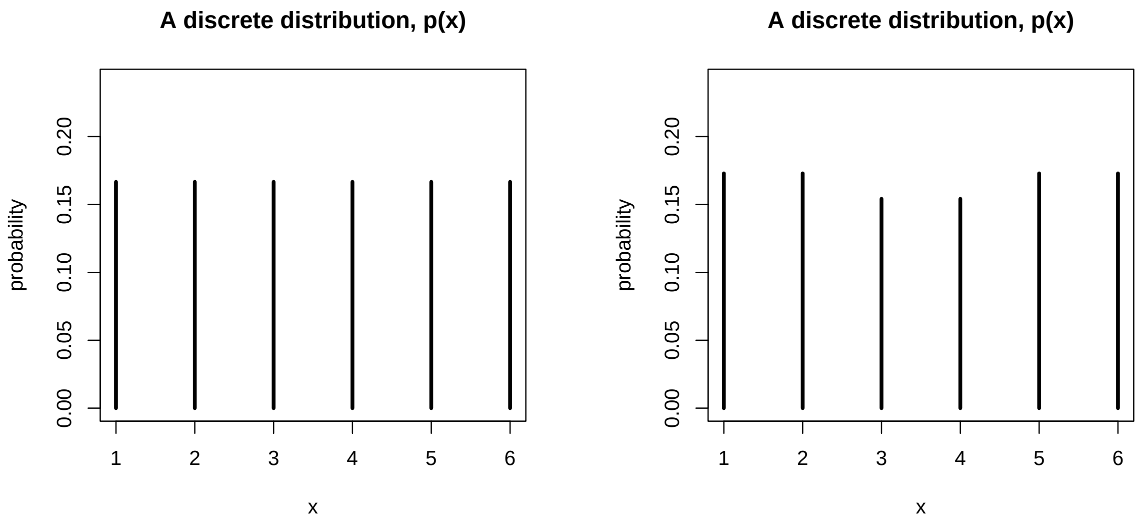
We are interested in whether or not $\{T(F_n)\}$ converges to $T(F)$ with respect to our tree distance measure. As described in Section 2.5.3, our measure of distance between two trees T_1 and T_2 is denoted $d(T_1, T_2)$. Since this measure calculates the distance between two trees whose root content is finite and identical, we consider the case that $\{X_n\}$ and X have the same finite support. Note that $\{T(F_n)\}$ converging to $T(F)$ with respect to our tree distance measure $d(\cdot)$ does not imply that $lim_{n\to\infty}F_n(x) = F(x)$ for all x. We are interested in the reverse. We formally demonstrate that in the following proposition.

Proposition 4.9.1 *Given that $\{X_n\}$ and X have the same finite support, $lim_{n\to\infty}F_n(x) =$*

$F(x)$ for all x does not imply that $\{T(F_n)\}$ converges to $T(F)$ with respect to the tree distance measure $d(\cdot)$ defined in Section 2.5.3.

Proof: We prove Proposition 4.9.1 by a counter-example as follows.

Figure 4.9(a) shows a discrete distribution with c.d.f. F and probability mass function $P(X = i) = 1/6$, for $i = 1, 2, \cdots, 6$. Figure 4.9(b) shows a discrete distri-



(a) Probability mass of F (b) Probability mass of F_n

Figure 4.9: The probability mass functions of two distributions

bution with c.d.f. F_n and probability mass function

$$P(X_n = i) = \begin{cases} 1/6 + 1/(2n) & \text{if } i = 1, 2, 5, 6 \\ 1/6 - 1/n & \text{if } i = 3, 4. \end{cases}$$

Clearly $lim_{n\to\infty} F_n(x) = F(x)$ for all x. However the high-probability mass cluster trees for F_n and F as constructed by our coupling graph method are different no matter what the value of $n < \infty$. $T(F)$ is a tree with only a root whereas for any $n < \infty$ $T(F_n)$ is a tree with a root and two child nodes such that one child node contains points 1 and 2 and the other contains points 5 and 6. By our tree distance measure, $d(T(F_n), T(F)) = 1$, which is a constant for all $n < \infty$. Therefore, for any $\epsilon > 0$, we cannot find an integer N such that when $n > N$, we have $d(T(F_n), T(F)) < \epsilon$. $T(F_n)$ does not converge to $T(F)$ in terms of our tree distance measure. $\square$

Actually, in the above example, the tree structure of $T(F_n)$ remains the same and never becomes similar to the tree structure of $T(F)$ as n increases. The content in each node of $T(F_n)$ also remains unchanged as n increases. Therefore, our distance measure $d(T(F_n), T(F))$ cannot get smaller as n increases.

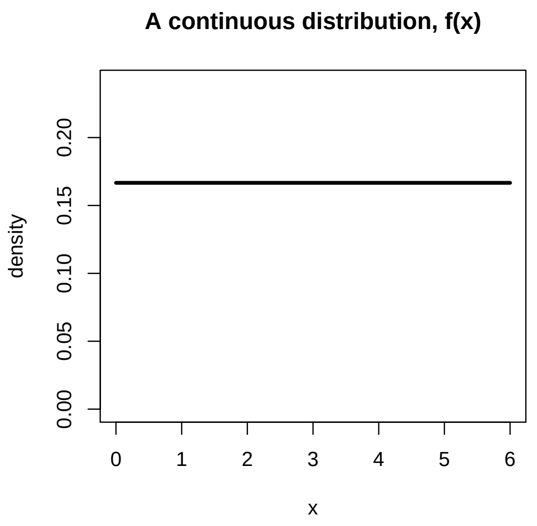
Since our distance measure cannot calculate the distance between two trees with infinite points in their nodes, we cannot study the convergence of $\{T(F_n)\}$ to $T(F)$ by $d(T(F_n), T(F))$ if either F_n or F is a c.d.f. for a continuous distribution. However, we have the following proposition.

Proposition 4.9.2 *Let both F_n and F be c.d.f.s for continuous distributions. Then $lim_{n\to\infty}$ $F_n(x) = F(x)$ for all x does not imply that $\{T(F_n)\}$ and $T(F)$ have the same tree structure.*

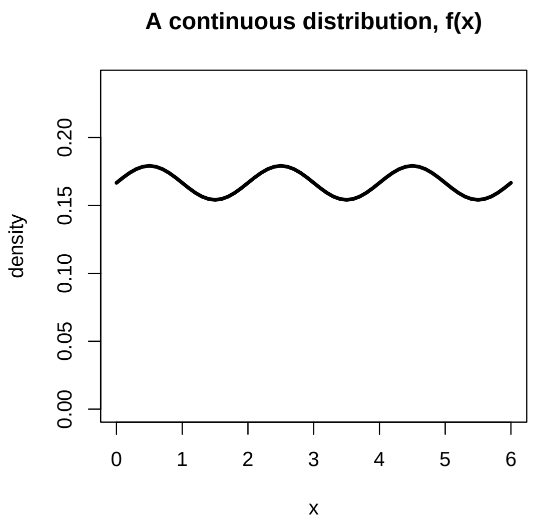
Proof: We prove Proposition 4.9.2 by a counter-example as follows.

Figure 4.10(a) shows a continuous distribution with c.d.f. F and density function $f(X = x) = 1/6$, for $0 < x < 6$. Figure 4.10(b) shows a continuous distribution with c.d.f. F_n and density function $f_n(X_n = x) = 1/6 + (1/(6n))sin(x\pi)$, where $n \geq 1$ and $0 < x < 6$. We have $F(x) = x/6$ and $F_n(x) = x/6 + 1/(6n\pi)(1 - cos(x\pi)))$ for $0 < x < 6$. Clearly $lim_{n\to\infty}F_n(x) = F(x)$ for all x. The high-density cluster tree for F, $T(F)$, is a tree with only a root. The high-density cluster tree for F_n, $T(F_n)$, is a 2-layer tree such that its root has 4 child nodes. The two trees are different not only in their structure but also in their node content. This difference remains the same as n increases. $\square$

From the above two propositions, we realize that in some cases we cannot discover the high-density/mass cluster tree of F by studying the cluster trees constructed for F_n even if $lim_{n\to\infty}F_n(x) = F(x)$ for all x.



(a) Probability density of F



(b) Probability density of F_n

Figure 4.10: The probability density functions of two distributions

4.9.2 A further exploration based on coupling graphs

Proposition 4.9.1 demonstrated, loosely speaking, that $F_n \rightarrow F$ does not imply $T(F_n) \rightarrow T(F)$. This convergence could be formalized because all distributions were discrete and had common finite support. This was not available in Proposition 4.9.2 where the continuous case was considered. There it was shown that the tree structures could never agree in the number of branches (i.e. in the number of separate components of the level set) for any finite n, even though the level sets converge.

In this section, we explore a more complex example in $\mathcal{R}^2$ where a discrete distribution converges to some absolutely continuous distribution. The sequence of discrete distributions each has support on a regular square mesh lattice. Each lattice in the sequence includes all lattices earlier in the sequence. Unfortunately, the support is different for each element in the sequence so our tree distance function is not available. It can however be shown that the area of the lattice formed by the coupling graph components converges to the area of the corresponding continuous density components. Though this does not imply that the tree from the coupling graph on the discrete distribution converges to that of the continuous distribution (N.B. our distance measure is not defined), it is encouraging that the component areas agree in the limit.

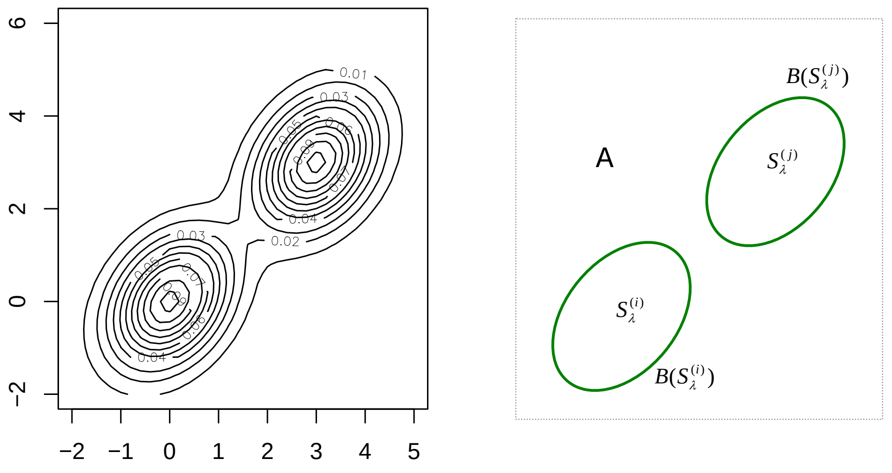
We anticipate, for example, that the same will hold for discrete distributions whose support is not so nicely nested along the sequence of discrete distributions. In Chapters 5 and 6, we explore examples where this is the case. In particular, Sections 5.3.1 and 6.3.1 consider the case of mixtures of "bivariate binomials" (each formed from the product of independent binomials) converging to the corresponding mixture of bivariate normal distributions. In these cases, a distance measure can be introduced and is empirically observed to decrease significantly as n increases.

Before we construct the above discrete distributions, some notation is necessary. Let $L(\lambda; f)$ be a density level set of an absolutely continuous distribution with density function f. Let the c.d.f. of this distribution be F. For simplicity, we consider the support of the distribution to be located in $\mathcal{R}^2$. By the definition of a density level set, we have $f(x) > \lambda$ if and only if $x \in L(\lambda; f)$.

Suppose there are c disjoint maximally and continuously contiguous subsets in $L(\lambda; f)$ and denote the i^{th} such subset by $S_\lambda^{(i)}$. We have $L(\lambda; f) = \cup_{i=1}^{c} S_\lambda^{(i)}$ with $S_\lambda^{(i)} \cap S_\lambda^{(j)} = \phi$ for $i \neq j$. We define $B(S_\lambda^{(i)})$ to be the boundary of $S_\lambda^{(i)}$. For simplicity, we assume that $S_\lambda^{(i)}$ contains no holes. $B(S_\lambda^{(i)})$ is a single closed curve and is surrounded by a continuously contiguous region, say A, such that $f(x) < \lambda$ if $x \in A$. By "surrounded" we mean that for any $a \in B(S_\lambda^{(i)})$ there exists $\delta > 0$ such that for all $x \notin B(S_\lambda^{(i)}) \cup S_\lambda^{(i)}$ with $||x - a|| < \delta$ we have $f(x) < \lambda$.

For example, Fig. 4.11(a) shows a contour of a distribution density f in $\mathcal{R}^2$. Figure 4.11(b) shows a sketch of $L(\lambda; f)$ at $\lambda = 0.03$. In Fig. 4.11(b), $S_\lambda^{(i)}$ and $S_\lambda^{(j)}$ are two disjoint maximally and continuously contiguous subsets in $L(\lambda; f)$. $B(S_\lambda^{(i)})$ and $B(S_\lambda^{(j)})$ are their respective boundaries. The surrounding region is A.

We now construct a discrete distribution on a lattice corresponding to an absolutely continuous distribution. Let F be the c.d.f. for the continuous distribution and let its density function be f. We can put a regular square lattice, say L_α, on the support space of the distribution, where α is the length of each gridline segment.



(a) Contour of a density function (b) Level set at $\lambda = 0.03$

Figure 4.11: A distribution density and its level set

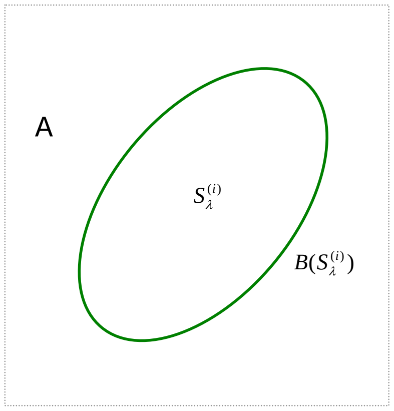
A gridline segment is a line segment between neighbouring vertices in the lattice. Let V be the vertex set from L_α and assign $f(v)$ to each vertex v in V. We can construct a discrete distribution by the vertex set V and a probability mass $h_\alpha f(v)$ on each element in V, where $\sum_{v \in V} h_\alpha f(v) = 1$. We denote this discrete distribution F_α and its probability mass function $m_\alpha(\cdot)$. We have $m_\alpha(v) = h_\alpha f(v)$ if $v \in V$ and $m_\alpha(u) = 0$ if $u \notin V$. We have $m_\alpha(v) > h_\alpha \lambda$ if $v \in S_\lambda^{(i)}$. This implies that all the vertices from L_α located in $S_\lambda^{(i)}$ have probability mass greater than $h_\alpha \lambda$ from the discrete distribution F_α.

The following is a property for the trees constructed by coupling graphs for F_α. Since F_α has support located on a regular square lattice, the high-probability mass cluster tree of F_α constructed by weak coupling graphs can be shown to be identical to that constructed by strong coupling graphs (see Theorem 4.10.2 in Section 4.10). The high-probability mass cluster tree of F_α constructed by the coupling graph method can be shown to be identical to Hartigan's approach, which was briefly introduced in Section 1.1.2 (see Theorem 4.10.1 in Section 4.10). We are interested in the relation of the tree constructed for F_α using our coupling graph method to the high-density cluster tree constructed for F.

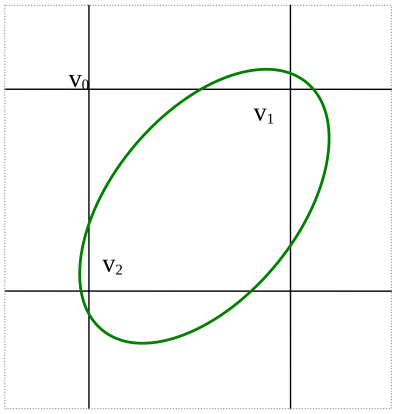
To make the discussion simple, for density level λ, we focus on one of the subsets in $L(\lambda; f)$, say $S_\lambda^{(i)}$. Our discussion in the rest of this section is based on $S_\lambda^{(i)}$. We denote a square mesh in a lattice as a square formed by four gridline segments each connecting two neighbouring vertices in the lattice. If the area of each square mesh in the lattice L_α is not small enough, the coupling graph constructed for F_α at level $h_\alpha\lambda$ might separate vertices from L_α that are located in $S_\lambda^{(i)}$ into different components.

For example, Figure 4.12(a) shows a sketch of $S_\lambda^{(i)}$ with its boundary and the region A with lower density. Figure 4.12(b) shows a lattice L_α on $S_\lambda^{(i)}$ and region A. In Fig. 4.12(b), v_1 and v_2 are vertices located in $S_\lambda^{(i)}$. We have $v_1, v_2 \in L(h_\alpha\lambda; m_\alpha)$. Since v_0 is a vertex outside $L(h_\alpha\lambda; m_\alpha)$ and its distance to v_1 is smaller than the distance between v_1 and v_2, by the definition of our coupling graph, v_1 and v_2 are not connected in the coupling graph constructed for F_α at level $h_\alpha\lambda$. Let V be the vertex set from the lattice L_α. We have $V \cap S_\lambda^{(i)}$ as the subset from V such that each vertex in this subset is located in $S_\lambda^{(i)}$. As Figure 4.12(b) shows, if the mesh of L_α is not small enough, $V \cap S_\lambda^{(i)}$ may not be a vertex set of a component in the coupling graph of F_α at level $h_\alpha\lambda$.

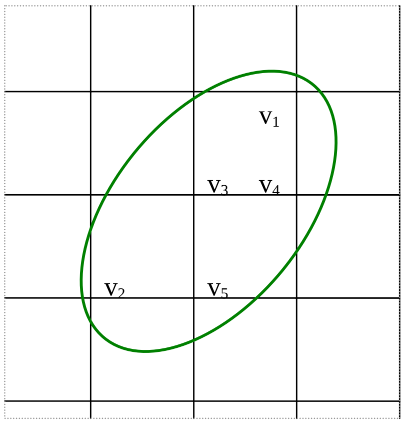
We construct a sequence of lattices and a corresponding sequence of discrete distributions as follows. We add a new horizontal line halfway between the existing horizontal lines in L_α and a new vertical line halfway between the existing vertical lines in L_α. In this way, we construct a new lattice $L_{\alpha/2}$ with each gridline segment having length $\alpha/2$. If we continue, we can construct a sequence of lattices, L_α, $L_{\alpha/2}$, $L_{\alpha/2^2}$, $\cdots$, $L_{\alpha/2^k}$, $\cdots$. Note that a vertex in $L_{\alpha/2^i}$ is still a vertex in $L_{\alpha/2^{i+1}}$ and the length of each gridline segment in $L_{\alpha/2^{i+1}}$ is half of that in $L_{\alpha/2^i}$. We denote the c.d.f. and mass function of the discrete distribution corresponding to a lattice $L_{\alpha/2^i}$ by $F_{\alpha/2^i}$ and $m_{\alpha/2^i}$ respectively.



(a) Sketch of $S_\lambda^{(i)}$



(b) Lattice with large mesh



(c) Lattice with small mesh

Figure 4.12: $S_\lambda^{(i)}$ and lattice

For example, Figure 4.12(c) shows a lattice $L_{\alpha/2}$ constructed from the lattice L_α shown in Fig. 4.12(b). Let V be the vertex set from the lattice $L_{\alpha/2}$. By our coupling graph method, in the example shown in Fig. 4.12(c), $V \cap S_\lambda^{(i)}$ is a vertex set from a component in the coupling graph of $F_{\alpha/2}$ at level $h_{\alpha/2}\lambda$.

Before giving a convergence discussion of the relation between components from coupling graphs and the corresponding component, say $S_\lambda^{(i)}$, from the level set of the continuous distribution at level λ, we prove a convergence property of square meshes in a lattice if they are located in $S_\lambda^{(i)}$. This property is given in the following Proposition 4.9.3. Some notation is necessary. Let $I_{(j)}$ be the largest set of square meshes from the lattice $L_{\alpha/2^j}$ contained in $S_\lambda^{(i)}$. Let $U_{(j)}$ be the smallest set of square meshes from the lattice $L_{\alpha/2^j}$ such that $U_{(j)}$ contains $S_\lambda^{(i)}$. Let $a(I_{(j)})$ and $a(S_\lambda^{(i)})$ be the areas of $I_{(j)}$ and $S_\lambda^{(i)}$ respectively.

Proposition 4.9.3 *Suppose $B(S_\lambda^{(i)})$ is closed, bounded, of measure zero, and has non-empty interior. We have $\{a(I_{(j)})\}$ to be a monotonic increasing and upper-bounded sequence. It converges to $a(S_\lambda^{(i)})$, i.e. $lim_{j\to\infty} a(I_{(j)}) = a(S_\lambda^{(i)})$.*

Proof: Each square mesh in $L_{\alpha/2^j}$ is divided into four smaller square meshes in $L_{\alpha/2^{j+1}}$ such that the length of each edge for each square mesh in $L_{\alpha/2^{j+1}}$ is half of that in $L_{\alpha/2^j}$. All four square meshes from $L_{\alpha/2^{j+1}}$ that are inside a square mesh in $L_{\alpha/2^j}$ are contained in $I_{(j+1)}$ if that mesh in $L_{\alpha/2^j}$ is contained in $I_{(j)}$. One or more square meshes from $L_{\alpha/2^{j+1}}$ that are inside a square mesh in $U_{(j)} - I_{(j)}$ may be assigned into $I_{(j+1)}$. Therefore, we have $I_{(0)} \subseteq I_{(1)} \subseteq I_{(2)} \subseteq \cdots \subseteq S_\lambda^{(i)}$ and $a(I_{(0)}) \leq a(I_{(1)}) \leq a(I_{(2)}) \leq \cdots \leq a(S_\lambda^{(i)})$. For example, Fig. 4.13(a) shows a sketch of $S_\lambda^{(i)}$ and a lattice $L_{\alpha/2^j}$. Figure 4.13(b) shows the corresponding mesh set $I_{(j)}$. Figures 4.13(c) and (d) show $L_{\alpha/2^{j+1}}$ and $I_{(j+1)}$ respectively. It is clear that $a(I_{(j)}) < a(I_{(j+1)}) < a(S_\lambda^{(i)})$ in this example.

Since the sequence $\{a(I_{(j)})\}$ is monotonic increasing and has an upper bound, we have $lim_{j\to\infty} a(I_{(j)}) = a_0$ say. We need to show that for any point x in $S_\lambda^{(i)}$ and for any $\epsilon > 0$ that no matter how small it is, we can find an integer N such that for all $n > N$ the point x is contained in $I_{(n)}$, where $I_{(n)}$ is the largest set of square meshes from the lattice $L_{\alpha/2^n}$. Then we will have that $lim_{j\to\infty} a(I_{(j)}) = a(S_\lambda^{(i)})$.

Take any existing point $x \in (S_\lambda^{(i)} - I_{(j)})$, since $S_\lambda^{(i)}$ is an open set, there exists an open square, say s, of width $\epsilon > 0$ and centred at x, such that $s \subset (S_\lambda^{(i)} - I_{(j)})$.

Let N be an integer such that $\alpha/2^N < \epsilon/2$ and let $L_{\alpha/2^n}$ be any lattice with mesh size $\alpha/2^n$ where $n > N$. There will always be a horizontal gridline of $L_{\alpha/2^n}$ intersecting s but above its centre x. To see this, assume there is no such gridline line. Then, because x is the centre of s, the distance between two neighbouring horizontal gridlines of $L_{\alpha/2^n}$ is at least $\epsilon/2$, which is not true whenever $\alpha/2^n < \alpha/2^N < \epsilon/2$. Similarly, there must be a horizontal gridline line of $L_{\alpha/2^n}$ that intersects s and is located below x. So x must be located between two horizontal gridlines of $L_{\alpha/2^n}$ intersecting s. Similarly, we have x must be located between two vertical gridlines of $L_{\alpha/2^n}$ intersecting s. We denote by R_x the rectangle formed by these two horizontal and two vertical gridlines. We have $x \in R_x$ and $R_x \subset s \subset S_\lambda^{(i)}$. Since R_x is either a square mesh or the union of more than one square mesh of $L_{\alpha/2^n}$, we have $R_x \subset I_{(n)}$.

Therefore no matter what the size $\epsilon > 0$ is, we can always find an integer N such that for any $n > N$, $x \in I_{(n)}$. This proves $lim_{j\to\infty} a(I_{(j)}) = a(S_\lambda^{(i)})$. $\square$

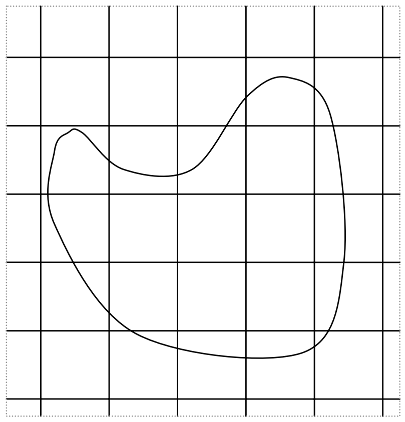
We have proved that the area of the union of all the square meshes in $S_\lambda^{(i)}$ converges to the area of $S_\lambda^{(i)}$. However, such a union of square meshes is not a coupling graph component. Moreover, not all the vertices from the mesh set $I_{(j)}$ can be contained in a coupling graph component constructed for $F_{\alpha/2^j}$ at level $h_{\alpha/2^j}\lambda$, and the component may contain vertices outside $S_\lambda^{(i)}$.

Let $C^{(k)}$ be a component of the coupling graph constructed for $F_{\alpha/2^k}$ at level $h_{\alpha/2^k}\lambda$ (Recall h_α is the normalizing constant for meshes of width α as defined early in this section). Let $M_{(k)}$ be the set of all square meshes in the lattice $L_{\alpha/2^k}$ whose four vertices are in the component $C^{(k)}$. We are interested in the area of $M_{(k)}$ as k increases. To study this area, it is necessary to explore the connectivity by coupling graph for any two vertices from a lattice if they are located inside $S_\lambda^{(i)}$.

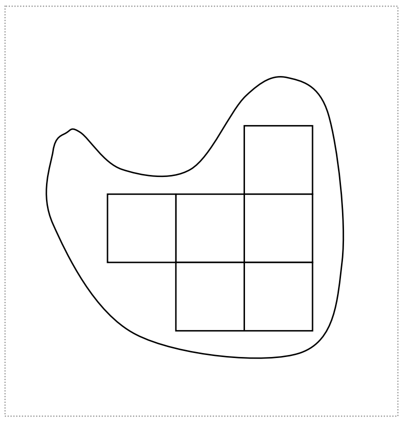
We want to show that any two vertices inside $S_\lambda^{(i)}$ on a lattice L_α can be connected by a coupling graph constructed for the discrete distribution corresponding to a lattice with smaller mesh width. We have the following proposition.

Proposition 4.9.4 *Suppose L_α is a square mesh lattice of mesh width $\alpha > 0$ and* *let v_1 and v_2 be any two vertices from L_α in $S_\lambda^{(i)}$ for some $\lambda > 0$. Then there exists an integer $N > 0$ such that for all $n > N$, v_1 and v_2 are connected by coupling graph at level $h_{\alpha/2^n}\lambda$ for the discrete distribution $F_{\alpha/2^n}$.*

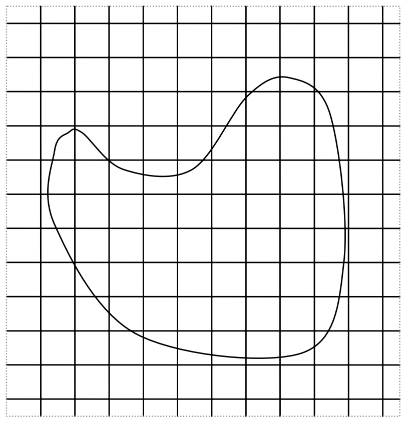
To prove Proposition 4.9.4, we need the following lemma.



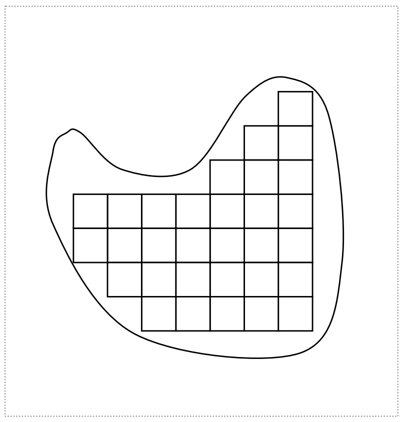
(a) $S_{\lambda}^{(i)}$ and $L_{\alpha/2^j}$



(b) Mesh set $I_{(j)}$



(c) $S_{\lambda}^{(i)}$ and $L_{\alpha/2^{j+1}}$



(d) Mesh set $I_{(j+1)}$

Figure 4.13: $S_{\lambda}^{(i)}$ and the mesh set

Lemma 4.9.1 *Let v_1 and v_2 be any two vertices from the lattice L_α in $S_\lambda^{(i)}$. Suppose there exist two rectangles R_1 and R_2 with sides parallel to the gridline directions of the lattice such that $v_1 \in R_1$, $v_2 \in R_2$, $R_1 \cup R_2 \subset S_\lambda^{(i)}$, and R_2 is attached to R_1 on one of its vertical edges with the length of the shared edge segment to be non-zero. Then $\exists N_1 > 0$ such that $\forall n > N_1$, there is a path in $S_\lambda^{(i)}$ formed by gridline segments of the lattice $L_{\alpha/2^n}$ that connects v_1 and v_2.*

Proof: We denote the shared part of the edge from R_1 and R_2 by e_s. Let $\epsilon = min(||e_s||, d_1, d_2)$, where d_1 and d_2 are the widths of R_1 and R_2 respectively. There exists an integer N_1 such that $\alpha/2^{N_1} < \epsilon$. For any integer n such that $n \geq N_1$, we can construct a lattice $L_{\alpha/2^n}$. Since the length of each mesh width in the lattice $L_{\alpha/2^n}$ is less than ϵ and the rectangle R_1 has gridline directions, any two vertices from $L_{\alpha/2^n}$ in R_1 are connected by gridline segments of $L_{\alpha/2^n}$ in $S_\lambda^{(i)}$. Similarly, any two vertices from $L_{\alpha/2^n}$ in R_2 are connected by gridline segments of $L_{\alpha/2^n}$ in $S_\lambda^{(i)}$.

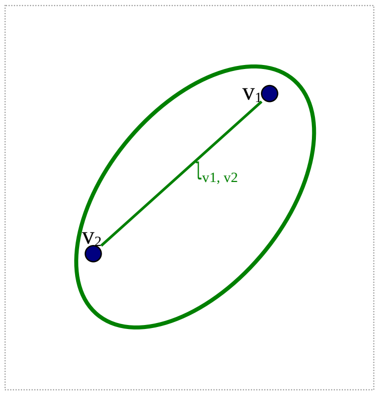
There is at least a horizontal line say l_s from the lattice $L_{\alpha/2^n}$ intersecting e_s and there are at least two vertices from the lattice located on l_s such that one is inside R_j and one is inside R_{j+1}. These two vertices are connected by gridline segments of $L_{\alpha/2^n}$ in $S_\lambda^{(i)}$. Therefore, for the pair of vertices v_1 and v_2, there is a path in $S_\lambda^{(i)}$ formed by gridline segments of $L_{\alpha/2^n}$ such that the two vertices are connected by the path. $\square$

Lemma 4.9.1 leads to Lemma 4.9.2 as follows.

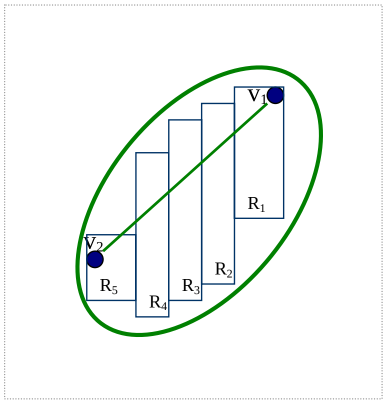
Lemma 4.9.2 *Let v_1 and v_2 be any two vertices from L_α in $S_\lambda^{(i)}$. There exists an integer $N > 0$ such that for all $n > N$, v_1 and v_2 are connected by gridline segments of the lattice $L_{\alpha/2^n}$ in $S_\lambda^{(i)}$.*

Proof: We construct a rectangle say R_1 in $S_\lambda^{(i)}$ with sides parallel to the gridline directions such that R_1 contains v_1. Let l_{v_1,v_2} be the shortest continuous path in $S_\lambda^{(i)}$ that connects v_1 and v_2. We construct a rectangle R_2 in $S_\lambda^{(i)}$ with sides parallel to the gridline directions such that R_2 is attached to R_1 on one of its vertical edges and R_2 intersects with l_{v_1,v_2}. Let p_1 be any point on l_{v_1,v_2} located in R_1. We choose the edge to be such that for any $p_2 \in R_2$ on l_{v_1,v_2}, p_2 lies between p_1 and v_2 along l_{v_1,v_2}.

We continue, constructing rectangle R_{j+1} from R_j as we construct R_2 from R_1, until we construct R_k such that R_k contains v_2. We denote the shared part of the edge from R_j and R_{j+1} by e_j. We further require that $||e_j|| > 0$, where $||e_j||$ is the length of e_j.



(a) Path connecting v_1 and v_2 inside $B(S_\lambda^{(i)})$



(b) Sequence of rectangles

Figure 4.14: Path connecting two vertices and constructed rectangles

Figure 4.14(a) shows an example of the path l_{v_1,v_2} connecting v_1 and v_2 in $S_\lambda^{(i)}$. Figure 4.14(b) shows a sequence of rectangles constructed for v_1 and v_2 by the above method.

By Lemma 4.9.1, there exists an integer N_j such that $\forall n > N_j$, any two vertices from the lattice $L_{\alpha/2^n}$ are connected by a path in $S_\lambda^{(i)}$ formed by gridline segments of $L_{\alpha/2^n}$ if the two vertices are located in $R_j \cup R_{j+1}$ with $j = 1, 2, \cdots, k-1$. Let $N = max(N_1, N_2, \cdots, N_{k-1})$, then $\forall n > N$, there is a path in $S_\lambda^{(i)}$ formed by gridline segments of the lattice $L_{\alpha/2^n}$ such that any two vertices including v_1 and v_2 from the lattice are connected by the path if they are located in $S_\lambda^{(i)}$. $\square$

We are now in a position to prove Proposition 4.9.4.

Proof: If a gridline segment of a lattice $L_{\alpha/2^n}$ is located in $S_\lambda^{(i)}$, it connects two vertices with probability mass greater than $h_{\alpha/2^n}\lambda$ in the discrete distribution $F_{\alpha/2^n}$, and it can be shown that this gridline segment appears in the coupling graph constructed for $F_{\alpha/2^n}$ at level $h_{\alpha/2^n}\lambda$ (see Lemma 4.10.1 in Section 4.10.1). Therefore Proposition 4.9.4 is proved by Lemma 4.9.2. □

We are now ready to show that the area of the lattice formed by the coupling graph components converges to the area of the corresponding continuous density components. To be more specific, recall that $M_{(k)}$ is defined to be the set of all square meshes in the lattice $L_{\alpha/2^k}$ whose four vertices are in $C^{(k)}$, where $C^{(k)}$ is a component of the coupling graph constructed for $F_{\alpha/2^k}$ at level $h_{\alpha/2^k}\lambda$. We are interested in the area of $M_{(k)}$. Proposition 4.9.5 follows from Proposition 4.9.4.

Proposition 4.9.5 *Let $I_{(j)}$ be the largest set of square meshes from the lattice $L_{\alpha/2^j}$ that is contained in $S_\lambda^{(i)}$. Then there exists an integer N_{a_j} such that $\forall k_j > N_{a_j}$, $a(M_{(k_j)}) \geq a(I_{(j)})$, where $a(M_{(k_j)})$ is the area of $M_{(k_j)}$.*

Proof: By Proposition 4.9.4, there exists an integer say N_{a_j} such that $\forall k_j > N_{a_j}$, all the vertices from $I_{(j)}$ are connected by the coupling graph say g_{k_j} constructed for $F_{\alpha/2^{k_j}}$ at level $h_{\alpha/2^{k_j}}\lambda$. It is clear that $k_j \geq j$. By the coupling graph, each square mesh say m of the lattice $L_{\alpha/2^{k_j}}$ is contained in $M_{(k_j)}$ if m is located in a mesh of $I_{(j)}$. We have $a(M_{(k_j)}) \geq a(I_{(j)})$. □

In Proposition 4.9.5, $M_{(k_j)}$ may contain some meshes not in $S_\lambda^{(i)}$. It is necessary to find some integer say n such that no vertex outside $S_\lambda^{(i)}$ is contained in any component of g_n if the component contains at least a vertex in $S_\lambda^{(i)}$, where g_n is the coupling graph constructed for $F_{\alpha/2^n}$ at level $h_{\alpha/2^n}\lambda$. We have the following proposition.

Proposition 4.9.6 *Let $S = B(S_\lambda^{(i)}) \cup S_\lambda^{(i)}$. Suppose $\forall x \in S$ and $\forall y \notin S$, $\exists \epsilon > 0$ such that $d(x,y) > \epsilon$ if $f(y) \geq \lambda$, where $d(x,y)$ is the distance between x and y. Then there exists an integer N' such that $\forall n > N'$, any two points v_0 and v_1 are not connected in g_n if v_0 and v_1 are vertices from the lattice $L_{\alpha/2^n}$, $v_0 \in S$ and $v_1 \notin S$, where g_n is the coupling graph constructed for $F_{\alpha/2^n}$ at level $h_{\alpha/2^n}\lambda$.*

Proposition 4.9.6 can be proved directly similarly to the proof of Proposition 4.9.3.

From Proposition 4.9.5, we have $a(M_{(k_j)}) \geq a(I_{(j)})$. (Note that $M_{(k_j)}$ and $I_{(j)}$ are denoted in Proposition 4.9.5 and its proof.) By Propositions 4.9.5 and 4.9.6, there exists an integer $N = max(N_{a_j}, N')$ (note that N_{a_j} and N' are denoted in Propositions 4.9.5 and 4.9.6 respectively) such that $\forall k'_j > N$, we have $a(S_\lambda^{(i)}) \geq a(M_{(k'_j)}) \geq a(I_{(j)})$. It is easy to make $\{k'_j\}$ a sequence of monotonically increasing integers since $\{j\}$ is a sequence of monotonically increasing integers. We have $\{a(M_{(k'_j)})\}$ to be a monotonically increasing sequence as well. By Proposition 4.9.3, the sequence $\{a(M_{(k'_j)})\}$ converges to $a(S_\lambda^{(i)})$. This implies that the area of the lattice formed by the coupling graph components converges to the area of the corresponding continuous density components.

In many real cases, since it is hard to know the true density on each vertex on a lattice, the discrete distribution constructed from the lattice is hard to obtain. However, the Monte Carlo method together with our clustering distance measure is a good means for examining the convergence of cluster trees empirically in some real cases. This is discussed in Chapters 5 and 6.

4.10 Some proofs

We prove Theorem 4.10.1 and Theorem 4.10.2 in this section. The following notation is necessary. Given a multivariate discrete random variable X with distribution function F_X, suppose the support of F_X is D which is the vertex set of a regular square mesh lattice. Let a gridline segment be a line segment between two neighbouring vertices from the lattice. Let $P(X)$ be the probability function of X. We have X located on a lattice. Note that we use $G_\lambda(X)$ as generic notation for both the weak and strong coupling graph of F_X at level λ.

Theorem 4.10.1 *The vertex sets of the components from the graph constructed by Hartigan's approach at level λ for F_X are identical to those from $G_\lambda(X)$.*

The following two lemmas lead to the above theorem and reveal further properties of the coupling graph method.

Lemma 4.10.1 *For any two vertices sharing a gridline segment, if both vertices are inside a level set $L(\lambda; P(X))$, then their gridline segment appears in $G_\lambda(X)$.*

Proof: Let x_1 and x_2 be any pair of vertices sharing a gridline segment e in the lattice. We have $P(x_1) \geq \lambda$ and $P(x_2) \geq \lambda$. Because the length of a gridline segment in a lattice is the shortest possible distance among any pair of vertices, we have $\forall z \in D - L(\lambda; P)$, $||x_1 - x_2|| = ||e|| \leq ||x_1 - z||$ and $||x_1 - x_2|| \leq ||x_2 - z||$. By Definitions 4.5.4, 4.5.5, and 4.5.6, e must appear in both the weak and the strong coupling graph of F_X at level λ. □

Lemma 4.10.2 *If there exists a non-gridline edge in $G_\lambda(X)$ between x_1 and x_2, then there must exist a gridline segment path in $G_\lambda(X)$ between x_1 and x_2.*

Proof: Suppose e is a non-gridline edge in $G_\lambda(X)$ between x_1 and x_2. We can always construct a smallest hyper-box made by gridlines in the lattice with e as a diagonal of the hyper-box. Let $B = \{x | x \in D$ and x is inside or on a face of the hyper-box$\}$. By the Pythagorean theorem, we have $\forall x \in B$, $||e|| \geq ||x - x_1||$ and $||e|| \geq ||x - x_2||$. Therefore, by Definition 4.5.4, we have $P(x \in B) \geq \lambda$.

We can always find a path in the hyper-box connecting x_1 and x_2 such that this path is made by gridline segments. Let e_1 be any gridline segment in the path that connects two vertices say x_a and x_b. Since $P(x_a) \geq \lambda$ and $P(x_b) \geq \lambda$, by Lemma 4.10.1, e_1 must be an edge in $G_\lambda(X)$. Therefore, the above path must appear in $G_\lambda(X)$ between x_1 and x_2. □

According to Hartigan's approach, a graph can be constructed at a mass level λ in the following way: for any two points in $L(\lambda; P(X))$, add an edge to connect them if they share a gridline segment [32].

Based on the above two lemmas, Theorem 4.10.1 is proved as follows.

Proof: By Lemma 4.10.1, if two points from the support of X are connected by an edge according to Hartigan's approach at a certain mass level λ, they are also connected in the coupling graph $G_\lambda(X)$. By Lemma 4.10.2, if two points from the support of X are connected by an edge in the coupling graph $G_\lambda(X)$, there exists a gridline segment path to connect these two points and they are also connected by this path in the graph constructed by Hartigan's approach at the same level. Therefore, the vertex sets of the components from the graph constructed by Hartigan's approach at level λ for F_X are identical to those from $G_\lambda(X)$. $\square$

Note that early in this section, we supposed the support of F_X to be D, the vertex set of a regular square mesh lattice. We have the following theorem.

Theorem 4.10.2 *Given a multivariate discrete random variable X with distribution function F_X, suppose the support of F_X is the vertex set of a regular square mesh lattice, the high-probability mass cluster tree built by weak coupling graphs of F_X is identical to that built by strong coupling graphs.*

To prove Theorem 4.10.2, the following lemmas are necessary.

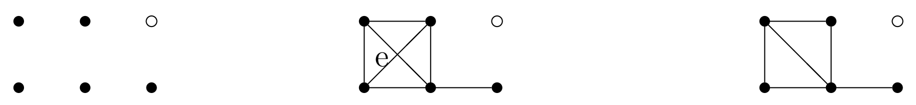
Lemma 4.10.3 *A gridline segment is an edge in the λ strong coupling graph of F_X if and only if it is an edge in the λ weak coupling graph of F_X.*

Proof: Let e be a gridline segment that connects two vertices x_1 and x_2 in the lattice of the support of X. If e appears as an edge in the λ weak coupling graph of F_X, by definition, we have $P(x_1) \geq \lambda$ and $P(x_2) \geq \lambda$. By Lemma 4.10.1, e must appear in the strong coupling graph of F_X at level λ as well. If e appears as an edge in the λ strong coupling graph of F_X, by definition of coupling graph, e is an edge in the λ weak coupling graph of F_X as well. $\square$

Lemma 4.10.4 *A path consisting only of gridline segments in the lattice is a path in the λ strong coupling graph of F_X if and only if it is a path in the λ weak coupling graph of F_X.*

Proof: If a path appears in the λ weak coupling graph of F_X and any edge say e in this path is a gridline segment, then by Lemma 4.10.3, e is an edge in the strong coupling graph of F_X. Thus, all the edges in a path in the λ weak coupling graph form a path in the λ strong coupling graph of F_X. By definition of coupling graph, if a path appears in the λ strong coupling graph of F_X, it must be a path in the λ weak coupling graph of F_X. $\square$

Lemma 4.10.3 addresses a property for a gridline segment edge in the coupling graph of X that has its support on a lattice. There is no such property if the edge is not a gridline segment. Suppose the support of X is a lattice with only the six vertices shown in Fig. 4.15. Then any point with a solid circle say x has $P(x) \geq \lambda$



(a) Level set at λ (b) λ weak coupling graph (c) λ strong coupling graph

Figure 4.15: An example of coupling graphs on a lattice support.

and any point with a hollow circle say x_0 has $P(x_0) < \lambda$. Thus, the edge e appears in the λ weak coupling graph of F_X but is not in the λ strong coupling graph of F_X. This example also shows that, for a given λ, the weak coupling graph may not be identical to the strong coupling graph even if all the vertices are located on a lattice. However, we can prove that, under the condition of a lattice support, given a λ, the vertex set of a weak component is identical to that of a strong component. For that proof, the following lemma is necessary.

Lemma 4.10.5 *If points x_1 and x_2 are connected in the coupling graph $G_\lambda(X)$ of F_X, there must exist a path between these two points in $G_\lambda(X)$ such that every edge in this path is a gridline segment in the lattice with the set of vertices being the support of F_X.*

Proof: If x_1 and x_2 are connected in $G_\lambda(X)$, there must exist a path between them in $G_\lambda(X)$. For any non-gridline edge say e in this path, suppose e connects two vertices x_a and x_b. By Lemma 4.10.2, there always exists a path in $G_\lambda(X)$ connecting x_a and x_b such that this path is made by gridline segments. We can always replace e

by this gridline segment path. After every non-gridline edge is replaced by a gridline segment path, we obtain a path made by gridline segments such that it connects x_1 and x_2 in $G_\lambda(X)$. $\square$

Lemma 4.10.5 also implies that, for any two points in a component of $G_\lambda(X)$, there exists a gridline segment path between them.

Lemma 4.10.6 *The vertex set of a λ weak component of F_X is identical to the vertex set of a λ strong component of F_X.*

Proof: Given λ, let C_1 be the vertex set of the λ weak component of F_X. If C_1 contains only one point say x_0, $\exists z \in D - L(\lambda; P(X))$ such that $\forall y \in L(\lambda; P(X))$, if $y \neq x_0$, $||x_0 - y|| > ||x_0 - z||$. Thus, by Definitions 4.5.4, 4.5.5, and 4.5.6, C_1 is a vertex set of the λ strong component of F_X as well.

If C_1 contains at least two points, let x_1 and x_2 be any pair of points in C_1. Since x_1 and x_2 are connected in the λ weak coupling graph of F_X, by Lemma 4.10.5 there must exist a gridline segment path in this λ weak coupling graph that connects x_1 and x_2. By Lemma 4.10.4, this path must exist in the λ strong coupling graph of F_X as well. Therefore, there must exist a strong component with vertex set say C_2 such that $x_1, x_2 \in C_2$. This implies that $C_1 \subseteq C_2$.

By definition of coupling graph, we also have $C_2 \subseteq C_1$. Therefore $C_1 = C_2$ and the lemma is proved. $\square$

We can use the coupling graphs of F_X at different levels as contiguity graphs to build a high-probability mass cluster tree. Theorem 4.10.2 follows directly from Lemma 4.10.6 and is proved as follows.

Proof: By Lemma 4.10.6, for every λ, the vertex sets of components in the λ weak coupling graph of F_X are identical to those in the λ strong coupling graph of F_X. If we use coupling graphs as contiguity graphs to build a high-probability mass cluster tree, the tree built by the weak coupling graphs of F_X is identical to that built by the strong coupling graphs. $\square$

Note that the two theorems proved in this section may not hold if the support of the distribution is not the vertex set of a regular square mesh lattice.

Chapter 5

Assess clustering via Monte Carlo

5.1 Chapter summary

The cluster-tree distance measure provides a way to compare estimates of a high-density or high-probability mass cluster tree. In many real cases, we do not know the underlying distribution and therefore can not compare the estimates to the true tree. The distance measure can be used to compare different clustering methods. However, the Monte Carlo method makes it possible to calculate the difference between a cluster tree estimate and its estimand and provides a direct way to assess clustering.

Our distance measure is designed for cluster trees constructed for a fixed finite sample. A high-density cluster tree is constructed on a continuous support. To make our distance measure work, we first define a high-density cluster tree for a set S. This definition makes our distance measure feasible for calculating the difference between a cluster tree estimate and its estimand. We further develop an algorithm to numerically construct a high-density cluster tree for a sample.

The main purpose of this chapter is to set up methodologies to assess clustering via Monte Carlo and our distance measure. The methodologies include the comparison of clustering performance via the difference between an estimate and its estimand and an empirical examination of convergence for cluster trees.

5.2 Performance comparison

By Monte Carlo, we can generate samples from a known distribution and find estimates of the true tree using different clustering methods. These estimates are cluster trees built for a sample. Let S be a sample drawn from a continuous distribution F. Let $CT(F)$ be the high-density cluster tree of F. To compare the estimates built for S, we need to construct a high-density cluster tree for the set S.

Definition 5.2.1 A high-density cluster tree for a set S *drawn from a continuous distribution* F*, denoted* $CT_S(F)$*, is a tree constructed in the following way: for each node* N *in* $CT(F)$*, construct a node* N_S *in* $CT_S(F)$ *such that* $Content(N_S) = Content(N) \cap S$*. If* $Content(N_S) = \phi$*, do not construct this node. If* N_1 *is the parent node of* N_2 *in* $T(F)$*, let* $N_{1,S}$ *and* $N_{2,S}$ *be the nodes in* $CT_S(F)$ *that are constructed corresponding to* N_1 *and* N_2 *respectively, then* $N_{1,S}$ *is a parent node of* $N_{2,S}$*.*

Similarly, if F is a discrete distribution, we can define a high-probability mass cluster tree for a set S. For convenience, the following notation is necessary.

Notation 5.2.1 *We denote an estimate of* $CT(F)$ *by* $\widehat{CT}(F)$*. If there are* m *different estimates of* $CT(F)$*, we denote them* $\widehat{CT_1}(F), \widehat{CT_2}(F), \cdots, \widehat{CT_m}(F)$*. If these estimates are built for a sample* S*, we further denote them* $\widehat{CT_{1,S}}(F), \widehat{CT_{2,S}}(F), \cdots, \widehat{CT_{m,S}}(F)$*.*

Using our distance measure, $d(\widehat{CT_S}(F), CT_S(F))$ shows how close the estimate built for S is to the true cluster tree for S. We use this distance to assess the performance of a clustering method.

For a distribution, we can construct its high-probability density/mass cluster tree. For a sample drawn from a distribution, we can construct the high-probability density/mass cluster tree for this sample. For any clustering obtained for this sample, if the clustering result already has a tree structure, such as that produced by single linkage, we can use this tree directly as an estimate of the high-probability density/mass cluster tree. If the clustering result is a partition, such as that produced by k-means, we can construct a two-layer cluster tree with the root containing all the sample points and each of the child nodes containing a cluster from the partition.

For the clustering methods that produce different partitions via different parameter settings or different random starts, we can construct a cluster tree using our method of partition integration.

For any clustering method, we can always construct a cluster tree for the sample as an estimate of the high-probability density/mass cluster tree for the underlying distribution. We can compare the performance of clustering methods using our distance measure to calculate the distance from any estimate generated by a method to the true cluster tree for the sample.

The methodology for comparing clustering performance is as follows.

1. Construct a mixture of several continuous distributions and let its c.d.f. be F.
2. Draw a sample of sample size n, say S_n, from F.
3. Construct the true high-density cluster tree of F on S_n and denote this tree $CT_{S_n}(F)$.
4. For each clustering method M_i, build a cluster tree for S_n and denote this tree $\widehat{CT_{i,S_n}}(F)$.
5. For each tree $\widehat{CT_{i,S_n}}(F)$, calculate $d(\widehat{CT_{i,S_n}}(F), CT_{S_n}(F))$ using our cluster-tree distance measure.
6. Repeat step 2 to step 5 several times such that each time we have a different independent sample S_n drawn from F with n fixed.
7. Repeat the above steps several times with increasing values of n.

The above methodology is also suitable if the distribution is discrete.

Let $\widehat{CT_{j,S_n}}(F)$ be the cluster tree built by a method say M_j for a sample say S_n from the distribution F. It is an estimate of $CT(F)$ as well. The bias of this estimate can be estimated by $\frac{1}{p}\sum_{i=1}^{p} d(\widehat{CT_{j,S_n^{(i)}}}(F), CT(F))$; here we draw the sample with size n from F p times. To use our cluster-tree distance measure, we use the true cluster tree for each sample $S_n^{(i)}$, denoted $CT_{S_n^{(i)}}(F)$, to replace $CT(F)$. The bias can be further estimated by $\frac{1}{p}\sum_{i=1}^{p} d(\widehat{CT_{j,S_n^{(i)}}}(F), CT_{S_n^{(i)}}(F))$.

5.3 An empirical examination of convergence

Under a certain transformation, a discrete distribution may converge in distribution to a continuous distribution. For example, a Binomial can converge in distribution to a Normal under a normalized transform. A high-probability mass cluster tree can be constructed for such a discrete distribution using our coupling graph method. We are interested in the convergence of the high-probability mass cluster tree constructed for such a discrete distribution. We use our distance measure and the Monte Carlo method to check whether or not this tree converges to the high-density cluster tree of the corresponding continuous distribution.

The purpose of the above convergence examination is not only to discover a property of our clustering parameter but also to set up a mechanism to reveal the behaviour of a sequence of different cluster trees. We can also use this mechanism to assess the performance of different clustering methods that construct a sequence of partitions or cluster trees for samples of increasing size.

Let $F_1, F_2, \cdots, F_n$ be a sequence of discrete distributions. Let $CT(F_n)$ be the high-probability mass cluster tree constructed by our coupling graph method for F_n. If there exists a continuous distribution F such that $lim_{n\to\infty}F_n(x) = F(x)$ for all x we define $CT(F)$ to be the high-density cluster tree of F. We are interested in the convergence from $CT(F_n)$ to $CT(F)$.

In preparation for the experiments conducted in Section 6.3, we need to answer the following two questions. The first is, how to set up a common grid as the support of a mixture of transformed Binomials. We address this question in Section 5.3.1. The second is, how to estimate the density on each vertex of a grid by a given sample. We address this qustion in Section 5.3.2. Both Section 5.3.1 and 5.3.2 also explain how to calculate the distance of cluster trees.

5.3.1 Binomial and Normal

We start our exploration by considering the Binomial case. Let $X_1 \sim Bin(1,p), X_2 \sim Bin(2,p), \cdots, X_n \sim Bin(n,p)$, where $0 < p < 1$. Let $Y_i = \frac{X_i - ip}{\sqrt{ip(1-p)}}$ where $i = 1, 2, \ldots, n$. Let $F_1, F_2, \cdots, F_n$ be the c.d.f.s of $Y_1, Y_2, \cdots, Y_n$. Let $Z \sim N(0,1)$ and F be its c.d.f. We have $lim_{n\to\infty} F_n(x) = F(x)$ for all x. We use the Monte Carlo method together with our distance measure to simulate the convergence from $CT(F_n)$ to $CT(F)$.

We set up the examination in 2-d. If we put a mixture of two Binomials with the same support in one dimension and a third Binomial with the same support in the second dimension, then we can try to apply the transform that makes a Binomial converge in distribution to a Normal. The goal is to make the mixture of Binomials converge in distribution to a mixture of two bivariate Normals. However, this does not work directly for the mixture of Binomials. The following example shows why. Let $X_{n,1} \sim Bin(n,p_1), X_{n,2} \sim Bin(n,p_2)$, and $p_1(1-p_1) \neq p_2(1-p_2)$. Using the standard transform, we have $Y_{n,1} = \frac{X_{n,1} - np_1}{\sqrt{np_1(1-p_1)}} \sim N(0,1)$ and $Y_{n,2} = \frac{X_{n,2} - np_2}{\sqrt{np_2(1-p_2)}} \sim N(0,1)$, but $Y_{n,1}$ and $Y_{n,2}$ appear on different grids.

We consider the following method to make a transformed mixture of Binomials converge to a mixture of Normal distributions. For simplicity, we let each Binomial in the mixture have the same support. Let $X \sim \sum_{i=1}^{k} \pi_i B_i$ be the mixture of Binomials where $\sum_{i=1}^{k} \pi_i = 1$, $\pi_i > 0$, $B_i = Bin(n, p_i)$. Let p be any constant such that $0 < p < 1$. We define a new common support set for the transformed mixture of Binomials by $S_Y = \{\frac{1-np}{\sqrt{np(1-p)}}, \frac{2-np}{\sqrt{np(1-p)}}, \cdots, \frac{n-np}{\sqrt{np(1-p)}}\}$. We further define the i^{th} element in this set to be $y_j = \frac{j-np}{\sqrt{np(1-p)}}$, $j = 1, 2, \cdots, n$.

For each $X_i \sim B_i$, let k be the value for X_i that is the closest to $E(X_i)$. Let μ_i be the mean of the Normal to which we want X_i to converge in distribution after the transform. If $y_m \in S_Y$ is the closest value to μ_i, we therefore transform $X_i \sim B_i$ to be $Y_i \sim F_i$ such that the support set of Y_i is S_Y and $P(Y_i = y_j) = P(X_i = j + k - m)$. This leads to the following algorithm:

1. Transform $Y_i^{(1)} = \frac{X_i - np_i}{\sqrt{np_i(1-p_i)}}$. (Conduct the standard transformation)

2. Transform $Y_i^{(2)} = \frac{\sqrt{np_i(1-p_i)}}{\sqrt{np(1-p)}} Y_i^{(1)} = \frac{\sqrt{p_i(1-p_i)}}{\sqrt{p(1-p)}} Y_i^{(1)} = \frac{X_i - np_i}{\sqrt{np(1-p)}}$. (Move the support of $Y_i^{(1)}$)

3. Transform $Y_i^{(3)} = Y_i^{(2)} - \frac{np-np_i}{\sqrt{np(1-p)}} = \frac{X_i - np}{\sqrt{np(1-p)}}$. (Make the support to be S_Y)

4. Transform $Y_i^{(4)} = Y_i^{(3)} + \frac{np-np_i}{\sqrt{np(1-p)}} + \tilde{\mu_i}$, where $\tilde{\mu_i}$ is the closest value to μ_i in S_Y. (Change the location)

5. Let the support of $Y_i^{(4)}$ be S'_Y. We construct Y_i by setting its support to be S_Y, and $P(Y_i) = P(Y_i^{(4)})$ for $S_Y \cap S'_Y$, $P(Y_i) = 0$ otherwise. (Make the support to be S_Y after the relocation in Step 4)

In the above algorithm, $Y_i^{(3)}$ replaces the support of $Y_i^{(1)}$ by S_Y. The construction of $Y_i^{(4)}$ together with step 5 corresponds to setting $P(Y_i = y_j) = P(X_i = j+k-m)$ in the above transform. This Y_i is not quite a discrete distribution because $\sum_{\forall y_i} P(y_i) < 1$, but the summation approaches 1 as the size of Y_i's support increases. Since $Y_i^{(4)} = Y_i^{(2)} + \tilde{\mu_i}$, Y_i will converge to $N(\mu_i, \sigma_i)$ in distribution with $n \to \infty$, where $\sigma_i = \sqrt{\frac{p_i(1-p_i)}{p(1-p)}}$.

Although the above transform method can not produce an arbitrary mixture of Normal distribution, we can still control the values of p with each p_i, μ_i, and π_i to obtain different mixtures of Normal distributions. This is adequate for our purposes.

Based on the above transform, our examination is as follows.

1. Let $X^{(1)} \sim \sum_{i=1}^{k_1} \pi_i^{(1)} B_i^{(1)}$ and $X^{(2)} \sim \sum_{i=1}^{k_2} \pi_i^{(2)} B_i^{(2)}$ be the mixture of Binomials for the first and second dimensions respectively.

2. With the setting of μ for each Binomial, use the above transform method to transform $X^{(1)}$ and $X^{(2)}$ to $Y^{(1)}$ and $Y^{(2)}$ respectively using the same values of p and n. $Y^{(1)}$ and $Y^{(2)}$ therefore form a mixture of k_1 times k_2 discrete distributions in the 2-d space.

3. Let $Y = (Y^{(1)}, Y^{(2)})^T$. We define the c.d.f. of Y to be F_n. Using the coupling graph method, we build the high-probability mass cluster tree of F_n and define this tree to be $CT(F_n)$.

4. Let $Z = (Z^{(1)}, Z^{(2)})^T$, where $Y^{(1)} \to Z^{(1)}$ and $Y^{(2)} \to Z^{(2)}$ in distribution. We define the c.d.f. of Z to be F.
5. Construct the high-density cluster tree for F on the support of Y. We denote this tree $CT_Y(F)$.
6. Use our distance measure to calculate $d(CT(F_n), CT_Y(F))$, which is the distance between $CT(F_n)$ and $CT_Y(F)$.
7. Repeat the above steps to obtain $d(CT(F_n), CT_Y(F))$ for increased n.

In the above setup, the support of Y forms a regular lattice. As n becomes larger, each mesh in the lattice becomes smaller.

5.3.2 KNN density estimation

Given a sample drawn from a continuous distribution with c.d.f. F, we can estimate F by the K Nearest Neighbour (KNN) density estimation method. Let x be a point in the support space of F. Let s be a sample of F with size n. By KNN, we have $\widehat{p(x)} = \frac{K}{nV}$, where V is the minimum volumn surrounding x that encompass K points in the sample s.

Let $F_{n,k}$ be the estimator of $F(X)$ by KNN. We have $lim_{n\to\infty,k\to\infty,k/n\to 0}F_{n,k}(x) = F(x)$ for all x. During the process of this convergence as n and k increase, we can keep constructing a discrete distribution with the sample points and their estimated densities. We can also put a regular lattice in the feature space of the original continuous distribution and estimate the density on each vertex of the lattice using KNN on the sample. This setup is similar to our Binomial case. The difference is that in the Binomial case, we have to control only the size of the support of the mixture of Binomials, i.e. the size of the set of vertices in the lattice formed by the support of Binomials. In KNN, we must control the size of the sample as well.

The purpose of the KNN examination is to simulate the convergence of the cluster trees constructed by our coupling graph method under density estimations for samples from a continuous distribution. This has a practical value. For simplicity, we first consider the 2-d cases. Given a continuous distribution in the 2-d space with c.d.f. F, we set up the KNN examination as follows.

1. Draw a sample with size m from F and denote this sample S_m.
2. Put a regular lattice in the feature space with n vertices at each dimension. We define the entire vertex set of the lattice to be V_n.
3. Let $\widehat{F_{n,m}}$ be the c.d.f. formed by estimating the density of each element in V_n using the KNN method.
4. Let $CT(\widehat{F_{n,m}})$ be the high-probability mass cluster tree built by our coupling graph method for $\widehat{F_{n,m}}$.
5. Let $CT_{V_n}(F)$ be the high-density cluster tree of F constructed for the set V_n.
6. Use our distance measure to calculate $d(CT(\widehat{F_{n,m}}), CT_{V_n}(F))$, the distance between $CT(\widehat{F_{n,m}})$ and $CT_{V_n}(F)$.
7. Repeat the above steps to obtain $d(CT(\widehat{F_{n,m}}), CT_{V_n}(F))$ for a sequence of pairs with increasing values of n and m.

We are interested in the trend of $d(CT(\widehat{F_{n,m}}), CT_{V_n}(F))$ as n and m increase.

5.4 An algorithm

Sometimes it is not easy to construct a high density cluster tree for a sample. We now develop an algorithm to numerically construct such a tree. As we explained in Section 4.9, if we put a regular lattice in the support space of a continuous distribution F and assign each vertex say v a mass $f(v)$, where $f(\cdot)$ is the density function of F, we can construct a discrete distribution. Therefore we can construct a probability mass cluster tree using our coupling graph method on the discrete distribution. This leads to the following algorithm.

1. Put a lattice in the support space of F.

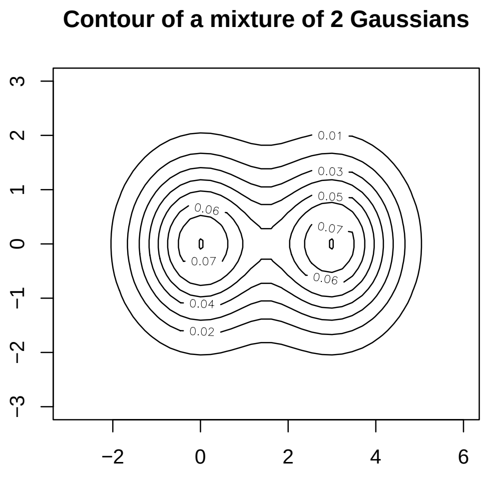
50	100	150	200	250	300	350	400	450	500
0	0	0	0	0	0	0.1429	0.1283	0	0

Table 5.1: Distance from the tree generated by our algorithm to the true tree

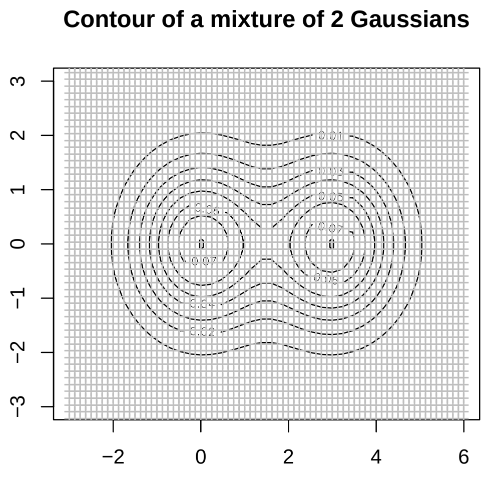
2. Find the vertex set V of the lattice and $f(V)$.
3. Apply the coupling graph method to V and $f(V)$ to construct a cluster tree.
4. For each layer L of the tree and for each point p in the sample S, assign p to a node N in layer L if its nearest neighbour in V, say v_p, is in N. If v_p is not a vertex in any node of layer L, do not assign p to a node in L.
5. Find the cluster tree for the sample by removing all the vertices from the tree.

To check if the outcome from our algorithm is an approximation of the true high-density cluster tree for a sample, we use the following test cases.

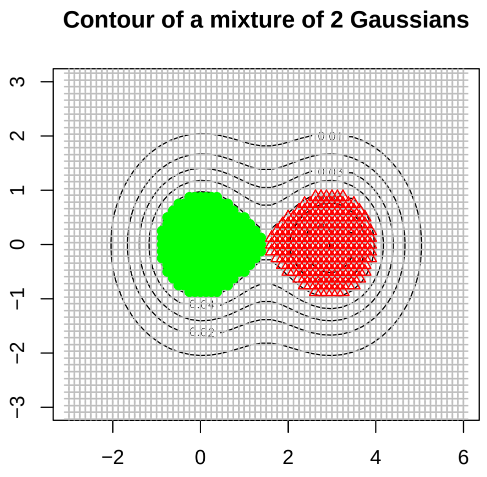
Let F be the c.d.f. of a mixture of two Normal distributions where each has the proportion 0.5 in the mixture. Because of the symmetry of this mixture, we can easily construct the true high-density cluster tree for a sample. We then compare the tree generated from our algorithm to the true tree using our distance measure. Figure 5.1(a) shows the contour of this mixture. Figure 5.1(b) illustrates a lattice in the feature space of this mixture. Figure 5.1(c) gives the coupling graph clustering result on the vertex set from the lattice with their true densities. In Figure 5.1(c), the vertices from the regions of the two different modes are separated exactly.



(a) Mixture of 2 Bivariate Normals



(b) Lattice in the feature space of (a)



(c) Clustering result for the lattice of (b)

Figure 5.1: Test of the algorithm (Case 1)

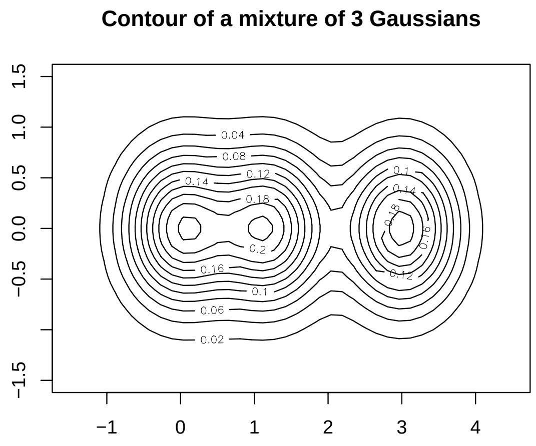
50	100	150	200	250	300	350	400	450	500
0	0.2245	0.0649	0	0	0.0335	0	0.1318	0	0

Table 5.2: Distance from the tree generated by our algorithm to the true tree

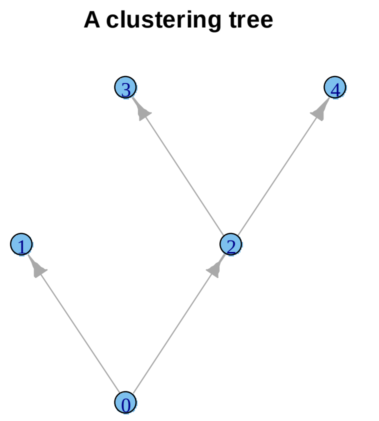
Table 5.1 shows the distance from the tree generated by our algorithm for random samples to the true tree. The first row in this table shows the size of the random samples and the second row shows the distance. The generated tree is exactly identical to the true tree in most cases. Moreover, the differences are very small.

Let F_2 be the c.d.f. of a mixture of three Normal distributions where each has the proportion 1/3 in the mixture. The true high-density cluster tree built for a sample can be obtained by searching the local minimum of the density and the trend of density change in the regions that separate two modes. Figure 5.2 shows how our algorithm works on F_2. The results in Table 5.2 are similar to those in Table 5.1.

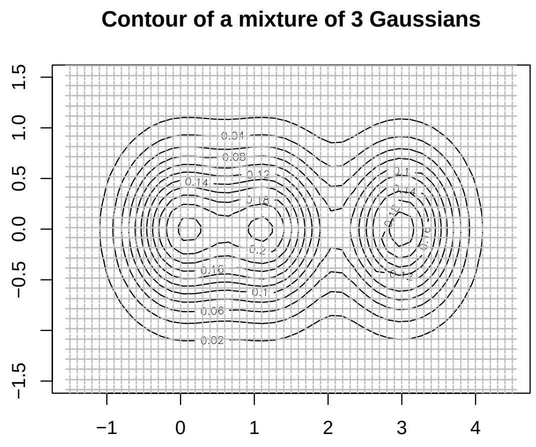
From the above two test cases, the difference between the tree generated by our algorithm and the true tree is so small that it can be ignored. Our algorithm provides a way to numerically construct the high-density cluster tree for a sample.



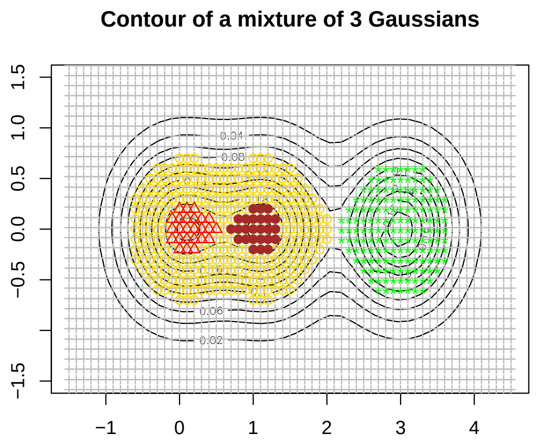
(a) Mixture of three Normals



(b) High-density cluster tree



(c) Lattice in the feature space



(d) Clustering for the lattice

Figure 5.2: Test of the algorithm (Case 2)

Chapter 6

Experiments

6.1 Chapter summary

The previous chapter describes the methodologies for assessing clustering. Using Monte Carlo, the difference between a cluster tree estimate and its estimand can be calculated by our distance measure. Since we obtain cluster tree estimates from clustering methods, our distance measure can be used to evaluate clustering performance. We have discussed methodologies to empirically examine the convergence of cluster trees. In this chapter, we implement these methodologies.

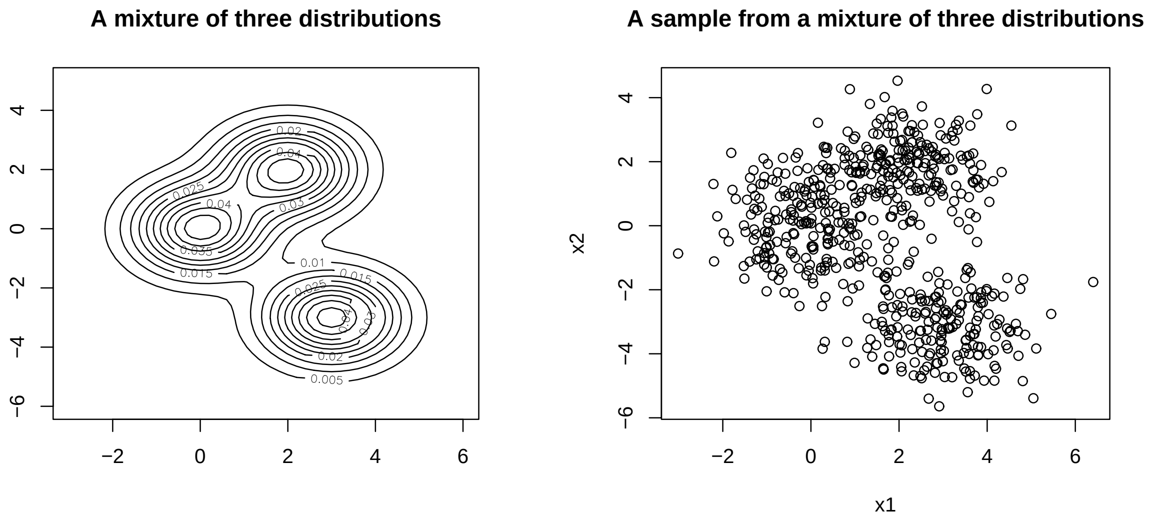
We also present experiments for applications of the graph family framework, such as cluster tree averaging and bagging. The clustering methods used are listed in Appendix E.

6.2 Performance comparison

Based on the methodologies described in Section 5.2, we run the first two experiments in which random samples of different sizes are drawn from a continuous distribution with multiple modes. In the second two experiments, we draw samples from discrete distributions. Cluster tree estimates are constructed and the difference between each estimate and the corresponding high-density or probability mass cluster tree is calculated.

6.2.1 Experiment 1

Figure 6.1 shows the distribution used in this experiment. The distribution is a



(a) Contour of a mixture (b) Sample from (a)

Figure 6.1: Contour and sample of a distribution

mixture containing three centre-based Gaussians. Figures 6.2 and 6.3 show the experimental results. Note that the clustering methods we used for the experiments in this section are listed in Appendix E. Appendix E also lists the parameter settings for these methods.

Figure 6.2 shows the average of the distances from the tree constructed by each method to the high-density cluster tree for samples drawn from the mixture shown in Fig. 6.1 with sizes from 100 to 800. In this figure, the average from DBSCAN shows a quadratic shape. DBSCAN performs best with middle-sized samples. When the sample size is small, there is not enough information for DBSCAN to capture the patterns of the density and many of the sample points are considered noise. When the sample size is large, with $Minpts$ fixed at 4 as suggested by the original DBSCAN researchers, too many points are regarded as high-density points and this misleads the patterns of the density. This result implies that the $Minpts$ of DBSCAN should be adjusted for relatively large samples.

In Fig. 6.2, runt pruning and the KNN coupling graph method have the overall best performance. The KNN coupling graph is the only method showing a decreasing trend in its distance to the true tree. The Gaussian methods, k-means partition, and linkage clusterings (except for single linkage) demonstrate a stable distance to the true tree as sample size varies. Single linkage tends to have an increasing distance as sample size increases. A cluster with 10 points is not trivial for samples of size 100, but it is trivial for samples of size more than 500. The tendency to find trivial clusters affects the performance of single linkage, especially in larger samples. Because it applies a pruning process to ensure that the clusters have reasonable sizes, runt pruning performs much better than single linkage.

K-means is good for finding centre-based clusters. However, it shows an average performance for the samples from a mixture of three Gaussians. K-varying and k-fixed are the estimators generated from partitions of k-means using our method of partition integration. They have better performance than k-means in most of the samples.

Figure 6.3 shows the boxplots of the distances for each method and each sample size. Here $size = 1$ indicates that the sample size is 100 and so on. Some methods including Gaussian partition, Gaussian tree, k-means, KNN coupling graph, and runt pruning demonstrate a clear trend of reduction in the variation of the estimation error as sample size increases. In the other methods the variation in the estimation error does not depend on the sample size. A decrease in variation does not imply a better performance of the estimator.

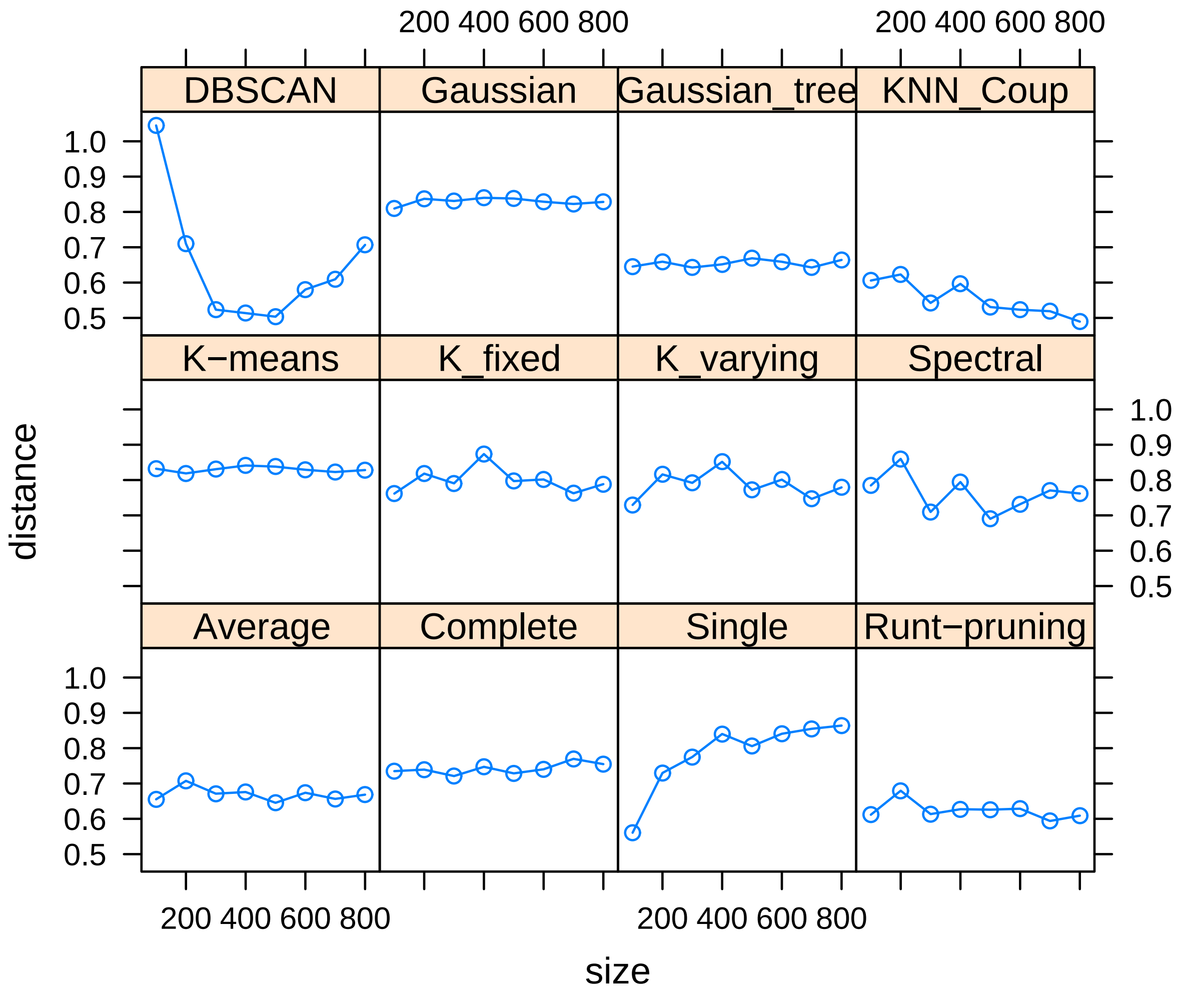


Figure 6.2: Mean of distances versus sample size for each method

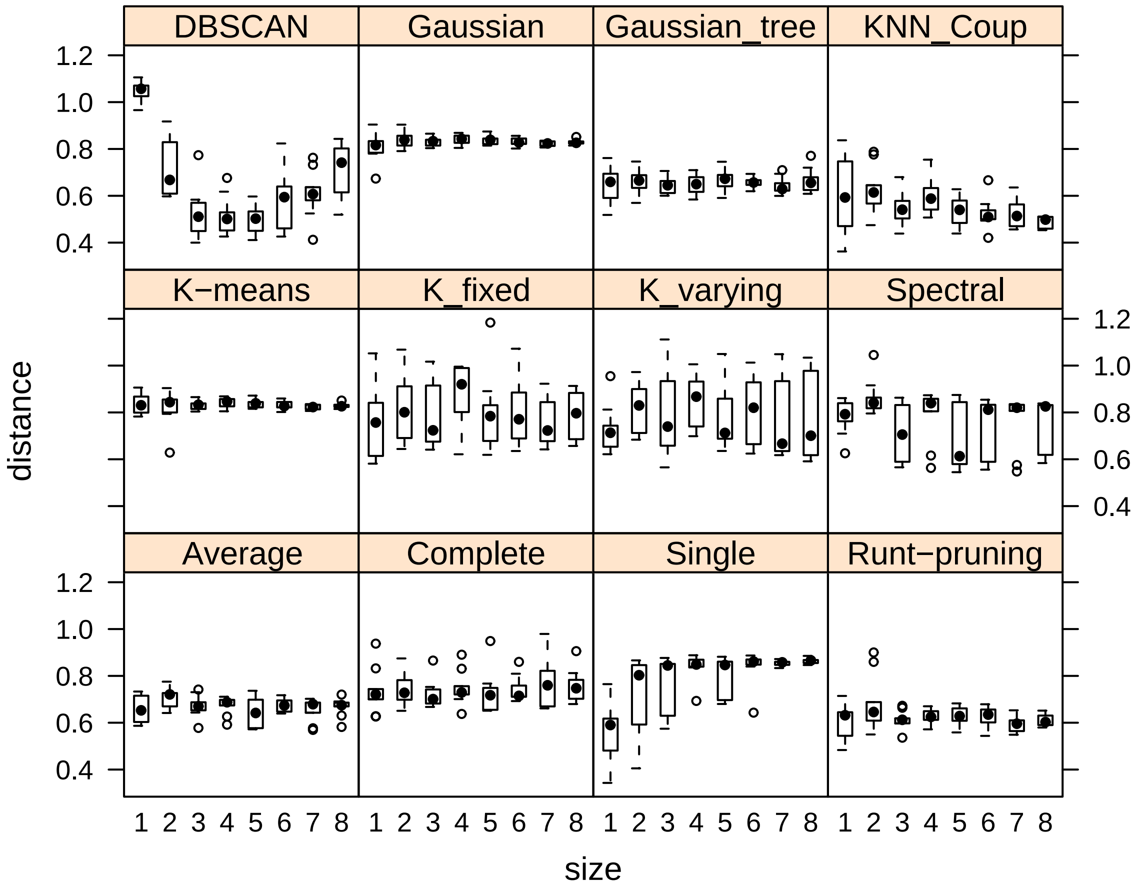
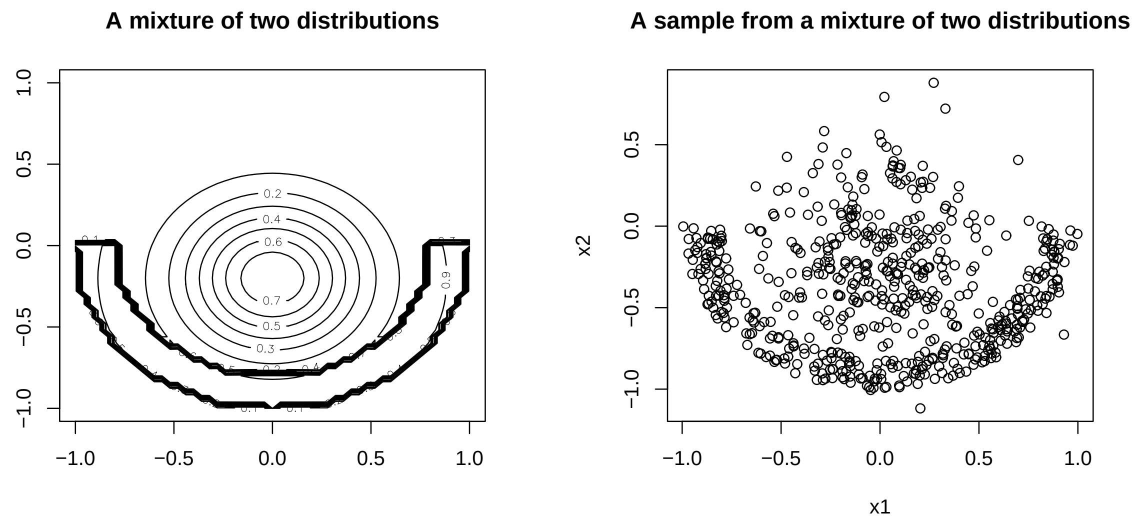


Figure 6.3: Boxplot of distances for each sample size and each method

6.2.2 Experiment 2

Figure 6.4 shows the distribution used in this experiment. The distribution is a



(a) Contour of a mixture (b) Sample from (a)

Figure 6.4: Contour and sample of a distribution

mixture containing a Gaussian and a bow-shaped distribution that does not look like a Gaussian. Figures 6.5 and 6.6 show the results of the experiment with the mixture as shown in Fig. 6.4.

Figure 6.5 shows the average of the distances from the tree constructed by each method to the high-density cluster tree for samples drawn from the mixture shown in Fig. 6.4 with sizes from 100 to 800. In this figure, runt pruning has the best performance. The estimators from two methods, runt pruning and KNN coupling graph, show a convergence trend to the true high-density cluster tree. DBSCAN shows a convergence trend for smaller sample sizes and then remains stable for larger sample sizes. All the other methods generate estimators that do not depend on the sample size and remain around some distance value. K-means with $k = 2$ is the worst method; its estimators have distance around 1, indicating “far away”. K-varying and k-fixed perform better than k-means. The Gaussian partition performance is similar to that of the average linkage. The single linkage performs much worse than runt pruning, which applies a pruning process to single linkage.

Figure 6.6 shows the boxplots of the distances for each method and each sample size. Here $size = 1$ indicates that the sample size is 100 and so on. Some methods including Gaussian partition, k-means, spectral partition, and single linkage, demonstrate a clear trend of reduction in the variation of the estimation error as sample size increases. In some methods, such as DBSCAN and K-varying, the variation in the estimation error does not depend on the sample size. A decrease in variation does not imply a better performance of the estimator.

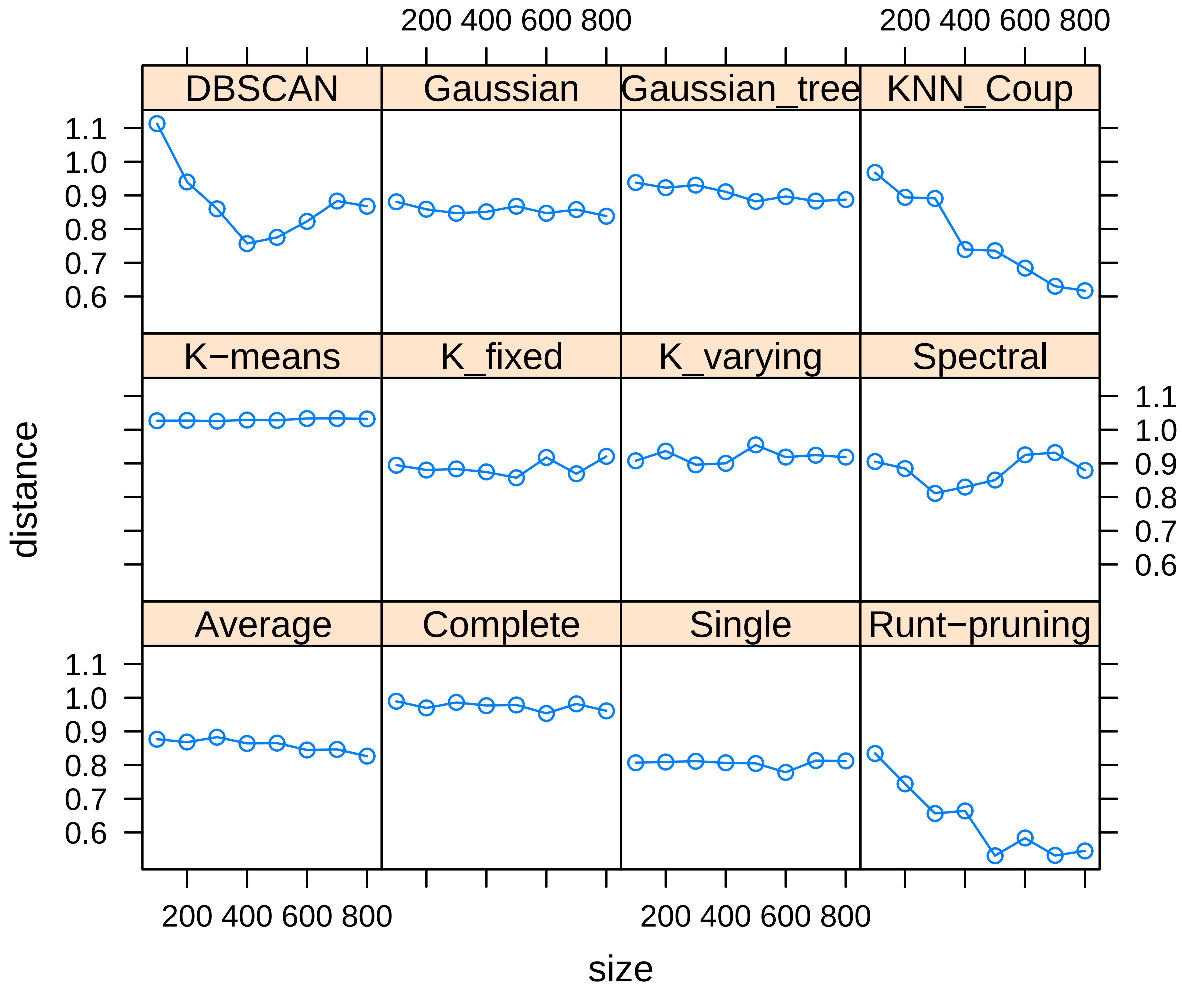


Figure 6.5: Mean of distances versus sample size for each method

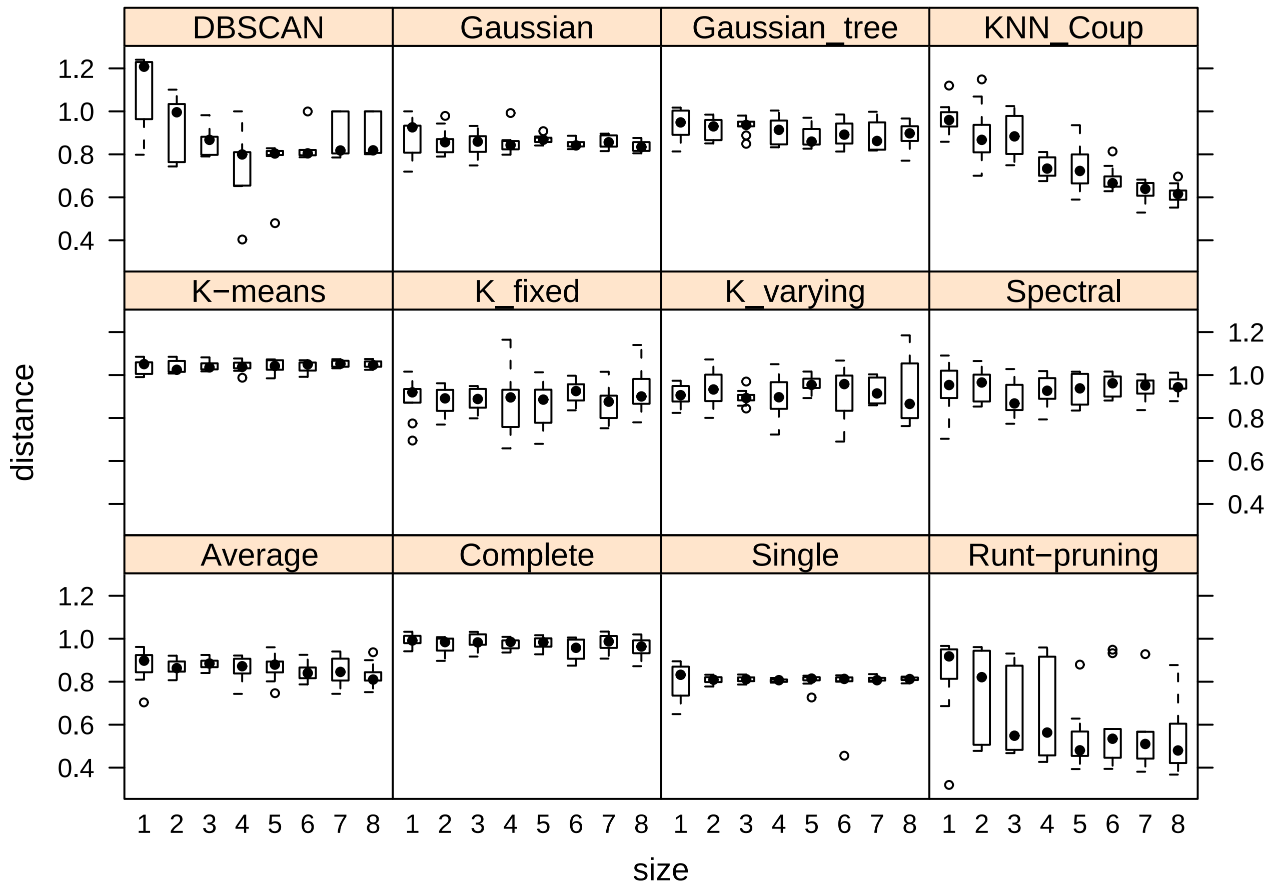


Figure 6.6: Boxplot of distances for each sample size and each method

6.2.3 Experiment 3

This experiment is similar to the first two except that samples used in this experiment are drawn from a discrete distribution shown in Fig. 6.7. This discrete distribution

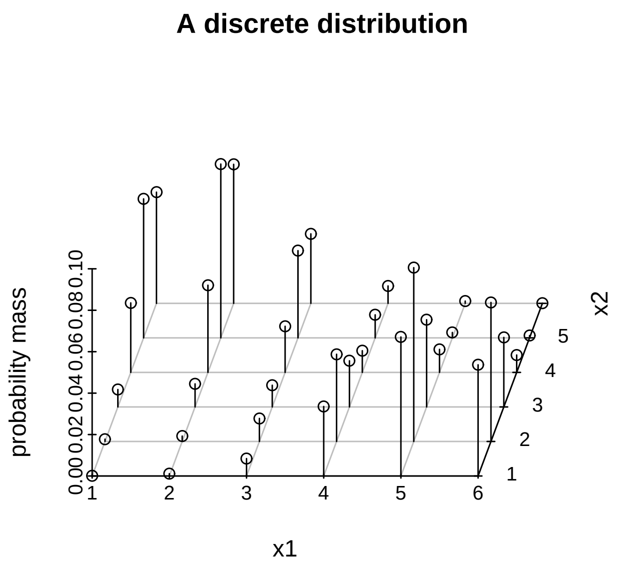


Figure 6.7: A discrete distribution

is exactly the same as the distribution shown in Fig. 4.6 (see Section 4.7). The high-probability mass cluster tree constructed by the coupling graph method is given in Fig 4.7 (see Section 4.7). To make our discussion simple, we denote a tied location in a sample as a location where at least two sample points are located. Note that a sample drawn from a discrete distribution may have tied locations.

Figure 6.8 shows the average of the distances from the tree constructed by each method to the high-probability mass cluster tree for samples with sizes from 100 to 800. Different from the method “KNN-Coup” that is listed in Appendix E, the method denoted by “EMP-Coup” uses the proportion of sample points that are located in a location as the estimated probability mass for that location. The method denoted by "DBSCAN2" uses the maximum number of sample points from a tied location in a sample as $Minpts$ that is a parameter of DBSCAN. The two methods "EMP-Coup" and "DBSCAN2" are only used in this and the following experiments that have samples drawn from a discrete distribution; we do not list them in Appendix E.

In Fig. 6.8, the average of distances from "EMP-Coup" shows a clear decreasing trend as sample size increases. This result is reasonable as the estimation error of the probability mass for each point in the distribution is getting smaller when the sample size is getting larger. The method denoted by "DBSCAN" (using $Minpts = 4$ as suggested by the researchers of DBSCAN) performs the worst among all the methods used in the experiment. Since there are tied locations in each sample and many such locations have more than 4 sample points, using $Minpts = 4$ makes too many sample points to be a core-point (a core-point is a point used to construct a cluster in DBSCAN) and makes DBSCAN find too many small clusters. By adjusting $Minpts$, "DBSCAN2" performs much better in this case. Some methods such as the spectral clustering, k-means, complete and average linkages perform very well in this case. The two methods, k-varying and k-fixed, worsen the performance of k-means because we set $k = 2$ (this is the correct information of number of clusters in this case) for k-means. As we have seen from the previous experiments, by a pruning process, the runt pruning method also improves the performance of single linkage in this experiment. However in this case, samples with larger sizes have more sample points located in tied locations than samples with smaller sizes. This makes the runt pruning method on larger sized samples perform slightly worse than that on smaller sized samples.

Figure 6.9 shows the boxplots of the distances for each method and each sample size. Here $size = 1$ indicates that the sample size is 100 and so on. The variation from "DBSCAN" is very small because the distances from this method are all around 1.2. The variation of distances from some methods such as "DBSCAN2" and runt pruning is relatively large. Some methods including k-means, the spectral clustering, single and complete and average linkages have small variation in their distances. In this figure, there is no other obvious pattern.

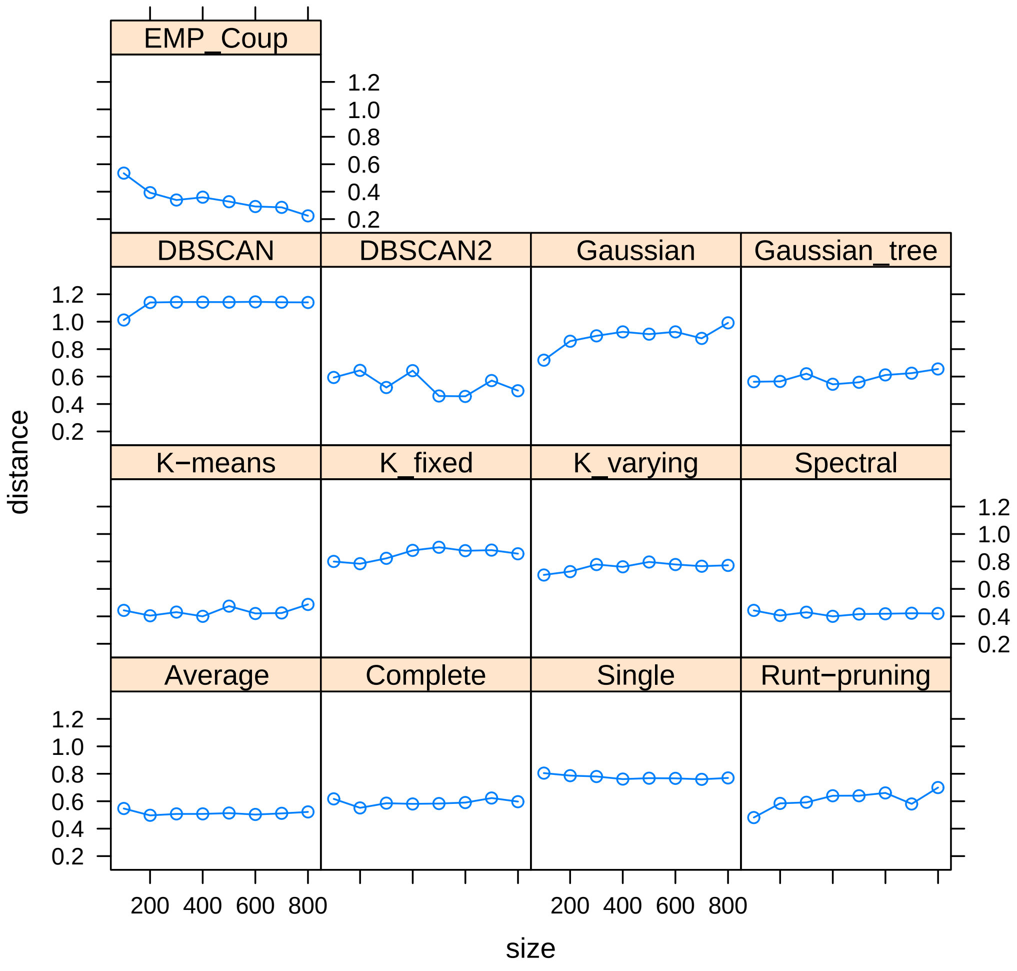


Figure 6.8: Mean of distances versus sample size for each method

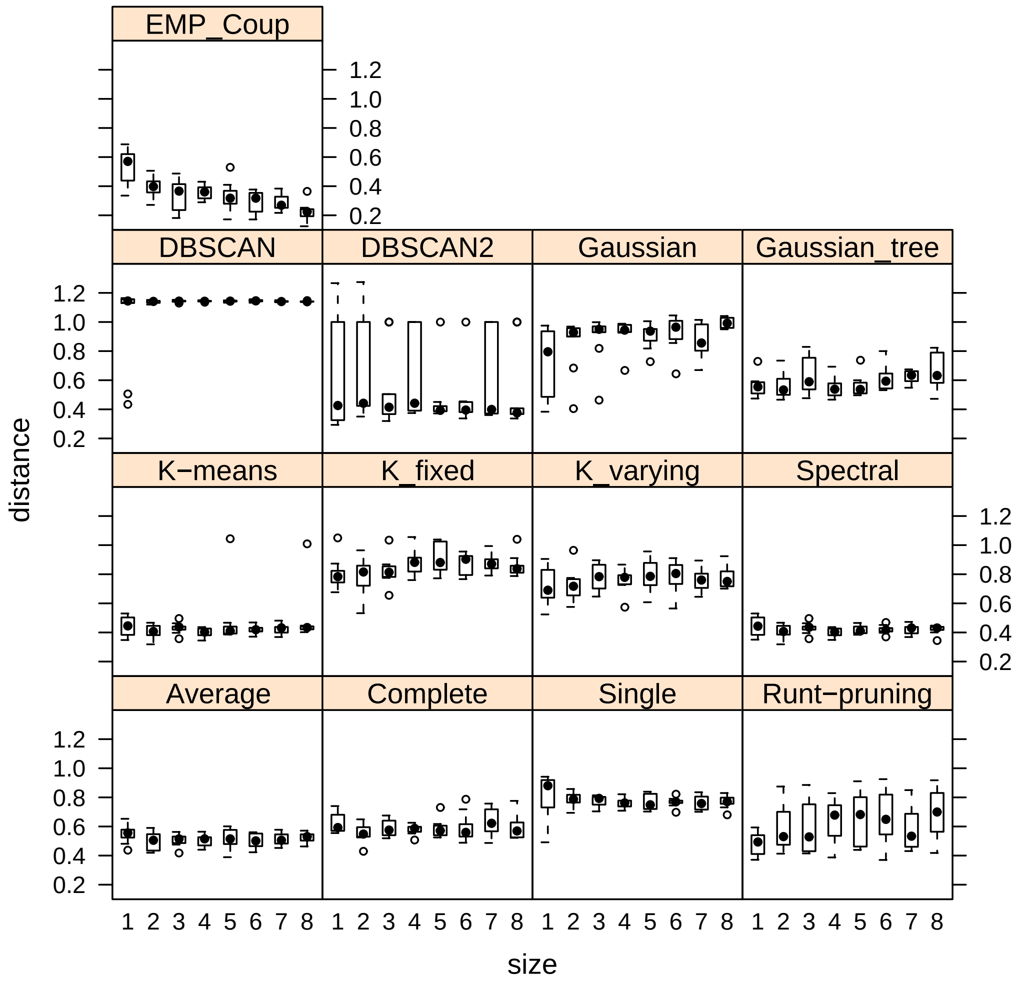


Figure 6.9: Boxplot of distances for each sample size and each method

6.2.4 Experiment 4

The discrete distribution we used for this experiment is shown in Fig. 6.10. This

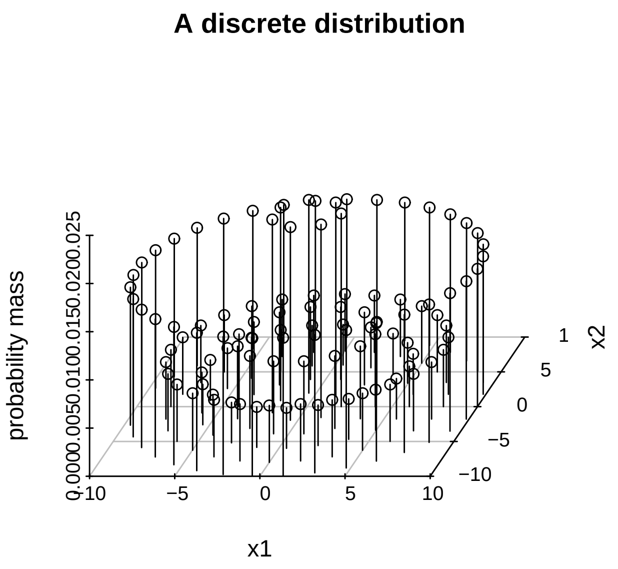


Figure 6.10: A discrete distribution

distribution is similar to that shown in Fig. 4.8 (see Section 4.7). Unlike the distribution used in the previous experiment, the support of this distribution is not a vertex set from a regular lattice. The high-probability mass cluster tree of this distribution constructed by the coupling graph method is a two-layer tree with two child nodes of its root such that one child node contains points from the first two inner circles and the other contains points from the first two outer circles.

Figures 6.11 shows the average of the distances from the tree constructed by each method to the high-probability mass cluster tree for samples drawn from the discrete distribution with sizes from 100 to 800. Similar to the previous experiment, the average of distances from "EMP-Coup" shows a clear decreasing trend as sample size increases. More than half of the methods have their average of distances around 1 (i.e.

"far away") no matter what the sample sizes are. The ability to discover chain-shaped clusters makes single linkage a better method than complete and average linkages. The pruning process also helps the runt pruning method perform better than single linkage in this case. K-means is not good at discover chain-shaped clusters. Based on our method of partition integration, k-varying and k-fixed improve the performance of k-means in this experiment.

Figure 6.12 shows the boxplots of the distances for each method and each sample size. Here $size = 1$ indicates that the sample size is 100 and so on. The variation from k-means, the spectral clustering, complete and average linkages is very small because the distances from these methods are all around 1. There is no obvious pattern from other methods.

Our experiments use different mixtures of distributions. The purpose of these experiments is not to determine which clustering method is the best, since there is no "universally best" method: each method performs well in particular cases. Our purpose is to set up the mechanism for comparing clustering methods for samples drawn from distributions. The experimental results can give us information about the performance of different methods in situations similar to our experiments.

By analyzing the results from all the four experiments in this section, we have the following general conclusions for the clustering methods we used in the experiments.

1) Increasing sample size does not imply the decreasing of estimation error.

2) Increasing sample size does not imply the decreasing of the variation from the estimation error.

3) A small variation of estimation error does not imply a small estimation error.

4) A clustering method is nice if its estimation error is decreasing as sample size increasing.

5) The methods, k-varying and k-fixed, can reduce the estimation error from the original k-means with k to be the correct number of clusters in the sample.

6) Runt-pruning is a good method in terms of the small estimation error.

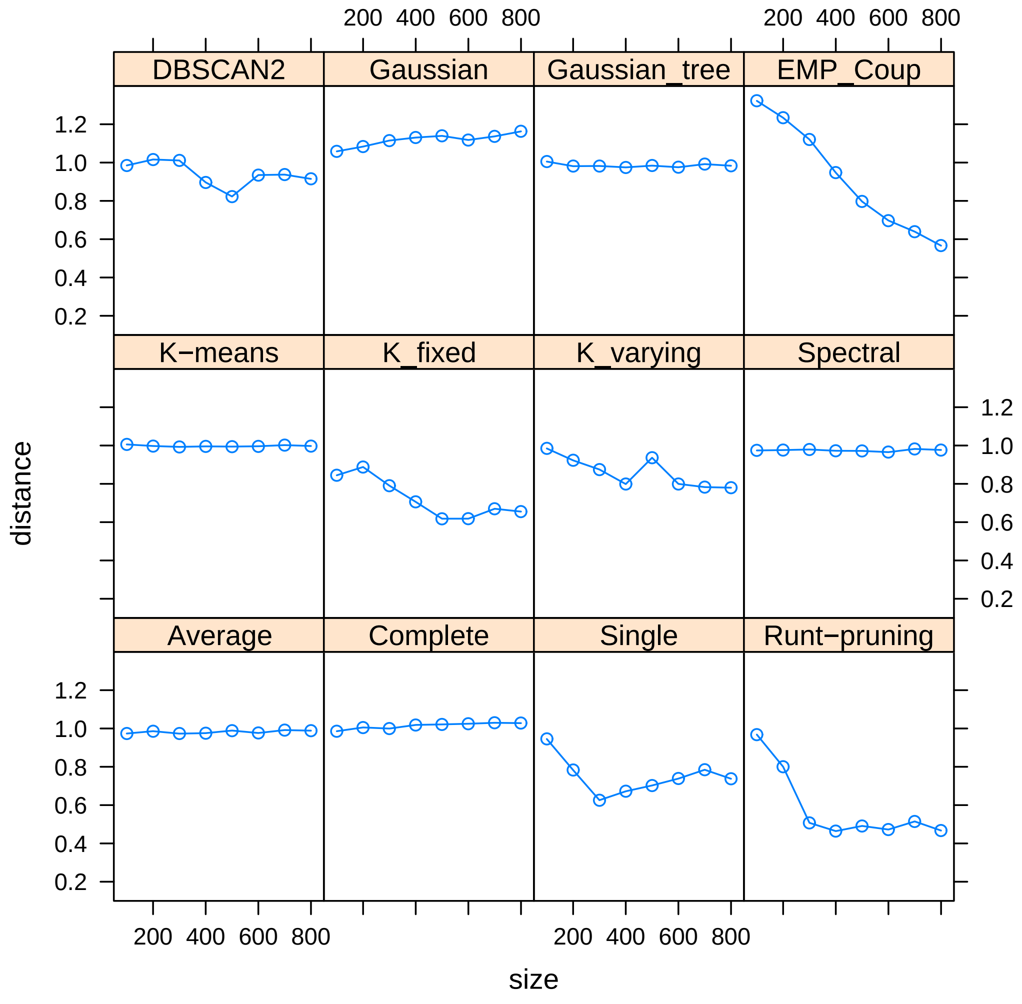


Figure 6.11: Mean of distances versus sample size for each method

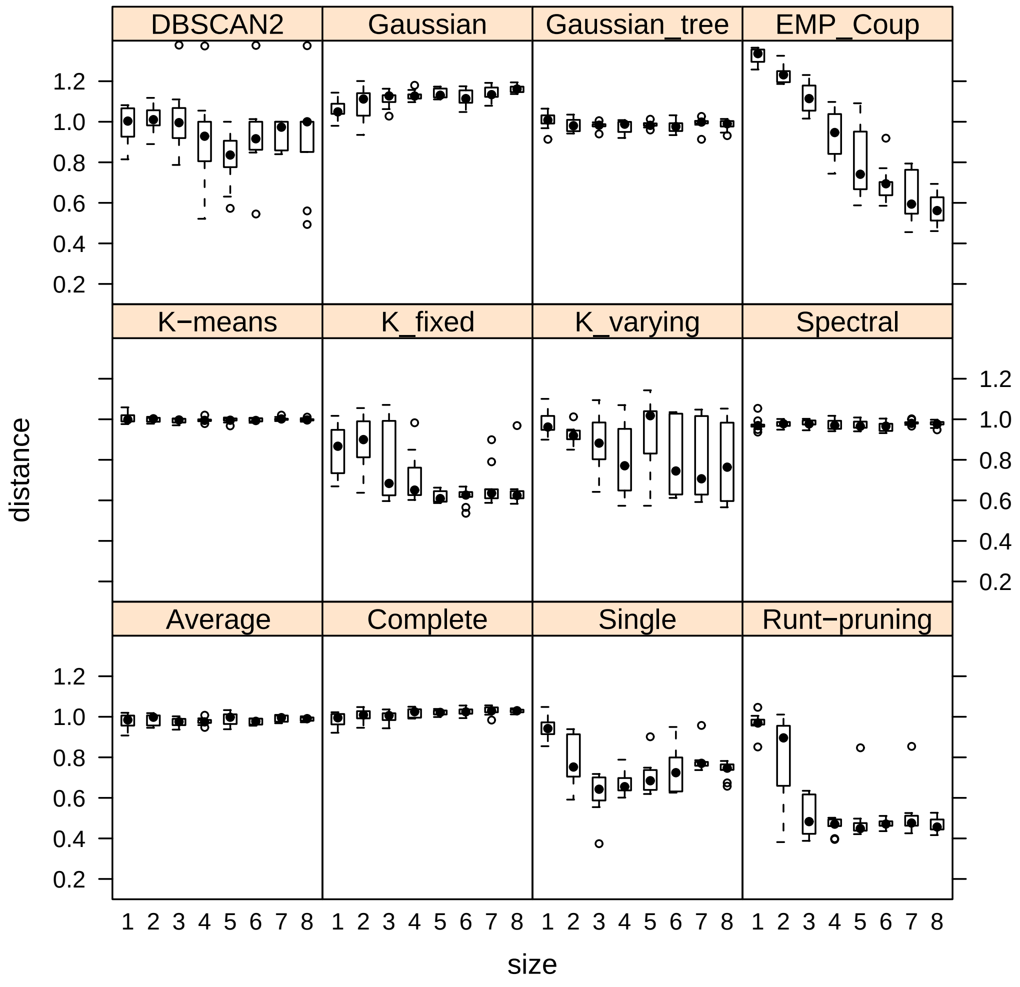


Figure 6.12: Boxplot of distances for each sample size and each method

6.3 An empirical examination of convergence

As described in Section 5.3, we set up experiments to study the convergence of cluster trees.

6.3.1 Binomial and Normal

The following experiment is constructed according to the methodology introduced in Section 5.3.1. Let $X^{(1)} = \sum_{i=1}^{3} \pi_i^{(1)} B_i^{(1)}$, where $B_1^{(1)} = Bin(n, 0.3)$, $B_2^{(1)} = Bin(n, 0.7)$, $B_3^{(1)} = Bin(n, 0.3)$, and $\pi_1^{(1)} = \pi_2^{(1)} = \pi_3^{(1)} = 1/3$. Let $X^{(2)} = B_1^{(2)}$, where $B_1^{(2)} = Bin(n, 0.7)$. Let $\mu_1^{(1)} = 0$, $\mu_2^{(1)} = 4$, $\mu_3^{(1)} = 7$ be the means of the three different Normal distributions to which $B_1^{(1)}$, $B_2^{(1)}$, and $B_3^{(1)}$ will converge in distribution after our transform. Similarly, we set $\mu_1^{(2)} = 0$ for $B_1^{(2)}$.

If we transform X to Y based on the method described in Section 5.3.1, we will have Y converge to Z in distribution, where Z is a mixture of three Bivariate Normal distributions with the same covariate matrix

$$\Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

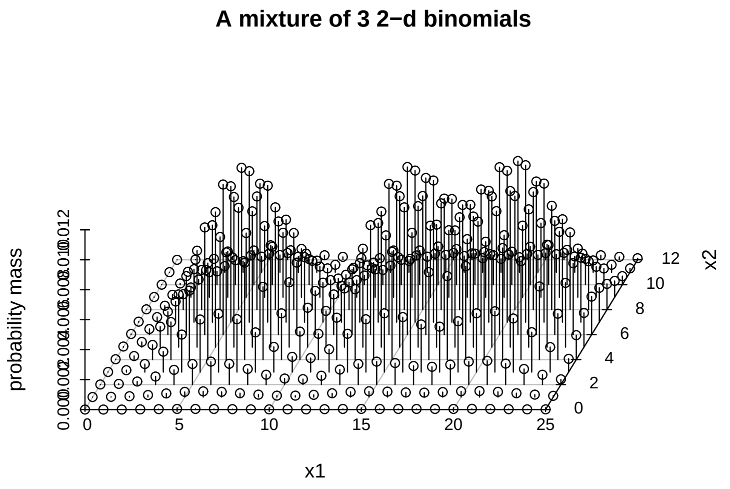
and mean vectors $(0, 0)^T$, $(4, 0)^T$, $(7, 0)^T$ respectively. The proportion of each Bivariate Normal in Z is identical. Figure 6.13 shows a mixture of 2-d Binomials $X = (X^{(1)}, X^{(2)})^T$ with $n = 25$, and a contour plot for Z.

We calculate the distance from $CT(F_n)$ to $CT_Y(F)$, denoted by $d = d(CT(F_n), CT_Y(F))$, for each n, with n increasing in increments of 1 from 5 to 204. The construction of $CT(F_n)$ and $CT_Y(F)$ were introduced in Section 5.3.1.

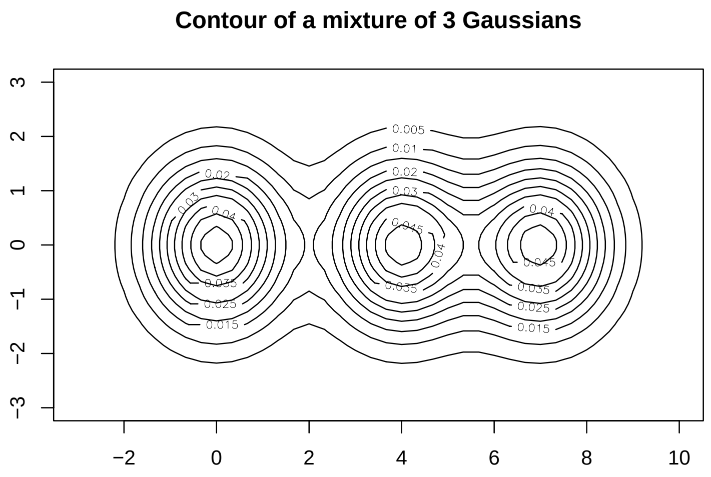
Figure 6.14 shows a scatter plot of these 200 distances with the Loess smoothing curve. From the Loess curve, we can see a trend of decreasing distance with increasing n. We can also observe that the distance goes up and down locally with increasing n. This is because, when n is not large enough, the shape of the Binomial mass function is not close to the density function of the Normal, and the support set of Y changes as n increases. Therefore, if there are points in the current support set with locations in the region with lower density separating the two modes, these points may be assigned to incorrect clusters. Actually, as n increases, the points of the support set jump in and out of the critical regions that have lower densities and separate the different modes.

To check the significance of the decreasing trend, we fit a simple linear model, $d = \alpha + \beta n + \epsilon$. The fitted model is $\hat{d} = 0.389 - 0.0011n$, with the p-values for both α and β less than $2(10^{-16})$. The fitting of this model testifies to the significance of the trend. The decreasing trend is much more dramatic with small n than larger n. To reduce this effect from small n, we further fit the same model with n starting from 50. The result is the same.

The above Monte Carlo experiment gives us an explicit sense of the convergence we are interested in.



(a) Mixture of 2-d Binomials



(b) Mixture of Bivariate Normals

Figure 6.13: Binomial and its corresponding Normal

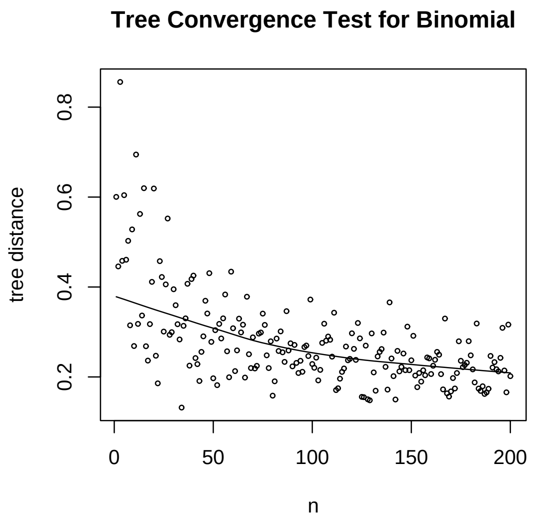


Figure 6.14: Tree distance for the convergence of Binomial to Normal

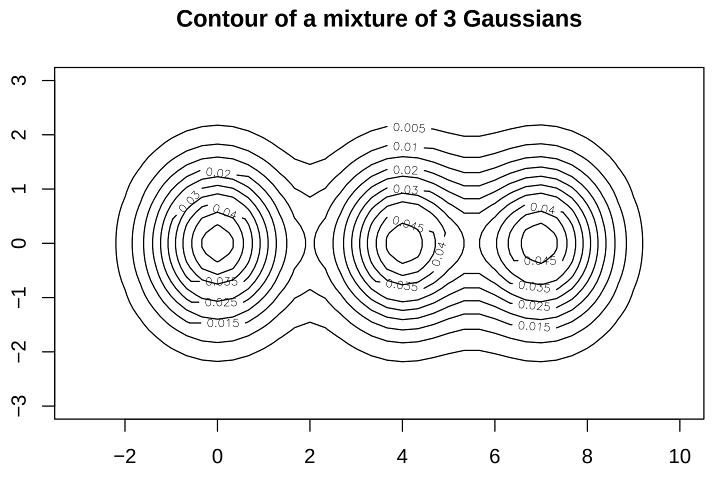
6.3.2 KNN density estimation

In this experiment, we use the mixture of Bivariate Normal distributions shown in Fig. 6.13(b) as the continuous distribution F. With the setup described in Section 5.3.2, we increase the size of the vertices on the lattice from 5 by 5 to 70 by 70. Note that, we say a lattice size instead of a size of the vertices on the lattice for simplicity. In our experiment, a larger lattice size implies a smaller mesh size of the lattice. For each lattice size, we draw samples from F with sample sizes from 300 to 6000. The distance of $CT(\widehat{F_{n,m}})$ to $CT_{V_n}(F)$ is calculated for each pair of n and m, where n is the square root of the total number of vertices in the lattice and m is the sample size. The construction of $CT(\widehat{F_{n,m}})$ and $CT_{V_n}(F)$ were introduced in Section 5.3.2. We denote the distance between these two trees by $d = d(CT(\widehat{F_{n,m}}), CT_{V_n}(F))$.

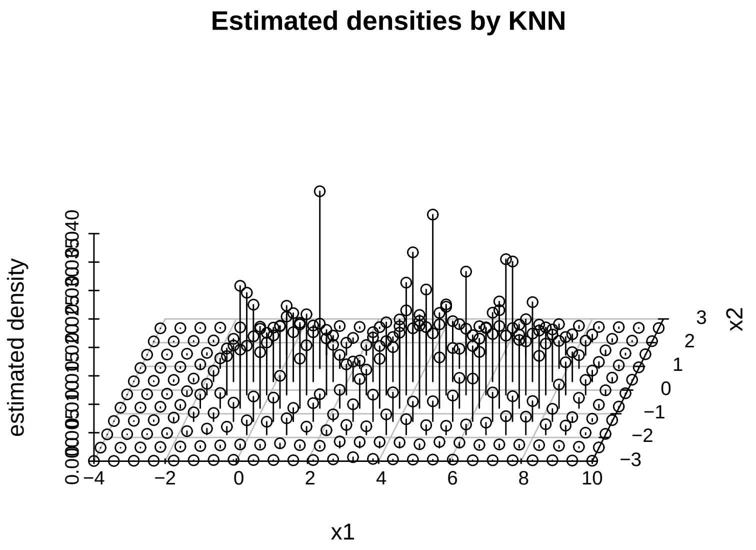
In this experiment, n is increased in increments of 5 from 5 to 70, and m is increased in increments of 300 from 300 to 6000. Figure 6.15 shows the contour of F and estimated densities on a lattice of size 25 by 25, from a sample of F of size 3000.

Figure 6.16 (a) shows the surface formed by the distances on m as sample size and n as lattice size. From this plot, we can observe the trend of decreasing distance with increasing m and n. We also observe that the surface is not smooth. This is caused by the sample error and the density estimation error. Figure 6.16 (b) and (c) show the trend of decreasing distance with an increasing lattice size and an increasing sample size respectively.

To check the significance of this trend, we fit a simple linear model: $d = \alpha + \beta_1 m + \beta_2 n + \epsilon$. The fitted model is $\hat{d} = 0.85 - 1.45(10^{-5})m - 2.99(10^{-3})n$ with the p-values for both α and β_2 less than $2(10^{-16})$ and the p-value for β_1 equal to 0.000192. By further exploration, we find that there is no cross-effect from m and n. The results from the model show the significance of the trend of the convergence from $CT(\widehat{F_{n,m}})$ to $CT_{V_n}(F)$.

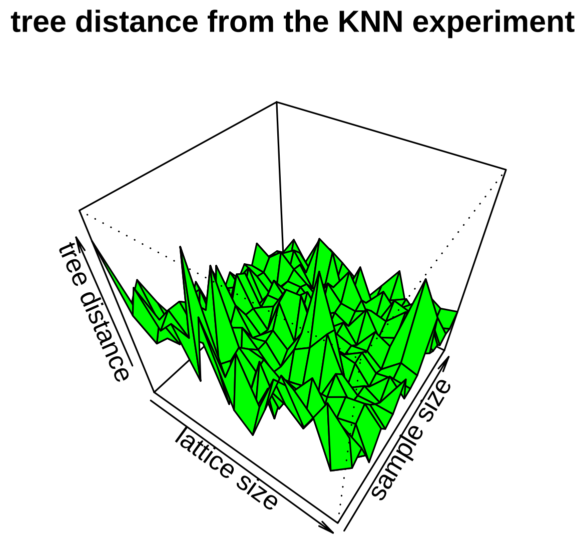


(a) Mixture of Bivariate Normals

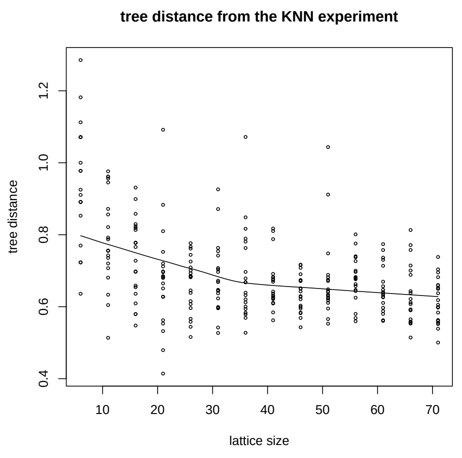


(b) Estimated densities by KNN for a sample from (a)

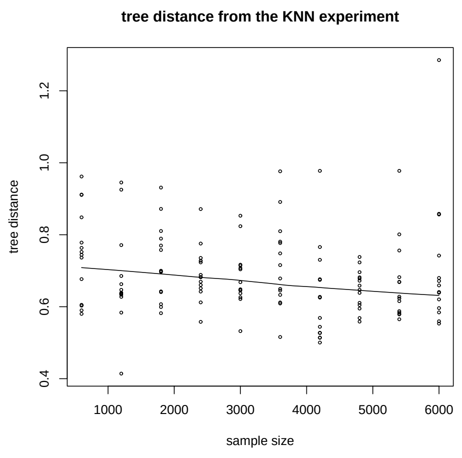
Figure 6.15: A mixture of Bivariate Normals and densities estimated by KNN



(a) A 3-d view



(b) Tree distance versus lattice size



(c) Tree distance versus sample size

Figure 6.16: Tree distance for the convergence of the KNN method to Normal

6.4 Cluster tree averaging and bagging

We introduced cluster tree averaging and bagging in Section 3.3. We now set up experiments to see how they work in clustering and to get a sense of their properties.

6.4.1 Cluster tree averaging

Let $\widehat{CT_{i,S_n}}(F)$ be a cluster tree constructed by a clustering method, say M_i, for a sample of size n drawn from F. We can write the cluster tree averaging as $\frac{1}{p}\sum_{i=1}^{p}\widehat{CT_{i,S_n}}(F)$, where p is the total number of cluster trees in this averaging. We regard this cluster tree average as a new method for estimating $CT(F)$.

To be more specific, let G_i be the weighted graph constructed from $\widehat{CT_{i,S_n}}(F)$. Let W_i be the vector of edge weights from G_i. Let $W_{ave} = \frac{1}{p}\sum_{i=1}^{p} W_i$, and G_{ave} be the weighted graph with the vector of edge weights being W_{ave}. The cluster tree constructed from G_{ave} is our cluster tree average.

In the first experiment, we draw thirty independent random sample sets from the mixture shown in Fig. 6.1 with size 100 for each set. We use five different clustering methods to construct a cluster tree for each sample. For each cluster tree for each sample S, we calculate its distance to the high-density cluster tree for S. We further plot a boxplot from all thirty distances for each method. The five clustering methods are: single linkage, complete linkage, averaging linkage, DBSCAN, and Gaussian-based partition.

We also construct four cluster tree averages with different combinations of the five methods. In Fig. 6.17, above the boxplots of the five clustering methods, tree-ave1 is the average of single and complete linkage. The average is constructed for each pair of cluster trees obtained by single linkage and complete linkage for the same sample set. We therefore have thirty such averages for tree-ave1. Since each average is a cluster tree, we calculate its distance to the high-density cluster tree for the corresponding sample and draw the boxplot of these distances. Tree-ave2 is the average of the three linkage methods; tree-ave3 is the average of DBSCAN and Gaussian-based partition; tree-ave4 is the average of all five methods.

By comparing the boxplots, we find that the average of single linkage and complete linkage has a smaller variation and its median is close to that of single linkage, which has a better performance than complete linkage. The average of the three linkages has a symmetric distribution of distances; its variation is also small but slightly larger than that of tree-ave1. The average of DBSCAN and Gaussian-based partition has a better performance than these two methods individually. The average also has a much smaller variation than DBSCAN. The average of the five methods also has a symmetric distribution of distances. Its variation is larger than that of tree-ave2. The median of tree-ave4 is the lowest among all the averages and close to that of single linkage, which has the best performance among the five methods.

From these boxplots, we find that the cluster tree average tends to reduce the variation of each estimator inside the average. The performance of the average tends to be close to or even better than the best estimator inside the average. However, if the average contains too many different methods, the variation will increase because the variety of these methods tends to be large.

Figure 6.18 shows a similar experiment using the mixture of Fig. 6.4, which is not a mixture of Gaussians. The boxplots lead to the same conclusion for the cluster tree average.

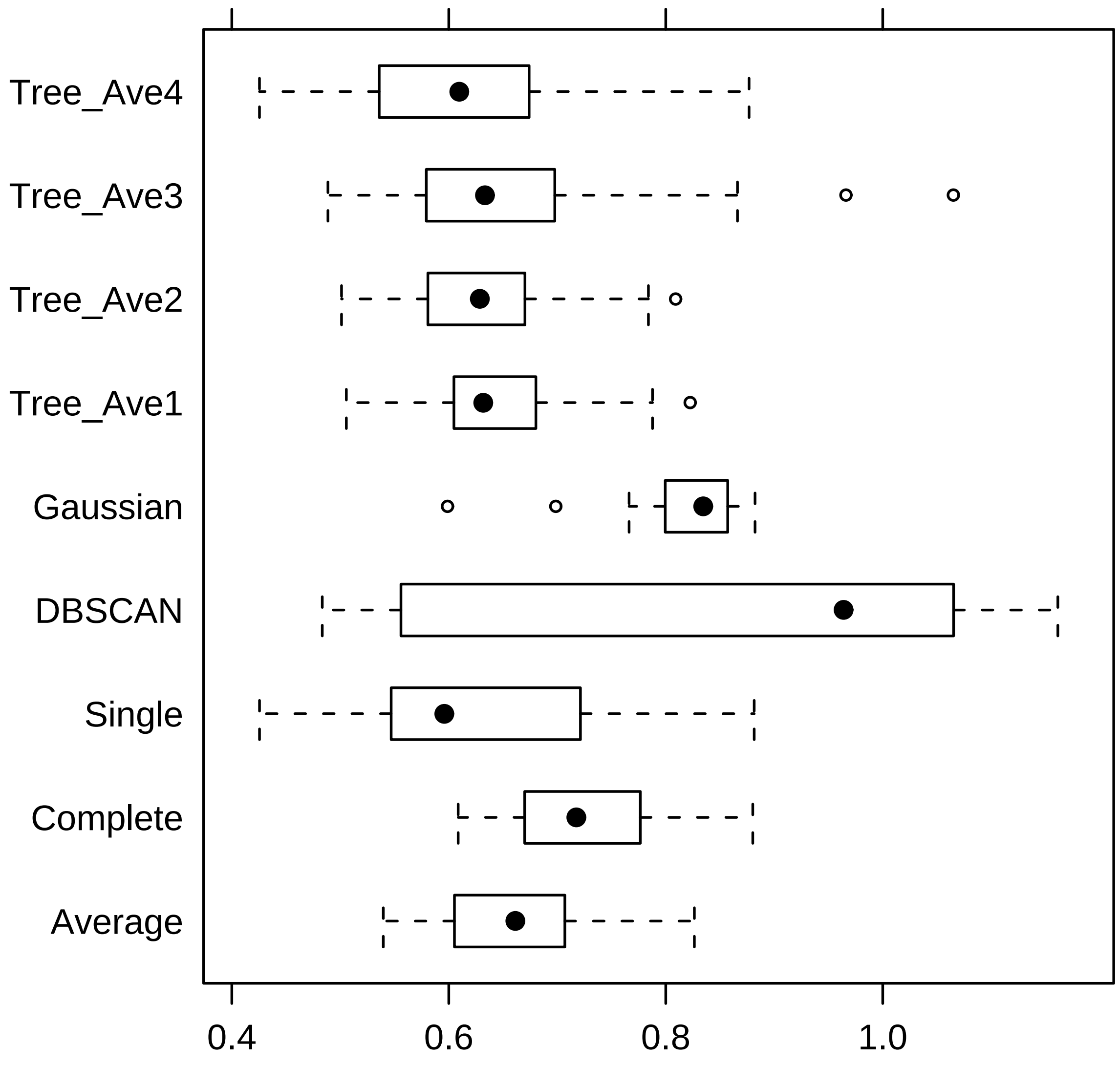


Figure 6.17: Experiment 1 on cluster tree averaging

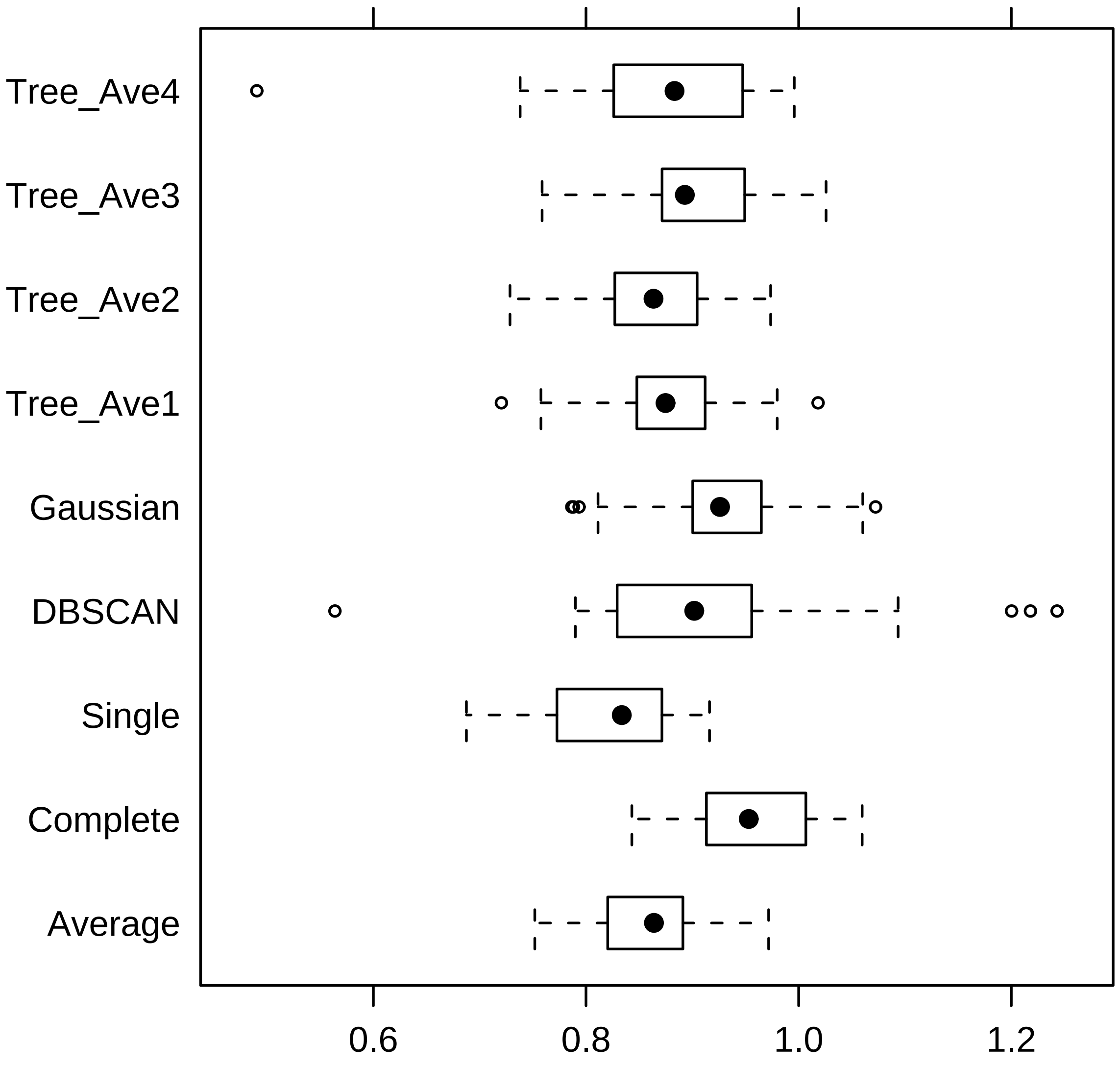


Figure 6.18: Experiment 2 on cluster tree averaging

6.4.2 Cluster tree bagging

The cluster tree bagging can be written as $\frac{1}{B}\sum_{i=1}^{B}\widehat{CT_{j,S^{*(i)}}}(F)$, where B is the number of bootstrapped samples drawn from the original sample S_n, which is drawn from a distribution F, and $\widehat{CT_{j,S^{*(i)}}}(F)$ is the cluster tree constructed by a clustering method M_j on the i^{th} bootstrapped sample $S^{*(i)}$.

Again, the Monte Carlo method helps us to understand the performance of cluster tree bagging. Figure 6.19 illustrates an experiment based on samples drawn from the mixture shown in Fig. 6.1. In this experiment, we draw 25 independent random sample sets of size 100 from the mixture. The clustering methods used for this experiment are listed in Appendix E. For each method we construct a cluster tree for each sample set, and calculate their distances to the corresponding high-density cluster tree for each sample set. The boxplot labeled "sample" for each method in Fig. 6.19 is constructed from 25 such distances for 25 sample sets.

For each sample set, say S_n, we found 30 bootstrapped sample sets. Using one set, S_n, for each method, we constructed 30 cluster trees for these bootstrapped sample sets. The cluster tree bagging constructs the average of these 30 trees. Since this average is a cluster tree as well, we calculate the distance from this average to the high-density cluster tree for S_n. In this way, for each method we get 25 distances from the cluster tree bagging for the original sample sets. The boxplot labeled "bagging" for each method in Fig. 6.19 is obtained from these 25 distances. Comparing the distances from the cluster trees constructed for the original samples to the high-density cluster tree shows that the cluster tree bagging reduces the distances for every method except single linkage and hierarchical Gaussian-based clustering.

Figure 6.20 illustrates a similar experiment but using the mixture shown in Fig. 6.4. In this case, the distance reduction by the clustering bagging is still obvious for some methods such as DBSCAN, complete linkage, and the coupling graph method. However, other methods have no such effect.

The variation in the estimation errors as an estimator of the high-density cluster tree is reduced for most of the methods for both mixtures. This reduction is remarkable for DBSCAN in Fig. 6.19.

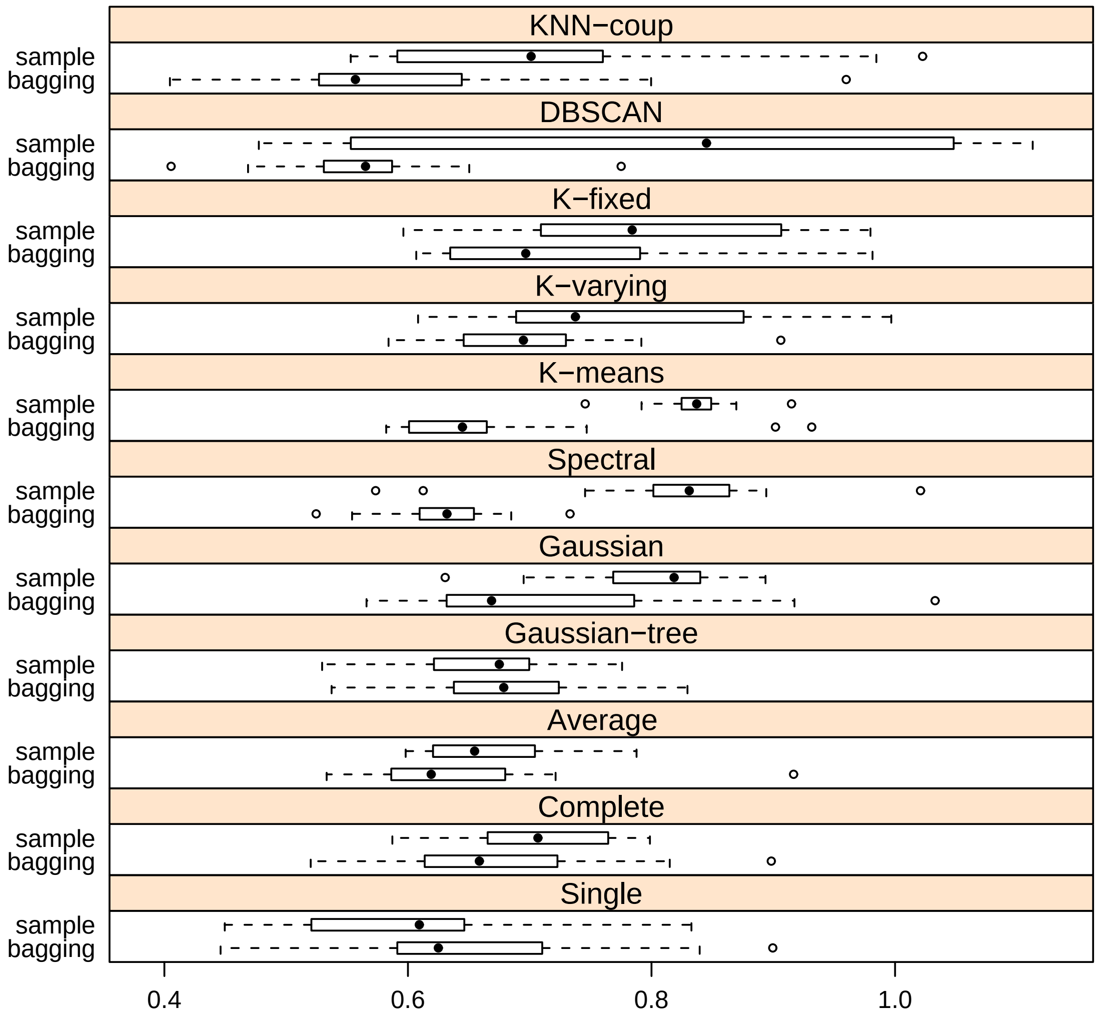


Figure 6.19: Experiment 1 on cluster tree bagging

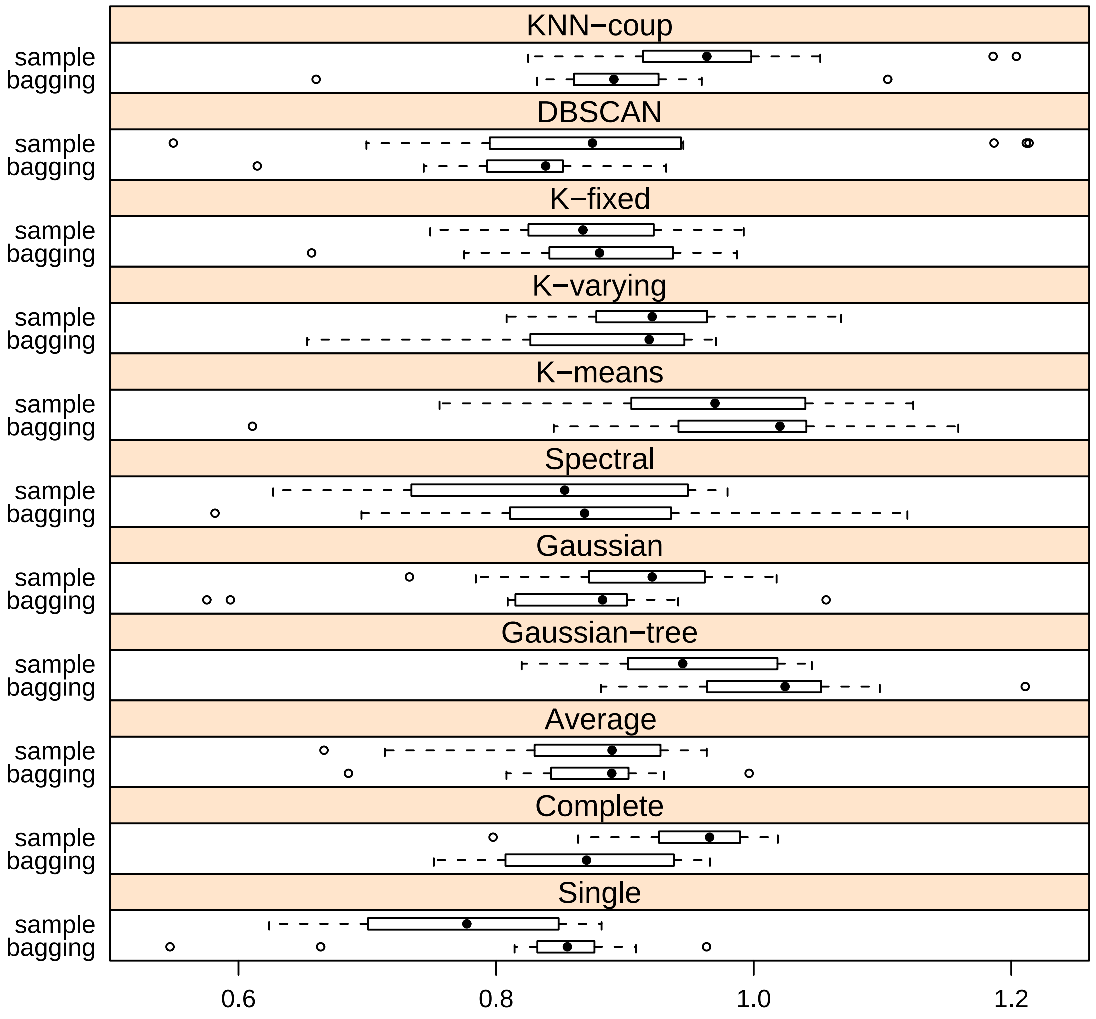


Figure 6.20: Experiment 2 on cluster tree bagging

These two experiments show that cluster tree bagging can reduce the variation in the estimation error in many cases.

Chapter 7

Applications to real data sets

7.1 Chapter summary

We have discussed methodologies for statistical assessment using the Monte Carlo method based on our clustering framework. We have also proposed new methods, such as cluster tree averaging and bagging, as the estimators of the high-probability density/mass cluster tree of a distribution. In many real cases, we observe a sample without knowing the true underlying distribution. In this chapter, we demonstrate possible applications of our framework to real data sets.

7.2 Enron email data

The Enron email data contains email messages exchanged among 184 email addresses over 1316 days. We obtain a 184 by 184 matrix with each element f_{ij} being the number of email messages sent from email address i to j. This is an asymmetric matrix and we call it the frequency matrix M_f. We further get a 184 by 184 distance matrix M_d with each element $d_{ij} = \frac{1316}{1+f_{ij}+f_{ji}}$. M_d is a symmetric matrix and can be used as a distance or a dissimilarity matrix for the email addresses [1]. We use M_d as

[1]Matrices M_f and M_d were originally constructed by Dr. Hugh Chipman, Department of Mathematics and Statistics at Acadia University.

our source data.

We first apply linkage clusterings, such as single linkage, complete linkage, and average linkage, to M_d. We then use DBSCAN with different parameter settings (Minpts is fixed at 4, and ϵ varies). DBSCAN with each set of parameters can produce a partition of the email addresses. (Actually, the union of all the disjoint subsets in the partition from DBSCAN is a subset of the 184 email addresses; this is because DBSCAN classifies some sample points as noise.) We can build a cluster tree from this sequence of partitions through our method of partition integration.

To illustrate the clustering results, we derive a new type of plot, called a cluster-ordering plot. To draw this plot, we first order the email addresses as follows:

1. Set up an ordering list for all 184 email addresses, and initialize the order as the sample index.

2. Let L be the largest layer in a cluster tree. For layers 2 to L perform the following steps (the root is in layer 1).

3. Find the union of all the content in every node of the current layer, say layer i. If there are m nodes in layer i, the union is written as $U_i = \cup_{j=1}^{m} Content(N_{ij})$, where N_{ij} is the j^{th} node in layer i and $Content(N_{ij})$ is the email addresses contained in N_{ij}.

4. Order all the elements (email addresses) in U_i by their associated node id in layer i.

5. Find all the locations for every element of U_i in the ordering list, and reorder these locations by the order obtained in step 4. Update the ordering list by assigning the index of every element in U_i according to its ordered location.

After obtaining the ordering list of the email addresses via the above algorithm, we reorder the matrix M_d to M'_d with the order of both rows and columns being identical to the ordering list. We then display M'_d as a lattice with the brightness of the colour of the ij^{th} square in the lattice determined by the value of the ij^{th} element in M'_d, such that the square is brighter if the associated value is larger.

Any cluster-ordering plot for the Enron email data will have some general patterns as follows. A light-coloured square implies few emails have been sent between the two corresponding employees. A dark-coloured square implies many emails were sent between the two corresponding employees. Since the matrix M_d is symmetric, after the reordering by the above five steps, we have a matrix M'_d that is also symmetric. Therefore the colour of squares in a cluster-ordering plot is symmetric. The colour of the diagonal in a cluster-ordering plot is associated with the frequency of email senders have sent themselves. Since many of the Enron employees copied emails to themselves while sending emails to others, the obvious diagonal line reflects these copies. If there are frequent emails sent among a group of employees and these employees are neighbours in the ordered matrix M'_d, a block of dark-coloured squares will appear on the diagonal of the corresponding cluster-ordering plot at the locations associated with these employees. An off-diagonal block of dark-coloured squares implies frequent emails have been sent between two groups of "neighbour employees". (Note that, "neighbour employees" means these employees associate with neighbour columns and rows in the ordered matrix M'_d.)

From the above general patterns, a good clustering method for the Enron email data should make blocks of dark-coloured squares appear on or close to the diagonal of the cluster-ordering plot.

Our cluster-ordering plot is similar to Hurley's method [35] for clustering visualization. The difference is that Hurley uses m_{ij} to determine the similarity of sample objects, where m_{ij} is called the index of merit between object i and j, and can be calculated by a similarity measure. Hurley also uses clustering methods such as single linkage on the matrix with m_{ij} as the ij^{th} element. The purpose of this clustering is to find pairs of similar variables to be co-displayed in plots such as scatter plots. In our cluster-ordering plot for the Enron email data, we apply clustering to M_d with d_{ij} as its ij^{th} element, and we treat d_{ij} to be just like m_{ij} in Hurley's method. Without displaying email addresses pairwisely, we find the global ordering list of all the email addresses from a cluster tree and display the reordered M_d as a lattice to obtain a general view of the clustering result.

We also apply our method for cluster tree averaging to the cluster trees constructed by single linkage, complete linkage, and DBSCAN.

Figure 7.1 shows the cluster-ordering plot using the original order of the email addresses without reordering by clustering. The dark-coloured squares in the plot are roughly uniformly located. With clustering, it is reasonable to see that the frequency of emails sent within each cluster should be higher than that between different clusters. Reordering these email addresses by clustering, we should observe more blocks of dark-coloured squares along the diagonal than by the original order.

Figure 7.2 shows the cluster-ordering plot with the order of the email addresses determined by single linkage on the distance matrix M_d. The dark-coloured squares are located towards the top right corner of the diagonal. We also see several blocks of dark-coloured squares on the diagonal. Compared to the original plot, this plot shows an obvious pattern of clusters. Figure 7.3 gives the dendrogram for this single-linkage clustering. The clusters at the bottom left corner of the dendrogram are formed by the smallest single-linkage distances, and they represent a close relationship between people inside the same cluster. The cluster tree is quite unbalanced with many small clusters, as is usual for single linkage. From the single linkage dendrogram, for most of the layers, there is a single point joining a big group, these single points form the first part of the ordered email addresses and there is lower frequency of emails among employees in this part. Therefore we see an almost empty region around the bottom left corner and blocks of dark-coloured squares appear at the top right corner in the cluster-ordering plot.

Figures 7.4 and 7.5 show the cluster-ordering plot and the dendrogram constructed using complete linkage on the distance matrix M_d. The complete-linkage dendrogram is still an unbalanced tree. In contrast to single linkage, the blocks of dark-coloured squares are spread out along the diagonal in the cluster-ordering plot. The dendrogram also illustrates this pattern. The difference between the complete linkage dendrogram and that of single linkage is that, a small group instead of just a single point keeps joining a big group in most of the layers. This makes some blocks of dark-coloured squares appear at the bottom left corner of the diagonal of the cluster-ordering plot.

Figure 7.6 shows the cluster-ordering plot of the cluster tree averaging of single

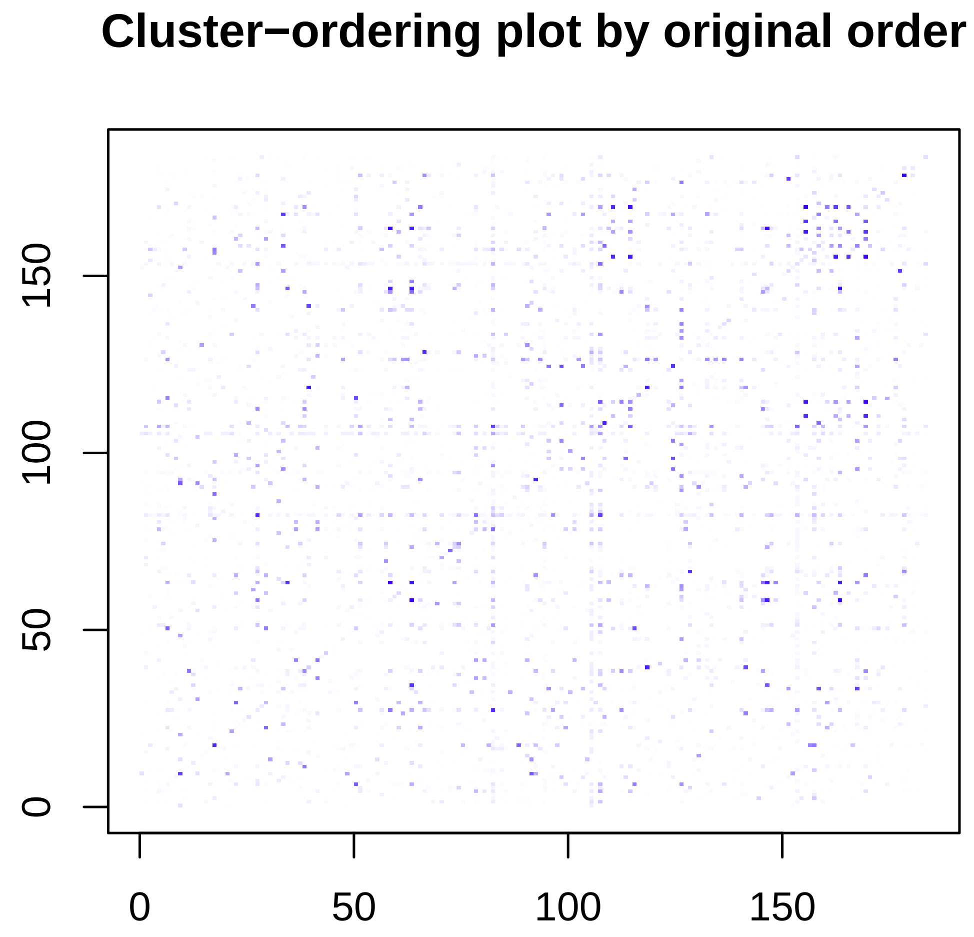


Figure 7.1: The cluster-ordering plot using the original order

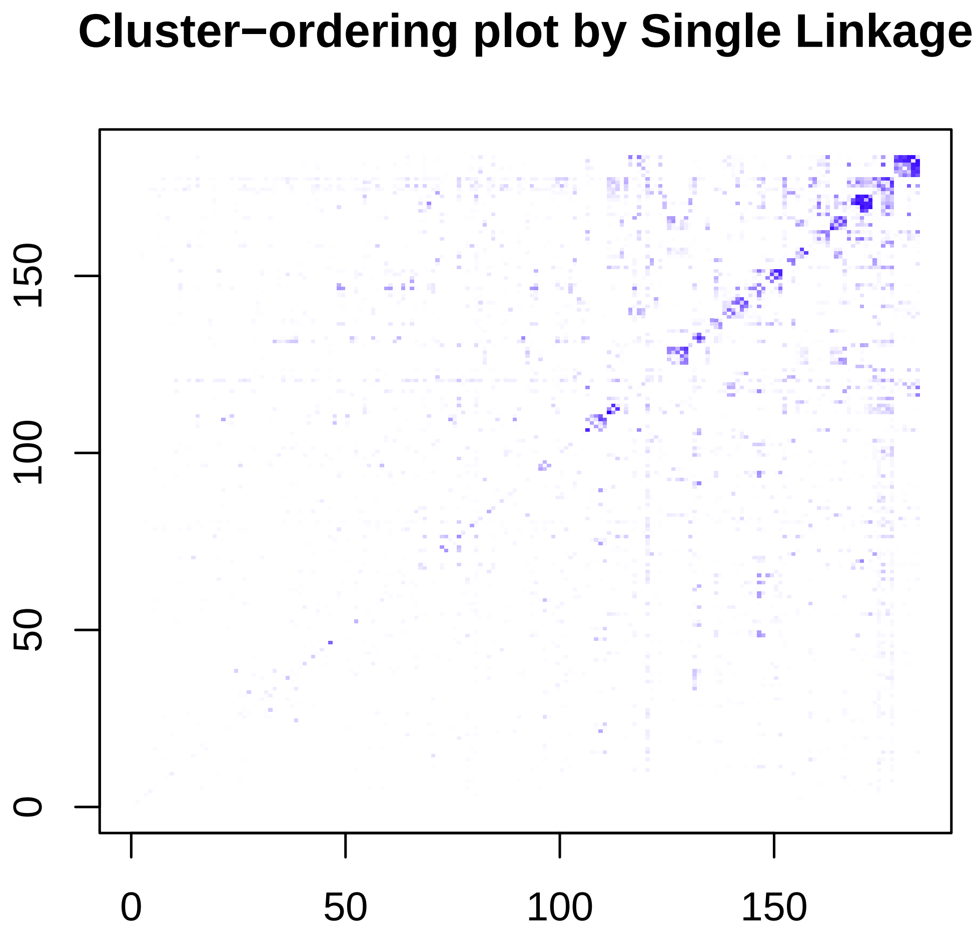


Figure 7.2: The cluster-ordering plot using single linkage

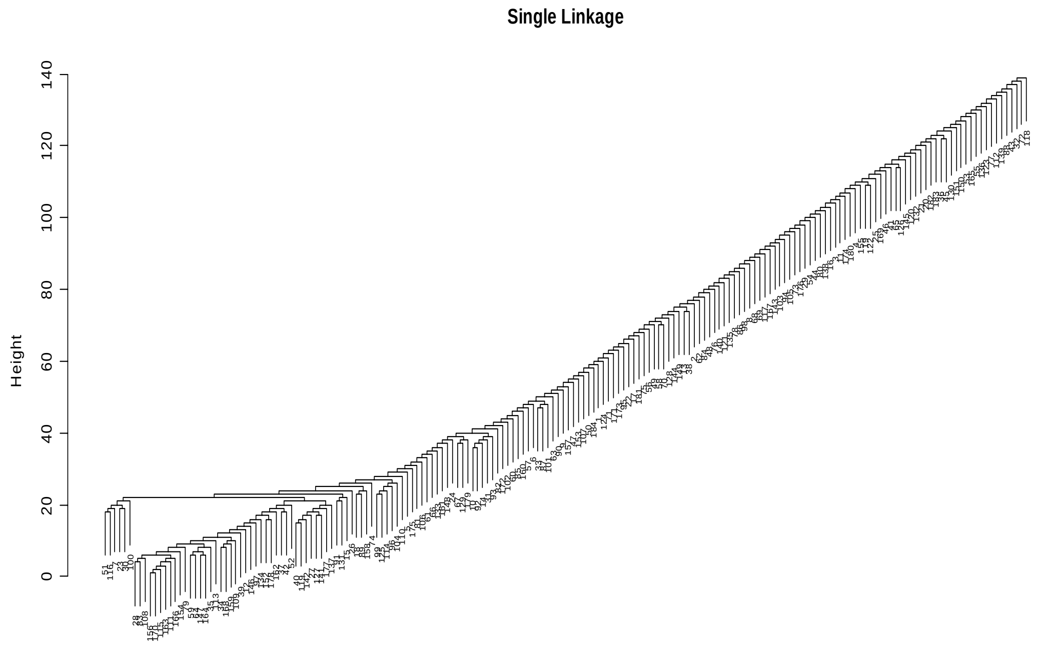


Figure 7.3: The dendrogram of the single-linkage clustering

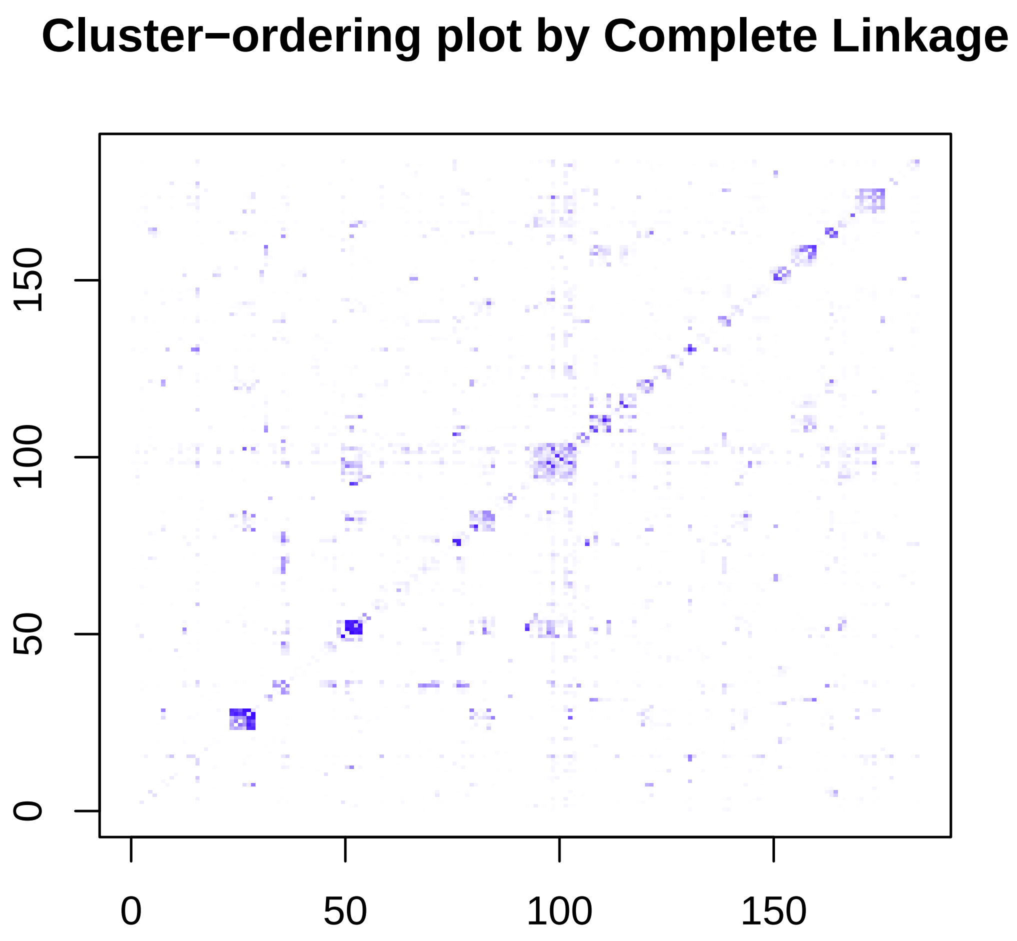


Figure 7.4: The cluster-ordering plot using complete linkage

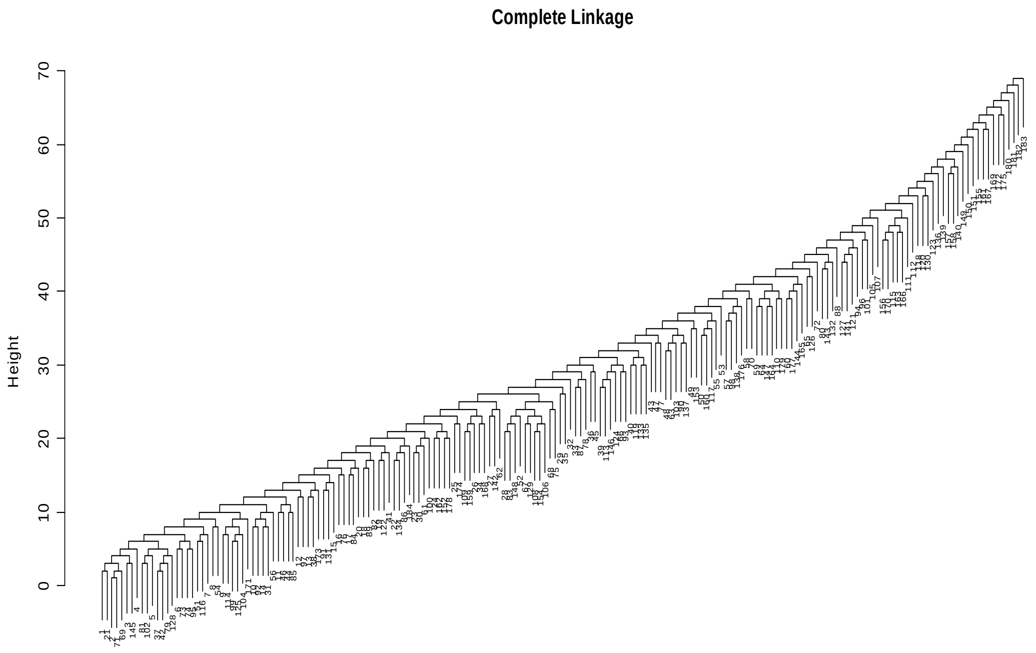


Figure 7.5: The dendrogram of the complete-linkage clustering

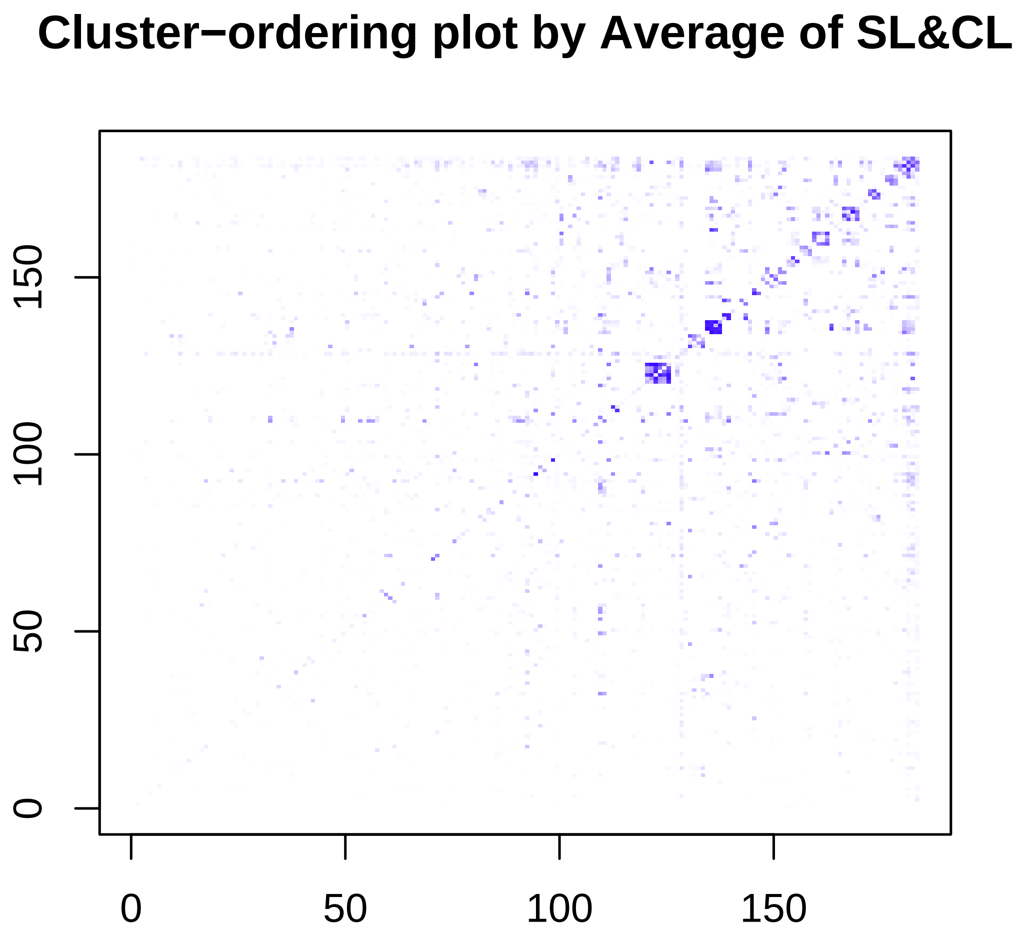


Figure 7.6: The cluster-ordering plot using the average of two linkages

and complete linkage. We see an obvious change in the clustering patterns. The blocks of dark-coloured squares are now located closer to the top right corner, and they are more concentrated. The dendrogram shown in Fig. 7.7 illustrates this change more clearly. The cluster tree is more balanced with larger clusters and fewer layers. In each layer of the dendrogram constructed by cluster tree averaging, a small number of points remain at each layer. Since there are fewer emails sent between employees corresponding to these points, we see an almost empty region around the bottom left corner in the cluster-ordering plot. The blocks of dark-coloured squares in the plot are associated with those clusters appearing in the deepest layers of the dendrogram.

It is interesting to discover which employees are grouped together, i.e. they communicated frequently, and the difference in groups from different clustering methods. We check the ids of employees corresponding to some interesting blocks of dark-coloured squares in each cluster-ordering plot.

In the plot derived by single linkage (see Fig. 7.2), the first block on the diagonal at the top right corner corresponds with a cluster with ids 111, 115, 156, 163, 166 and 170. There is little information about the employees in this group. A small block to the left of the first one on the diagonal contains ids 28, 83 and 108. They were two CEOs and the president of Enron. The third block on the diagonal to the left of the second has ids 59 (VP, government affairs), 64 (government relation executive), 147 (VP, regulatory affairs), and 164 (VP, chief of staff). We also see some communication between the second and the third group from the cluster-ordering plot.

In the plot derived by complete linkage (see Fig. 7.4), ID 108 is not in the cluster with 28 and 83, and is instead assigned to a cluster with 154, which is the ID of a chief operating officer. These two small groups appear slightly to the right of the middle of the diagonal of the plot. The group with ids 111, 115, 156, 163, 166 and 170 has also been identified by complete linkage, it appears as the first block on the diagonal at the bottom left corner in the plot. The group with ids 59, 64, 147, and 164 appears as the second block on the diagonal at the bottom left corner.

In the plot derived by the averaging of single and complete linkages (see Fig. 7.6), the group with IDs 59, 64, 147, and 164 and the group with ids 111, 115, 156, 163,

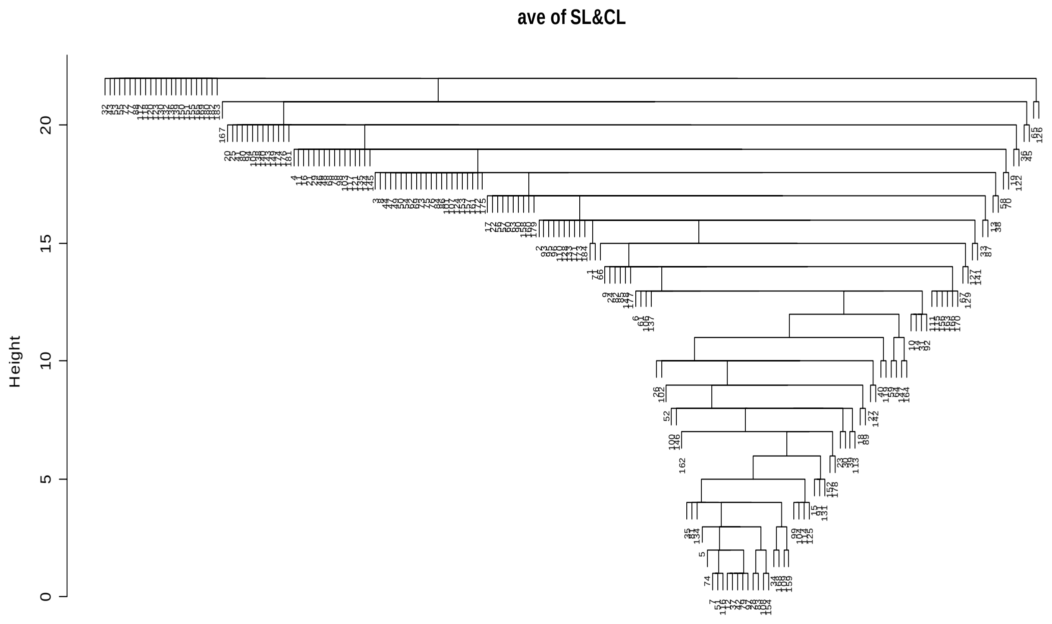


Figure 7.7: The dendrogram of the average of two linkages

166 and 170 are still appearing as two blocks of dark-coloured meshes. Unlike the clustering results by single and complete linkages, IDs 28 and 83 join IDs 108 and 154 to form a new cluster by the averaging.

We find a similar pattern of change when averaging DBSCAN with single linkage. In this Enron email application, cluster tree averaging shows an obvious ability to preserve the clustering results from each method in the average and also to discover new patterns.

7.3 Olive oil data set

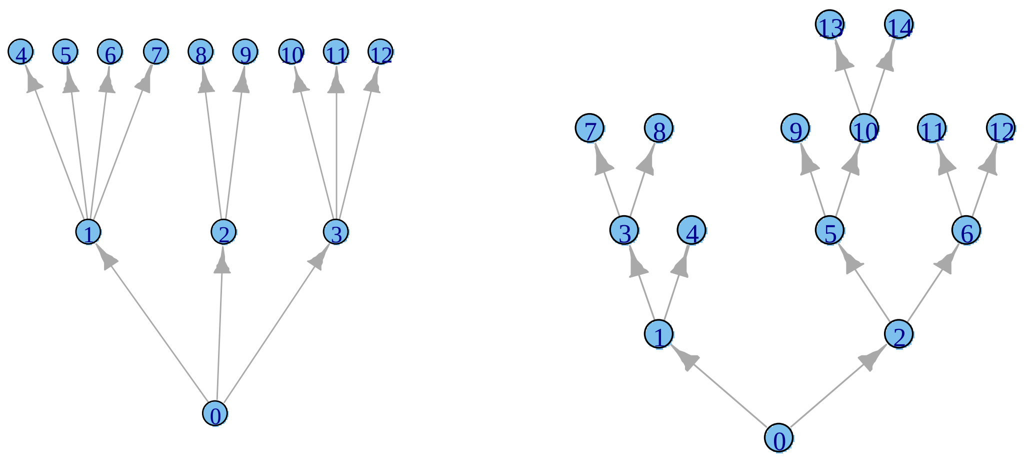
The olive oil data set contains 572 olive oil samples from Italy. Each sample has 10 variables. The first variable is the geographic region of Italy: North, South, or Sardinia. The second variable shows the area within each region. There are 3 areas in the North, 4 in the South, and 2 in Sardinia. The remaining variables are the percentages of 8 different fatty acids in the sample.

A major objective of data analysis for this data set is to determine the geographic locations of the samples from the eight acid contents. Although this is a typical classification problem, we can use clustering analysis to group the oil samples and to check the clustering result against the sample geographic locations.

Stuetzle [62] uses this data set to build a cluster tree using his runt pruning method. With our method of partition integration, we can build different cluster trees from different partitioning methods and compare their performance on the olive oil data set using our distance measure. We can further apply cluster tree averaging and bagging to different cluster trees constructed for this data set. This can give us a sense of how our distance measure and methods such as bagging would work on this data.

The first two variables can be used to construct the true cluster tree of the unknown underlying distribution of these samples. Figure 7.8(a) shows the true cluster tree structure of the olive oil data. The second layer of the tree contains 3 nodes corresponding to the 3 areas. The third layer has 9 nodes for the 9 regions. The performance of a clustering method as an estimator of this true tree is measured by our distance measure.

The clustering methods are denoted as in Appendix E. Table 7.1 shows the distance from the cluster tree constructed by each method to the true tree. Runt-pruning is a good estimator with the second smallest distance from the true tree; its cluster tree structure is shown in Fig. 7.8(b). Runt-pruning finds two areas instead of three at the second layer of the tree, and this contributes to the distance. However, the leaf nodes of the runt-pruning tree do a good job of grouping the samples into different regions. This makes the method a good estimator in this case.



(a) True tree structure (b) Tree structure from runt-pruning

Figure 7.8: Tree structures of the olive oil data

Distance to the true tree						
Method	K-varying	K-fixed	Single	Complete	Average	DBSCAN
Distance	0.6919	0.7535	0.9139	0.8078	0.6558	0.8960
Distance to the true tree						
Method	Spec-tree	KNN-coup	Runt-pruning	K-means	Spectral	Gaussian
Distance	0.9225	0.9868	0.5426	0.8580	0.7765	0.5162

Table 7.1: Estimate error of clustering methods for the olive data

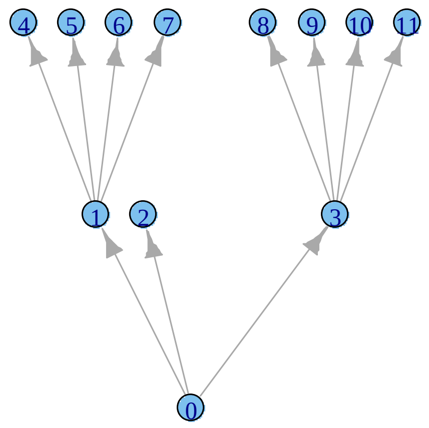


Figure 7.9: Tree structure using partitions from two methods

The smallest distance comes from the Gaussian partition. The Gaussian partition method finds three clusters in the data set. This partition almost finds the three areas in the data and therefore makes the first split of the root almost correctly. This makes the distance smaller. However, if we are interested in finding the regions rather than the areas, the Gaussian partition is not a good estimator.

Our method of partition integration provides a flexible platform for combining different partitions. We can combine the Gaussian partition with the partition from the leaf nodes of the runt-pruning tree. This gives us a better cluster tree for estimating both the area and the region. This tree has distance 0.4655, smaller than any distance in Table 7.1. The structure of this tree is illustrated in Fig. 7.9. The purpose of showing this tree is to demonstrate the flexibility of our framework. This flexibility is more useful when we use our experience to find cluster patterns in particular data sets.

Table 7.2 shows the results of bagging. Clustering bagging improves the performance of most of the estimators, especially the spectral tree method. For the best

Distance to the true tree						
Method	K-varying	K-fixed	Single	Complete	Average	DBSCAN
Method d	0.6919	0.7535	0.9139	0.8078	0.6558	0.8960
Bagging d	0.6537	0.7371	0.9040	0.7523	0.6188	0.8172
Distance to the true tree						
Method	Spec-tree	KNN-coup	Runt-pruning	K-means	Spectral	Gaussian
Method d	0.9225	0.9868	0.5426	0.8580	0.7765	0.5162
Bagging d	0.6191	0.9088	0.6575	0.7855	0.7810	0.6009

Table 7.2: Estimate error of clustering methods for the olive data

estimators such as runt-pruning and the Gaussian partition, bagging slightly worsens the performance. It gives us an overall stable estimator for this data set.

Boosting is a method for obtaining a better model from weak models through a weighted averaging [25]. Although the weights can not be obtained directly for clustering, we can still use the idea of boosting to obtain a better estimator from the weak estimators. To use this idea, we set up an experiment as follows:

1. For each fatty acid variable i, construct a 1-d data set with the values from this variable for all the samples. Denote this data set S_i, $i = 1, 2, \cdots, 8$.

2. For each S_i, obtain each partition $C_i^{(j)}$ by k-means with $k = j$ on S_i, $j = 3, 4, \cdots, 12$.

3. Construct a cluster tree $\widehat{T_i^{(j)}}$ for each $C_i^{(j)}$ by assigning each cluster of $C_i^{(j)}$ to a child node of a root that contains all the samples. In most cases, $\widehat{T_i^{(j)}}$ is a weak estimate of the true cluster tree.

4. Using our cluster averaging method, obtain $\widehat{T^{(j)}} = \frac{1}{8}\sum_{i=1}^{8} \widehat{T_i^{(j)}}$ for each j.

5. Check the performance of each $\widehat{T^{(j)}}$ by calculating the distance from $\widehat{T^{(j)}}$ to the true cluster tree for the 572 samples.

Distance to the true tree									
K	var 1	var 2	var 3	var 4	var 5	var 6	var 7	var 8	Boosting
3	0.959	0.992	1.072	0.871	1.005	0.989	0.964	0.825	0.870
4	0.983	0.991	1.111	0.950	1.028	1.001	1.02	0.915	0.827
5	0.996	1.00	1.154	1.019	1.043	1.073	1.068	0.975	0.826
6	1.019	1.057	1.180	1.028	1.062	1.083	1.089	0.978	0.887
7	1.043	1.068	1.197	1.035	1.087	1.122	1.072	1.005	0.847
8	1.083	1.112	1.204	1.052	1.101	1.104	1.105	1.018	0.833
9	1.083	1.076	1.227	1.070	1.124	1.145	1.145	1.037	0.675
10	1.131	1.082	1.238	1.079	1.137	1.153	1.104	1.058	0.803
11	1.136	1.138	1.238	1.112	1.123	1.156	1.139	1.042	0.961
12	1.150	1.158	1.253	1.121	1.149	1.161	1.137	1.125	1.000

Table 7.3: An example of cluster tree boosting

Table 7.3 shows the result of the above experiment. For most j, $\widehat{T^{(j)}}$ performs better than each $\widehat{T_i^{(j)}}$. For some j, e.g. $j = 9$, $\widehat{T^{(j)}}$ even becomes a good estimate of the true cluster tree. When $j = 3$, the first 7 estimates are bad but the 8^{th} estimate is relatively good, and this makes the boosting result slightly worse than the 8^{th} estimate.

The above applications to real data sets indicate that our framework can be a common tool for the clustering analysis of a real data set. It can combine different partitions; it can compare the performance of clustering methods; and it can reveal interesting patterns and stable clustering results through cluster tree averaging or bagging.

Chapter 8

Conclusion and discussion

Density-based clustering methods find clusters for a sample from an underlying continuous distribution. Hartigan defines a high-density level set [31, Section 11] and Stuetzle further defines a high-density cluster tree for a continuous distribution [62]. We regard this tree as a parameter of interest for a continuous distribution, which determines a cluster tree for a sample from this distribution. We further extend it to a discrete distribution whose support is located in a Euclidean space. The extension makes the parameter more general.

Any clustering method can be used to produce a partition of the data into disjoint groups from which a graph with disjoint connected subgraphs can be derived. This motivates us to propose a graph family framework.

Based on the graph family framework, we further propose a cluster-tree distance measure, which can be written as an inner product or kernel. This distance measure is simple, easy to understand, and easy to implement. Comparisions with related measures show that our measure is particularly suitable for both partitions and tree structures. Further testing verifies that our measure is good at comparing clustering performance.

More methods are obtained from the graph family framework. The method of partition integration combines different partitions; the method of cluster tree averaging combines different cluster trees; the method of cluster tree bagging is derived using bootstrapping. These methods produce a single cluster tree as a new estimate of the corresponding high-density cluster tree. Experiments and examples in the thesis show that the new estimate can have a better performance than the clusterings to be integrated.

The parameterization of clustering and the graph family framework and its applications are the major components of our clustering framework. This framework is an open platform that is able to take different clusterings as an input and to generate cluster trees as new estimates of high-density or -probability mass cluster trees. Statistical assessment of clustering via Monte Carlo simulation is another direct application of our clustering framework. The experiments and examples in the thesis show that the framework is also suitable for this purpose.

Our research and experiments have demonstrated the flexibility of our framework. An immediate research issue is how to further extend its use. One possible direction is to derive an algorithm that automatically searches the space of cluster trees using our framework. Interactive clustering methods, such as reducing, reassigning, and refining [53] could be considered in this search algorithm. Another possible direction is to develop methodologies that make our clustering distance measure more suitable for clustering stability.

Appendix A

Brief review of clustering methods

From 1960 to 1980, clustering was a widely used approach in numerical taxonomy [60]. The use of clustering in this area included applications in astronomy [2], financial markets [41], medicine [7], marketing [22], archaeology [33, 34], economics [19], business [27], phytosociology [45], microbiology [14, 26], psychiatry [18], and agriculture [63]. At that time, k-means [48] and linkage hierarchical clustering methods [31], including single linkage, complete linkage, and average linkage, were available. In the 1980s, with the development of information retrieval systems [59], document clustering [67, 71] attracted much attention. Hierarchical clustering methods were commonly used for document clustering [51, 29, 66, 69]. Motivated by the efficiency requirement for document clustering from large databases, reducing the computational expense became a hot issue [58, 12]. Testing and measuring the performance of a cluster also received attention [68, 1]. Internet technologies grew dramatically in the 1990s. Information retrieval is still a common application of clustering. Document and text clustering were applied to retrieving information from the internet [43, 61, 73]. To meet the increasing performance demands, new methods were introduced, including mix model methods [21], wavelet-based methods [72], principal component clustering [50], graph-theory-based clustering [44, 13, 39], and density-based clustering [17, 4].

In recent years, the explosive growth in data and databases has dramatically increased the demand for data mining [38]. Data mining requires efficient techniques and tools to discover useful information and knowledge from data. In this application, cluster analysis is growing rapidly in importance.

A well-accepted categorization of existing clustering methods is: partitioning methods, hierarchical methods, graph-theory-based methods, and statistical-model-generated methods (including mixture-model-based methods and density-based methods).

A.1 Partitioning methods

Partitioning methods appeared early in the history of clustering, long before the emergence of data mining. A common idea of partitioning is as follows: Separate objects into a fixed number of clusters such that the total deviation of each object from its cluster centre is minimized; then objects in the same cluster are close to each other, whereas those in different clusters are far away.

K-means [48] is a typical partitioning clustering method. In k-means each cluster is represented by its mean, the average of the data vectors in a cluster. K-means tries to minimize a cost function $\mathcal{E}$, which is the average dissimilarity from any object in the data set to the centre of the cluster to which it belongs. The k-means algorithm uses k, the number of clusters, as a parameter and applies greedy searching to find a local optimum. Different local optima can be found using k-means with different initial values. The computational complexity of k-means is $O(nkt)$, where n is the number of objects in the data set, k is the number of clusters, and t is the number of iterations for the algorithm to converge. It is efficient if $k << n$ and $t << n$.

K-means works well when the objects within each cluster are close to each other while the objects from different clusters are far away. Since k-means uses the mean of all the data objects in a cluster to represent that cluster, the result can easily be affected by outliers in the data set, and the method does not work well when the clusters are not sphere-shaped.

Similarly to k-means, k-medoids constructs clusters by using a single object—

a medoid—to represent a cluster [40]. Given a cluster C, we define $d(O_p, O_c)$ to be the distance between an object O_p in C and the cluster centre O_c, which is the centre of all the objects in C. The object O_p is a medoid of C only if $d(O_p, O_c) = min_{\forall O_q \in C} d(O_q, O_c)$. The cost function in k-medoids is the same as that in k-means but uses the medoid as the centre of a cluster.

The major difference between k-medoids and k-means is that the medoid of a cluster is an existing object whereas the mean of a cluster does not usually exist in the data set. These two algorithms also differ in how they recalculate the cluster representatives and how they reconstruct the clusters when the cluster representatives are changed.

K-medoids is not efficient for large data sets. The complexity of one iteration is $O(k(n-k)^2 t)$. Sampling techniques have been used to improve the efficiency of the algorithm. CLARA [40] is a typical such approach and CLARANS [52] is a method that is motivated by improving the effectiveness of CLARA.

A.2 Hierarchical methods

These methods construct a hierarchical decomposition of the data set. Two typical approaches are used. One is bottom-up or agglomerative and the other is top-down or divisive. In the initial step of bottom-up methods, each object forms a cluster. If the number of objects in the data set is n, there are n clusters at the initial state of the algorithm. The algorithm seeks the two closest clusters according to some distance measurement and merges these two clusters to form one. This is then repeated until only one cluster is left. In top-down methods, the algorithm starts with only one cluster in which all the objects are contained, and coninues splitting until each distinct object forms its own cluster.

Early hierarchical methods employ simple metric functions to measure the dissimilarities between any two clusters, say C_i and C_j. Three common measurements are [31]:

- Single linkage:

$$d_{min}(C_i, C_j) = min_{\forall p \in C_i, \forall q \in C_j} \|p - q\|$$

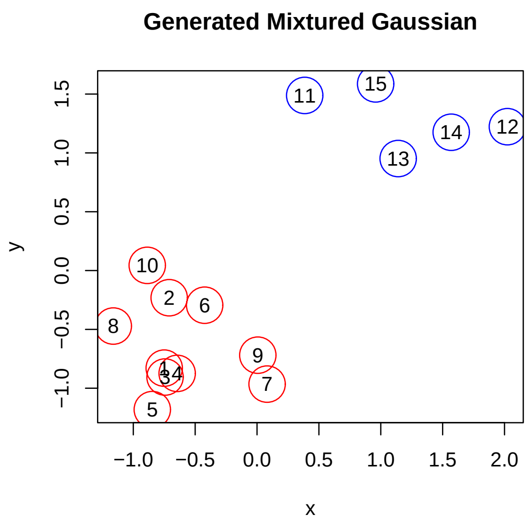
- Complete linkage:

$$d_{max}(C_i, C_j) = max_{\forall p \in C_i, \forall q \in C_j} \|p - q\|$$

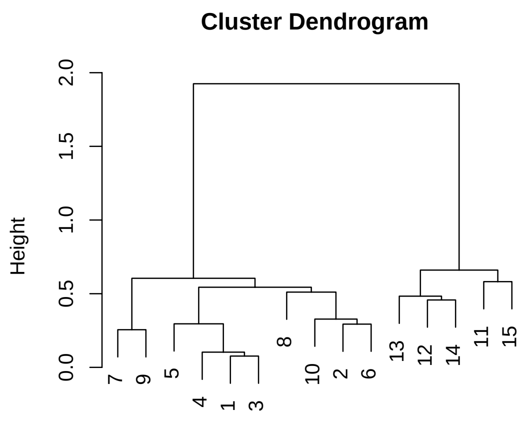
- Average linkage:

$$d_{avg}(C_i, C_j) = \frac{1}{|c_i||c_j|} \sum_{p \in C_i, q \in C_j} \|p - q\|$$

A dendrogram (a tree structure) is used to record the merging or splitting of the clusters and to indicate the distance between two joined clusters. The dendrogram is a cluster tree that displays a nested sequence of clusters and can be used to choose the number of clusters. Figure A.1(a) shows a data set generated from a mixture of



(a) Artificial data set



(b) Single-linkage dendrogram

Figure A.1: An example of a data dendrogram

two Gaussians. Figure A.1(b) shows the dendrogram of the single-linkage clustering of this data set. In this dendrogram, points or groups of points close to each other were merged first by the single-linkage distance measure, and after two large groups were formed, they were merged into one group. The height of a horizontal bar in the dendrogram is the distance between two groups that have been joined by the bar.

To obtain better hierarchical clustering, many approaches have been published, such as BIRCH [74] and CURE [30]. BIRCH creates a hierarchical structure of not only the data objects but also some general statistical information, such as the sample medians and sample variations. This information can be used to refine the clustering result and to reduce the computational requirement. CURE uses multiple points to represent a cluster and can find non-centre-based clusters.

A.3 Statistical-model-generated methods

Many clustering methods are based on the assumption that the data objects are generated from an underlying probability distribution that is unknown. In statistics, we often parameterize this distribution and then estimate the unknown parameters from the observed data.

A.3.1 Mixture-model-based clustering

The typical mixture-model-based clustering assumes that the observed data come from a mixture of distributions [21, 54]. The purpose of clustering is to estimate the unknown parameters of the mixture density function and the set of labels to assign each object to a subdistribution. The formulation is as follows:

Let the population **f** be a mixture of G subpopulations $f_k(\theta_k)$, where $k = 1, 2, \cdots, G$. Let the set of parameters for the subpopulation be $\theta = \{\theta_1, \theta_2, \cdots, \theta_G\}$. Given a data set $X = \{x_1, x_2, \cdots, x_n\}$ drawn from **f**, we have

$$\mathbf{f}(x_i|\theta) = \sum_{k=1}^{G} \pi_k f_k(x_i|\theta_k),$$

where π_k is the probability that an observation comes from subpopulation f_k and $\sum_{k=1}^{G} \pi_k = 1$. Let $\pi = \{\pi_1, \pi_2, \cdots, \pi_G\}$, then the likelihood for this mixture model is

$$L(\theta; \pi|X) = \prod_{i=1}^{n} \sum_{k=1}^{G} \pi_k f_k(x_i|\theta_k).$$

There is no closed form solution to maximize the likelihood. It is a nonlinear optimization problem, and is commonly solved by an EM algorithm.

The observed data set X is regarded as incomplete with missing values: $\gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_n\}$, where $\gamma_i = \{\gamma_{i1}, \gamma_{i2}, \cdots, \gamma_{iG}\}$. We have $\gamma_{ik} = 1$ if x_i comes from f_k and $\sum_{k=1}^{G} \gamma_{ik} = 1$. Now the complete data become $Y = \{y_1, y_2, \cdots, y_n\}$ with $y_i = (x_i, \gamma_i)$. With this complete data, the log likelihood becomes

$$l(\theta; \pi; \gamma|Y) = \sum_{i=1}^{n} \sum_{k=1}^{G} \gamma_{ik} log[\pi_k f_k(x_i|\theta_k)]$$

The E-step of the EM algorithm is

$$E(\gamma_{ik}|X; \pi^*; \theta^*) = \frac{\pi_k^* f_k(x_i|\theta_k^*)}{\sum_{k=1}^{G} \pi_k^* f_k(x_i|\theta_k^*)}$$

where π^* and θ^* are the current estimates of π and θ. The M-step maximizes the above log likelihood in terms of π and θ with γ fixed as the value calculated at the E-step.

The EM solutions for a mixture of Gaussians or a mixture of other distributions have been discussed in the literature [21]. We do not discuss such techniques here.

The above mixture likelihood and EM algorithm has a drawback. It requires knowledge of the underlying distribution that we may not have. For example, the EM process requires G, the number of subdistributions, to be fixed a priori. Some recent approaches try to solve this problem by for example using a hybrid hierarchical clustering on the mixture model algorithm with a large G or introducing stochastic search process to find local optima in the space of all possible partitions of the data [10]. Furthermore, the use of specific distribution densities should be questioned. For example, it is reasonable to use a mixture of Gaussians only if we have information that the data come from Gaussians.

In most cases, we do not have enough information to estimate the parameters to determine the mixture of density functions. Note also that in the "population" clusters can overlap. However, we may have enough information to estimate some of the parameters that determine a clustering.

A.3.2 Density-based methods

Density-based clustering assumes that multiple modes or disjoint high-density regions exist in the underlying distributions. The purpose of the clustering is to assign data objects from the same high-density region to the same cluster and objects from different high-density regions to different clusters. Clusters are found by obtaining the maximally connected subset of the data objects from high-density regions. Such connected subsets can be arbitrarily shaped.

DBSCAN [17] and OPTICS [4] use the empirical density estimation such that the density of a data object p is proportional to the number of data objects in the hypersphere centered at p with a predefined radius ϵ. If the number of points exceeds a predefined threshold $Minpts$, p becomes a core-point and a connected subset is formed by connecting every pair of points within the hypersphere. This connected subset can be extended when another core-point is found in the subset and all the points within its ϵ hypersphere region are added to the connected subset. When no more points can be added, the subset becomes a maximally connected subset of the data set and then this subset is a cluster. In DBSCAN and OPTICS, the algorithm for finding clusters involves a process of constructing a graph with disjoint connected subgraphs. The average run time complexity of DBSCAN is $O(nlogn)$.

The original DBSCAN finds a partition of the data using a global density level without considering nested high-density regions that would require different local density levels. These local density levels are easily defined from the building blocks in DBSCAN. We can get different density levels by fixing $Minpts$ and varying the value of ϵ. A sequence of nested partitions can be constructed by running DBSCAN several times with different density levels. OPTICS adds a complex structure to DBSCAN to efficiently discover nested density regions.

Based on Hartigan's definition, runt pruning clustering [62] is a way to build a high-density cluster tree that reveals natural clustering in many real data sets in which multiple modes exist with different density levels.

Runt pruning clustering estimates a high-density cluster tree using a nearest-neighbour density estimate via cutting the edges in the minimal spanning tree (MST) of the data set. It can therefore estimate density level sets at different levels. The MST of a data set is a graph used to connect all the vertices (objects in the data set) with a minimal edge length sum. Moreover, a pruning process is used in the algorithm to trim off insignificant clusters. Without the pruning process, the cluster tree constructed by runt pruning clustering is identical to a single-linkage dendrogram [28].

A.4 Graph-theory-based methods

A typical graph-based method is spectral clustering [16, 49]. Spectral clustering first constructs similarity graphs of the data set. The similarity graphs commonly used include: the ϵ-neighbourhood graph that connects all pairs of points if their distance does not exceed ϵ; the k-nearest neighbour graph; and the fully connected graph, a weighted graph constructed by connecting two points with a certain weight calculated from a similarity measure such as the Gaussian similarity function [47].

The next step of spectral clustering is to obtain a new data set using the graph Laplacians [15] of the similarity graph. This step involves the calculation of the graph Laplacian matrix L of the similarity graph and the eigen-decomposition of L. The new data set is constructed from the eigenvectors corresponding to the k largest eigenvalues of L, where k is the number of clusters to discover. The last step of spectral clustering applies a partition clustering method such as k-means to the new data set to get the result.

In contrast to spectral clustering, other graph-based approaches apply clustering algorithms directly to particular graphs such as the k-nearest neighbour graph [23].

Appendix B

Brief review of some clustering frameworks

By reviewing typical clustering approaches and theoretical studies of clustering from a statistical view, we discuss possible clustering frameworks.

B.1 General description of clustering

Clustering is normally regarded as an unsupervised learning algorithm that extracts "patterns" in the form of clusters from data. Although this description is satisfied by every clustering method, it is ambiguous. The patterns of interest might be high-density regions, groups of similar points with various definitions of similarity, or unusual regions or objects in the sample with different definitions of "unusual". It is necessary to find an unambiguous general description of clustering.

B.2 Underlying structure of clustering

All the approaches in this category assume that the data to be clustered come from unknown underlying distributions. However, they have different definitions of a cluster.

B.2.1 Density-based clustering

Some density-based clustering algorithms such as DBSCAN [17] assume that clusters come from high-density regions of the underlying distribution. They therefore find partitions of the data with some "noise" being rejected.

B.2.2 Mixture-model-based clustering

Mixture-model-based clustering assumes that the sample points in each cluster come from a component of an underlying mixture of distributions [21]. This framework has some good statistical features in that

1. Every distribution can be expressed as a mixture.

2. It has a close relation to classification models.

The second point is because the parameters, say C, that define the mixture can also define a clustering. Let X be a sample from the mixture. We can write the likelihood, prior, and posterior probabilities as $P(X|C)$, $P(C)$, and $P(C|X)$ respectively. The framework of mixture-model-based clustering is natural for these expressions.

However, the framework has some shortcomings. It relies on the assumption of a mixture of specific distributions. If we assume the data come from a mixture of Normals but they are actually drawn from a mixture of Exponentials, the clustering will be incorrect. Another problem of this framework is that if we look at the density function of the mixture, a bump or a mode need not correspond to a component distribution in the mixture. Figure B.1 shows the density function of a mixture with three component distributions. The solid curve shows the density function of the mixture and the dashed curves show the density functions of the three components in the mixture. Should there be just one cluster in a sample of this mixture, or three?

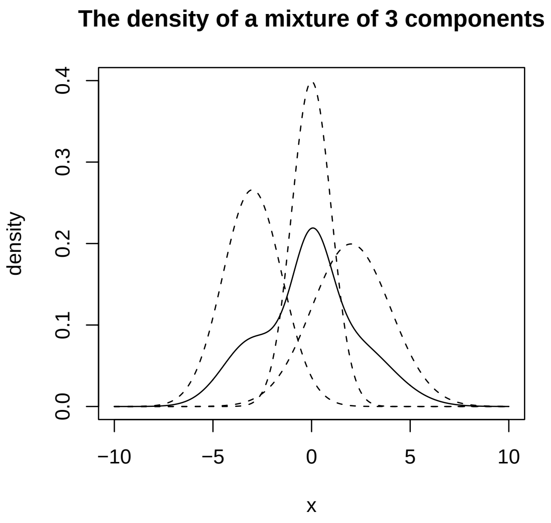


Figure B.1: A mixture of three Gaussians

B.2.3 High-density cluster tree

Hartigan and Stuetzle both assume that the goal of clustering is to discover bumps or modes in the density of the underlying distribution. Hartigan defined a high-density level set [31, Section 11]; Stuetzle further defined a high-density cluster tree [62] based on Hartigan's definition. This tree is a characteristic of the underlying distribution. It is reviewed in detail in Chapter 4 of the thesis.

B.3 Methodologies for combining clustering outcomes

Some existing approaches generate an integrated result from different clustering outcomes. For example, multiple-clustering generates a new partition from different partitions found by existing clustering methods, such as k-means with different values of k. There are different multiple-clustering approaches. Quan [56] proposes a method for constructing a vector for each sample point according to the input partitions and then applying a clustering method to the set of vectors derived. Ashlock [5] proposes a method that constructs a new partition by running k-means multiple times. Another approach constructs a tree or multiple trees using graph theory to find an optimal set of cliques from the graphs derived from a sequence of unnested partitions [13]. A clique is a complete subgraph.

A discussion of the above approaches is given in Chapter 3 for comparison with our method of partition integration.

B.4 Evaluation of clustering

B.4.1 Axiomatic approaches

In 1973, Wright proposed eleven axioms to capture five aspects of partitioning [70]. The aspects are:

1. The nature of the elements to be partitioned;
2. The similarity or distance measure for the elements to be partitioned;
3. The nature of the partitionings;
4. The nature of the partitioning results;
5. The clustering function.

Wright's axioms capture the general nature of clustering. Most of these axioms are obvious, for example changing the order of the elements should not change the partitioning result.

In some axiomatic approaches, a clustering function is defined as a function f that takes a distance function d on a set S and returns a partition τ of S. Axioms are proposed for this clustering function f. For example, Kleinberg proposes the following three axioms for f [42]:

Scale-invariance: Let $\alpha \cdot d$ be the distance function in which the distance between i and j is $\alpha \cdot d(i, j)$. For any distance function d and any $\alpha > 0$, we have $f(d) = f(\alpha \cdot d)$.

Richness: Let $Range(f)$ denote the set of all partitions τ such that $f(d) = \tau$ for some distance function d, then $Range(f)$ is equal to the set of all partition of S.

Consistency: When shrinking the distance between points inside a cluster and expanding the distance between points in different clusters, we get the same partition of S.

Kleinberg further proves an impossibility theorem that follows from these axioms. The theorem says that for each $n \geq 2$, where n is the size of S, no clustering function satisfies all three axioms. This theorem makes Kleinberg's three-axiom approach questionable. Indeed, the third axiom is less obvious. The shape of a cluster can be changed by the shrinking and expanding.

There exist other axiomatic approaches for a clustering function by Kalai, Papadimitriou, et al. [37], for hierarchical clustering results by Jardine and Sibson [36], and for cost functions for clustering by Puzicha, Hofmann et al. [55].

B.4.2 Clustering stability

The stability of a clustering method has been proposed as a measure of its performance [8]. For example, clustering stability is defined by Ben-David [8] as

$$\beta(\mathcal{A}, P, m) = E_p(d(C(S_m), C(S'_m))$$

where S_m and S'_m are two independent samples of size m drawn from an underlying distribution P, $C(S_m)$ and $C(S'_m)$ are two clusterings constructed on the corresponding samples by an algorithm $\mathcal{A}$, and d is a distance measure for clusterings. The clustering algorithm $\mathcal{A}$ is stable if $\beta(\mathcal{A}, P, m)$ is small. The distance measure is crucial for clustering stability.

Appendix C

Some standard definitions from graph theory

Definition C.0.1 *A* **path** *of a graph is a sequence of edges and vertices that connects two vertices in the graph. More specifically,* $v_1, e_1, v_2, e_2, \ldots, e_{k-1}, v_k$ *is a path from* v_1 *to* v_k *if each* v_i *is a vertex in the graph, each* e_i *is an edge in the graph, and each* e_i *is an edge between vertices* v_i *and* v_{i+1} *in the graph (regardless of direction).*

Definition C.0.2 *A* **directed path** *of a graph is a sequence of directed edges from one vertex to another in the graph. More specifically,* $v_1, e_1, v_2, e_2, \ldots, e_{k-1}, v_k$ *is a directed path from* v_1 *to* v_k *if each* v_i *is a vertex in the graph, each* e_i *is an edge in the graph, and each* e_i *is an edge from vertex* v_i *to* v_{i+1} *in the graph.*

In Fig. C.1(b), there exist paths from c to e but no directed path from c to e.

Definition C.0.3 *A* **weakly connected graph** *is a graph in which there exists a path between every pair of vertices in the graph.*

Definition C.0.4 *A* **strongly connected graph** *is a directed or mixed graph in which there exists a directed path from every vertex in the graph to every other vertex in the graph.*

Note that in a strongly connected graph there is a directed path between every pair of vertices, meaning that for every pair v_i and v_j, there is a directed path both from v_i to v_j and from v_j to v_i.

Figure C.1 shows simple examples of connected graphs. Graphs (a) and (b) are

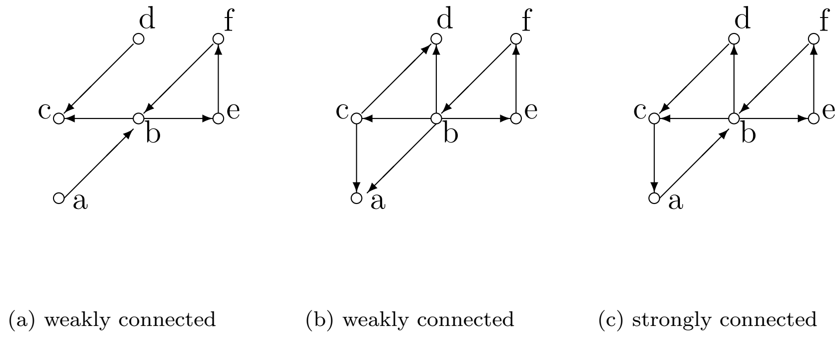


Figure C.1: Examples of connected graphs.

both weakly connected but not strongly connected. There exist paths but no directed paths between **a** and **d** in graphs (a) and (b). Graph (c) is strongly connected.

Using these definitions, a graph can be decomposed into separate components, defined as follows.

Definition C.0.5 *A* **weakly connected component** *of a graph is a weakly connected subgraph that is not part of any larger weakly connected subgraph. Such a weakly connected graph is also called a* **maximum weakly connected subgraph**.

Essentially, a path joins any two vertices in the same component and there is no path between vertices from different components. For example, the entire graph of Fig. C.1(b) is a weakly connected component.

Definition C.0.6 *A* **strongly connected component** *of a directed (or mixed) graph is a strongly connected subgraph that is not part of any larger strongly connected subgraph.*

Essentially, for any two vertices v_i and v_j in a strongly connected component, there is a directed path from v_i to v_j and from v_j to v_i. Note that there can be paths from vertices in one strongly connected component to vertices in another.

For example, in Fig. C.1(b), there are four strongly connected components: $\{b, e, f\}$, $\{a\}$, $\{c\}$, and $\{d\}$.

For convenience, we refer to a weakly connected component or a strongly connected component generically as a connected component.

It is helpful to have notation that defines the components of a graph. Let $c_j(G)$ denote the j^{th} connected component of a graph G. We have $c_i(G) \cap c_j(G) = \phi$ for $i \neq j$.

Appendix D

A four-spring model

In a two-dimensional discrete distribution, to determine whether two points from $L(\lambda; m)$ are contiguous or not, we consider an elastic system. For example, in Fig. 1.5 from Chapter 1, the contiguity of points a and g is reasonably determined by themselves and by points e and f. Here is a natural way to build an elastic system. We take a piece of balloon and place it on the four points a, g, e, and f located on a horizontal plane. We mark their locations on the balloon and pull the balloon vertically at each of the four marked locations to a height proportional to the probability mass of the corresponding original point. To be more specific, suppose the probability mass of the point g is m_g, the height of the location on the balloon that is marked by g is αm_g with $\alpha > 0$. Let $H_\lambda = \alpha\lambda$. We next find the point p on the balloon whose projection on the 2-d plane (determined by the original four points) is exactly at m, the middle point of the line segment joining points a and g. Finally we check the height of point p, and if this height is not less than H_λ, a and g are contiguous; otherwise they are not. Note that m can be any point (not necessarily the middle point) on the line segment connecting a and g; we use the middle point for simplicity.

Mathematically, the above elastic system is simply approximated by a four-spring model. Suppose we have four points in two-dimensional space $p_i^{(2)} =< x_{i1}, x_{i2} >$, $i = 1, 2, 3, 4$. We add a third-dimension coordinate x_{i3} to these points to get $p_i^{(3)} =<$

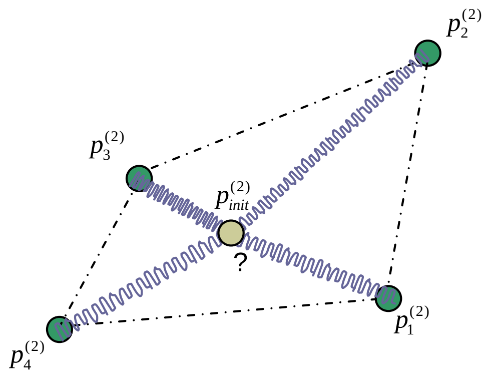


Figure D.1: The initial state of the four-spring model

$x_{i1}, x_{i2}, x_{i3} >$, where x_{i3} is the probability mass of the point p_i. If the line segment connecting $p_2^{(2)}$ and $p_4^{(2)}$ intersects the line segment connecting points $p_1^{(2)}$ and $p_3^{(2)}$, a convex hull can be constructed whose vertices are these four points. Let $p_0^{(2)} =< x_{01}, x_{02} >$ be an arbitrary point located inside the convex hull. For simplicity, we consider $p_0^{(2)}$ to be the middle point of the line segment connecting $p_2^{(2)}$ and $p_4^{(2)}$.

We can build a four-spring model for these points as follows. Let $p_{init}^{(2)} =< y_1, y_2 >$ be an unknown point in two-dimensional space. We place spring S_i to connect points $p_i^{(2)}$ to $p_{init}^{(2)}$ with length $|p_i^{(2)} - p_{init}^{(2)}|$ such that these four springs are at rest. Figure D.1 shows the four-spring model in its initial state. Now we move the outer end of S_i to $p_i^{(3)}$ and keep all the inner ends of these springs connected together. Let $p^{(3)}$ be the point at which these four springs connect when the combination of forces from the four springs is again (now in $\mathbb{R}^3$) at zero, and whose projection of $p^{(3)}$ onto the original two-dimensional space is exactly $p_0^{(2)} =< x_{01}, x_{02} >$. We denote $p^{(3)} =< x_{01}, x_{02}, y_3 >$. We are interested in finding the values of y_1, y_2, and y_3. Figure D.2 shows the four-spring model in 3-d space.

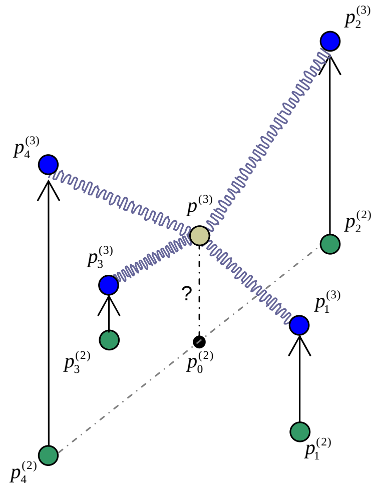


Figure D.2: The four-spring model in the 3-d space

If we project the force of each spring onto the three standard coordinates, the combination of the four projected forces is at zero as well on each coordinate. Hooke's law says

$$\mathbf{F} = -k\mathbf{X}$$

where $\mathbf{X}$ is the displacement by which the spring is elongated, $\mathbf{F}$ is the restoring force exerted by the spring, and k is the spring constant or force constant of the spring. Let $l_i^{(2)} = |p_i^{(2)} - p_{init}^{(2)}|$ be the original length at rest of the i^{th} spring denoted by S_i. The current length of S_i is $l_i^{(3)} = |p_i^{(3)} - p^{(3)}|$. Note that $l_i^{(2)}$ is a function of y_1 and y_2 and $l_i^{(3)}$ is a function of y_3.

We assert that the spring constant k_i of S_i in this model is inversely proportional to the original length of S_i, which is $l_i^{(2)}$. For generality we set $k_i = \left(l_i^{(2)}\right)^{-\beta}$ where $\beta > 0$. The extension of S_i on the j^{th} $(j = 1, 2, 3)$ coordinate is

$$\frac{l_i^{(3)} - l_i^{(2)}}{l_i^{(3)}}\left(p^{(3)}[j] - x_{ij}\right)$$

where $p^{(3)}[j]$ is the j^{th} coordinate of $p^{(3)}$. By Hooke's law, the force from S_i on the j^{th} coordinate is

$$-\frac{1}{\left(l_i^{(2)}\right)^{\beta}}\frac{l_i^{(3)}-l_i^{(2)}}{l_i^{(3)}}\left(p^{(3)}[j]-x_{ij}\right)=\left(\left(l_i^{(2)}\right)^{-\beta}-\frac{1}{l_i^{(3)}}\left(l_i^{(2)}\right)^{1-\beta}\right)\left(x_{ij}-p^{(3)}[j]\right)$$

The above model is defined mathematically as follows. Define $w_i=\left(l_i^{(2)}\right)^{-\beta}-\frac{1}{l_i^{(3)}}\left(l_i^{(2)}\right)^{1-\beta}$; note that when $\beta=1$, $w_i=1/l_i^{(2)}-1/l_i^{(3)}$. With β fixed, we have the following system of three equations with unknowns $y=(y_1,y_2,y_3)$:

$$\begin{cases} f_1(y)=\sum_{i=1}^4 w_i(x_{i1}-p^{(3)}[1])=0\\ f_2(y)=\sum_{i=1}^4 w_i(x_{i2}-p^{(3)}[2])=0\\ f_3(y)=\sum_{i=1}^4 w_i(x_{i3}-p^{(3)}[3])=0\end{cases}$$

This is equivalent to

$$F(y)=\sum_{i=1}^{4} w_i(p_i^{(3)}-p^{(3)})=0$$

Because $F(y)$ is not a linear system, we solve it numerically. Using Newton's method, the problem can be solved by the following iterative function:

$$y_{new}=y_{old}\ -\ J_{old}^{-1}\,F(y_{old})$$

where

$$J=\begin{bmatrix}\frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \frac{\partial f_1}{\partial y_3}\\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \frac{\partial f_2}{\partial y_3}\\ \frac{\partial f_3}{\partial y_1} & \frac{\partial f_3}{\partial y_2} & \frac{\partial f_3}{\partial y_3}\end{bmatrix}$$

and

$$\frac{\partial f_j}{\partial y_k}=\begin{cases}\sum_{i=1}^4\left(A_i(x_{ij}-p^{(3)}[j])(x_{ik}-y_k)\right) & \text{if } j=1,2,3;\ k=1,2\\ \sum_{i=1}^4\left(B_i(x_{ij}-p^{(3)}[j])(x_{ik}-y_k)\right) & \text{if } j=1,2;\ k=3\\ \sum_{i=1}^4\left(B_i(x_{ik}-y_k)^2+w_i\right) & \text{if } j=3;\ k=3\end{cases}$$

where $A_i=\beta\left(l_i^{(2)}\right)^{-\beta-2}-\frac{\beta-1}{l_i^{(3)}}\left(l_i^{(2)}\right)^{-\beta-1}$, $B_i=\left(l_i^{(2)}\right)^{1-\beta}\left(l_i^{-3}\right)$.

The above four-spring model is a building block for determining contiguity among points in a discrete distribution. A straightforward guideline would determine the contiguity of any pair of points in the distribution. However, from our previous assertion, it is not necessary to check the contiguity of two points far away from each other, and the influence from other points far away can be ignored. For example, in Fig. 1.5, it is more reasonable to discuss the contiguity of points a and c than a and i, and for the contiguity of a and c, points b and d play a more important role than h and f. This implies that we are more interested in finding contiguity of points locally than globally. To be more specific, we apply the Voronoi diagram [6] to determine the neighbourhood of a point and search for its contiguous points within this neighbourhood.

The four-spring model is built on two pairs of points: there are two points whose contiguity is being determined, called the main points, and two other points that help determine the contiguity, called the side points. To be more natural, we further require that the two main points and two side points should form a four-vertex convex hull with each edge connecting a main point and a side point. The model determines the contiguity of the two main points using both their configuration and the effect of the two side points.

The choice of the two side points can be arbitrary. We can choose all possible pairs of side points in the distribution. For each pair, we form a four-spring model together with the two main points. However, there are two problems with this approach:

- Determining the contiguity of all these models might be difficult;
- It is not efficient to define contiguity for a distribution with a large number of points.

We do not have to build so many models to determine the contiguity of a pair of points. We assert that points close to each other have more mutual effect than points far away. Thus, it suffices to search for side points in a common neighbourhood of the two main points and we use the Voronoi diagram to determine this neighbourhood.

In a Voronoi diagram, each point (or site) has a face around it and we say that two points share an edge if their faces have a common edge. We call a set of points the direct neighbourhood of a point $\mathbf{p}$ in a Voronoi diagram if any point in this set

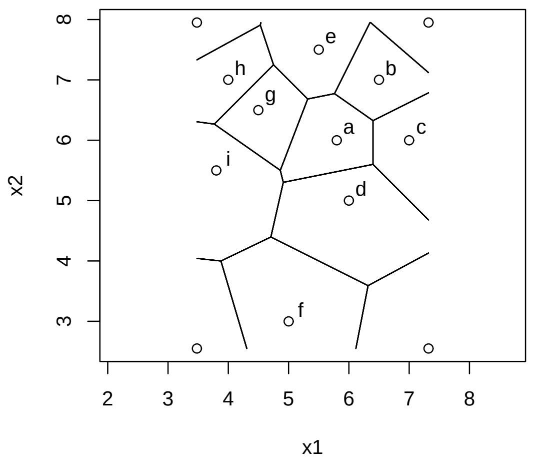


Figure D.3: The Voronoi diagram of a data set

shares at least one edge with **p**. The extended neighbourhood of a pair of points **p** and **q**, if **p** and **q** are located in the direct neighbourhood of each other, is the set such that any point in the set shares at least two edges with the points in the union of the direct neighbourhoods of **p** and **q**. If two points **p** and **q** are direct neighbours to each other, the neighbourhood of these two points is the union of three sets: the direct neighbourhoods of both **p** and **q** and the extended neighbourhood of **p** and **q**. For example, Fig. D.3 gives the Voronoi diagram of the data set shown in Fig. 1.5. The direct neighbourhood of point a is the set $\{b, c, d, e, g, i\}$ and the direct neighbourhood of point g is the set $\{a, e, h, i\}$. Points a and g are located in the direct neighbourhood of each other. The extended neighbourhood of a and g is the set including all nine points.

Using the Voronoi diagram, we can easily decide how to set up a pair of main points and a pair of side points to form a four-spring model. More precisely, we determine the contiguity of two points only if they are located in each other's direct neighbourhood; the two side points of a pair of main points must be located in the neighbourhood of these two main points. For simplicity, we pick up two side points only if they have the minimum sum of distances to the middle of the line segment connecting the two main points.

We use the Voronoi diagram rather than its dual graph, a Delaunay triangulation. The reason is that data points from a discrete distribution may not satisfy the assumption of a general position. For example, when the points are located on the vertices of a regular grid, the vertices of the voronoi diagram have degree four, and the dual graph has faces of convex quadrilaterals. To make the dual graph a Delaunay triangulation graph we have to add an additional edge to change the quadrilateral into two triangles. This makes the algorithm of searching for neighbourhoods complex because there is no unique way to add such an edge.

Before solving the nonlinear system of the four-spring model, we have to fix the value of β, say $\beta = 1$. For better performance, we optimize β. Using the four-spring model, we can estimate the probability mass of any point inside the convex hull formed by the two pairs of points from the model. This allows us to estimate the mass of any point in the distribution if there exist four points in its direct neighbourhood that satisfy the condition for forming a four-spring model. Therefore, we can optimize β by minimizing the sum of squares of the estimate error for the mass of each predictable point in the distribution. A point is not predictable if there do not exist four points in its neighbourhood such that a four-spring model can be constructed or if it has higher or lower mass than all the four points that can form the four-spring model.

To determine contiguity among points in a discrete distribution, the four-spring model approach is natural and easy to explain, but it has limits. The model is good for two-dimensional distributions, but it is not straightforward to build and explain such a model in a higher dimensional space. If we want to build the four-spring model in a higher dimensional space, the problems are:

- We use four points to build a model in two-dimensional space, but how many points are necessary in a higher dimensional space?
- In two-dimensional space we require the pair of main points and the pair of

side points to form a convex hull as described in this section, but this condition is difficult to satisfy and check in higher dimensional spaces.

- When handling data in a high dimensional space, the computational cost is a concern.

An alternative would be to use a weighted surface fitting such as weighted least squares to replace the four-spring model to determine contiguity locally. This approach is less natural but is not restricted to two-dimensional space. However, it still has the problem of finding a reasonable local region for the surface fitting especially in high dimensional spaces.

Both the four-spring model and the weighted surface fitting method require a local search of points for the model fitting which requires significant computational time. Furthermore, solving the four-spring nonlinear system or fitting the weighted least squares is not cheap computationally, and we have to build such models repeatedly to determine the contiguity among all the points in a distribution.

Actually, both methods more or less apply a density estimation method to determine contiguity. This makes the algorithm computationally expensive.

Appendix E

Clustering methods used for experiments

The clustering methods we used for the experiments in this thesis are as follows.

- K-varying: Cluster tree constructed by k-means with k from 2 to 8.
- K-fixed: Cluster tree constructed by k-means with fixed $k = 8$ but different random starts.
- Single: Single linkage.
- Complete: Complete linkage.
- Average: Average linkage.
- DBSCAN: Cluster tree constructed by DBSCAN with different parameter settings (fix $Minpts$ at 4, with ϵ varying).
- Spec-tree: Cluster tree constructed by Spectral clustering with the number of clusters 2 and 3 respectively.
- Gaussian-tree: Hierarchical Gaussian-model-based clustering.
- Runt-pruning: The runt pruning method.

- KNN-Coup: Coupling graph method on the estimated densities obtained by KNN ($K = int(log(n))$, where n is the sample size, $int(r)$ is a function to return the integer part of a real number r).
- K-means: K-means partition with k the number of modes in the distribution.
- Spectral: Spectral partition with the number of clusters being the number of modes in the distribution.
- Gaussian: Partition from the Gaussian-model-based clustering.

In the above methods, the cluster trees from K-varying, K-fixed, DBSCAN, and Spec-tree are constructed by our method of partition integration. The cluster trees from partitioning methods such as K-means, Spectral, and Gaussian are constructed by adding a common root node to different clusters in the partition found.

Note that the method of partition integration is proposed in Section 3.2; the coupling graph method is proposed in Section 4.5; and the other clustering methods in the above list are briefly reviewed in Appendix A.

Bibliography

[1] E.-H. Abdelmoula and P. Willett. Techniques for the measurement of clustering tendency in document retrieval systems. *Journal of Information Science*, 13:361–365, 1987.

[2] G. O. Abell. The distribution of rich clusters of galaxies. *The Astrophysical Journal Supplement Series*, 3(32):221–288, 1960.

[3] R. Ambauen, S. Fischer, and H. Bunke. Graph edit distance with node splitting and merging and its application to diatom identification. In *IAPR Workshop GbRPR*, pages 95–106, 2003.

[4] M. Ankerst, M. Breunig, H. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proc. ACM SIGMOD'99 Int. Conf. on Management of Data*, Philadelphia, 1999.

[5] D. Ashlock, E. Kim, and L. Guo. Multi-clustering: Avoiding the natural shape of underlying metrics. In *Canadian Statistical Society Meetings*, 2006.

[6] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Mathematical Software*, 22:469–483, 1996.

[7] D. N. Baron and P. M. Fraser. Medical applications of taxonomic methods. *British Medical Bulletin*, 24:236–240, 1968.

[8] S. Ben-David. A notion of stability for statistical clustering with applications to model selection. *Technical Report, University of Waterloo*, 2005.

[9] A. Ben-Hur, A. Elisseeff, and I.Guyon. A stability based method for discovering structure in clustered data. In *Pacific Symposium on Biocomputing*, volume 7, pages 6–17, 2002.

[10] J.G. Booth. Clustering using objective functions and stochastic search. *Journal of the Royal Statistical Society*, 2008.

[11] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[12] F. Can and A. Ozkarahan. Dynamic cluster maintenance. *Information Processing and Management*, 25:275–291, 1989.

[13] J. D. Carroll and J. E. Corter. Application of high-resolution computer graphics to pattern recognition analysis. *Journal of Classification*, 12:283–313, 1995.

[14] P. Chakraverty. Antigenic relationship between influenza b viruses. *World Health Org.*, 45:755–766, 1971.

[15] F. Chung. Spectral graph theory. *CBMS Regional Conference Series*, 92, 1997.

[16] W. Donath and A. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Develop.*, 17, 1973.

[17] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD-96*, 1996.

[18] B. S. Everitt, A. J. Gourlay, and R. E. Kendell. Attempt at validation of traditional psychiatric syndromes by cluster analysis. *Brit. J. Psychiat.*, 119:399–412, 1971.

[19] W. D. Fisher. *Clustering and Aggregation in Economics.* Johns Hopkins, Baltimore, 1969.

[20] E.B. Fowlkes and C.L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383), 1983.

[21] C. Fraley and A. E. Raftery. Mclust: Software for model-based cluster analysis. *Journal of Classification*, 16:297–306, 1999.

[22] R. E. Frank and P. E. Green. Numerical taxonomy in marking analysis, a review article. *Journal of Business Research*, 5:83–98, 1968.

[23] P. Franti, O. Virmajoki, and V. Hautamaki. Fast agglomerative clustering using a k-nearest neighbor graph. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 28, 2006.

[24] A. Fred and A. K. Jain. Evidence accumulation clustering based on the k-means algorithm. In *Structural, Syntactic and Statistical Pattern Recognition*, pages 442–451, 2002.

[25] Y. Freund. Boosting a weak learning algorithm by majority. *Inform. and Comput.*, 121:256–285, 1995.

[26] N. Goodfellow. Numerical taxonomy of some nocardioform bacteria. *J. Gen. Microbiol.*, 69:33–80, 1971.

[27] F. Goronzy. A numerical taxonomy on business enterprises. In Cole A. J., editor, *Numerical Taxonomy*, 1970.

[28] J. C. Gower and G. Ross. Minimal spanning tree and single linkage cluster analysis. *Applied Statistics*, 18:54–64, 1969.

[29] A. Griffiths, L. Robinson, and P. Willett. Hierarchic agglomerative clustering methods for automatic document classification. *Journal of Documentation*, 40:175–205, 1984.

[30] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *SIGMOD'98*, Seattle, 1998.

[31] J. Hartigan. *Clustering Algorithms*. John Wiley & Sons, Inc., New York, 1975.

[32] J. Hartigan. Statistical theory in clustering. *Journal of Classification*, 2, 1985.

[33] F. R. Hodson. Searching for structure within multivariate archaeological data. *World Archaeology*, 1:90–105, 1969.

[34] F. R. Hodson. Cluster analyses and archeology. *World Archaeology*, 1:299–320, 1970.

[35] C. Hurley. Clustering visualizations of multidimensional data. *Journal of Computational and Graphical Statistics*, 13(4):788–806, 2004.

[36] N. Jardine and R. Sibson. *Mathematical Taxonomy.* John Wiley & Sons, Inc., New York, 1971.

[37] A. Kalai, C. Papadimitriou, S. Vempala, and A. Vetta. *Personal communication.* 2002.

[38] M. Kantardzic. *Data Mining: Concepts, Models, Methods, and Algorithms.* John Wiley & Sons, Inc., New York, 2003.

[39] G. Karypis, E. Ahn, and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32:68–75, 1999.

[40] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, Inc., New York, 1990.

[41] B. F. King. Clusters of stocks are obtained by using a joining algorithm on stock price data. *Journal of Business*, 39:139–190, 1966.

[42] J. Kleinberg. An impossibility theorem for clustering. In *Proc. of the 16th Conference on Neural Information Processing Systems*, 2002.

[43] R. LaRose and A. Hoag. Organizational adoptions of the internet and the clustering of innovations. *Telematics and Informatics*, 13:49–61, 1996.

[44] B. K. Lavine and A. B. Stine. Application of high-resolution computer graphics to pattern recognition analysis. *Journal of Chemical Information and Computer Sciences*, 33:826–834, 1993.

[45] H. Lieth and G. W. Moore. Computerized clustering of species in phytosociological tables and its utilization in field work. In *Spatial Patterns and Statistical Distributions*, 1970.

[46] B. Liu, L. Ku, and W. Hsu. Discovering interesting holes in data. In *IJCAI Conference*, 1997.

[47] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 2007.

[48] J. MacQueen. Some methods for classification and analysis of multivariate observation. *Proc. 5th Berkeley Symp. Math. Stat. Prob.*, 1:281–297, 1967.

[49] M. Meila and J. Shi. A random walks view of spectral segmentation. In *8th International Workshop on Artificial Intelligence and Statistics*, 2001.

[50] J. Moore. Web page categorization and feature selection using association rule and principal component clustering. In *7th Workshop on Information Technologies and Systems*, 1997.

[51] F. Murtagh. Structure of hierarchic clusterings: Implications for information retrieval and for multivariate data analysis. *Information Processing and Management*, 20:611–617, 1984.

[52] R. T. Ng and J. Han. Clarans: A method for clustering objects for spatial data mining. In *IEEE Transactions on Knowledge and Data Engineering*, volume 14, 2002.

[53] R. W. Oldford. Structure of interactive cluster analysis. In *SSC Annual Meeting*, Halifax, 2003.

[54] C. Posse. Hierarchical model-based clustering for large data sets. *Journal of Computational and Graphical Statistics*, 2000.

[55] J. Puzicha, T. Hofmann, and J. Buhmann. Theory of proximity based clustering: Structure detection by optimization. *Pattern Recognition*, 33, 2000.

[56] T. Quan, S. Hui, and A. Fong. Mining multiple clustering data for knowledge discovery. *Lecture Notes in Computer Science*, 2843:452–459, 2003.

[57] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971.

[58] M. Rasmussen and P. Willett. Efficiency of hierarchic agglomerative clustering using the icl distributed array processor. *Journal of Documentation*, 45:1–24, 1989.

[59] A. Singhal. Modern information retrieval: A brief overview. In *IEEE Computer Society Technical Committee on Data Engineering*, volume 24, 2001.

[60] R. Sokal and A. Sneath. *Principles of Numerical Taxonomy.* W.H. Freeman, San Francisco, 1963.

[61] W. Song. Web document modeling and clustering. In *CMMIS Workshop in the Intl Conf. on Entity Relationship Approaches*, 1997.

[62] W. Stuetzle. Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. *Journal of Classification*, 20(5):25–47, 2003.

[63] P. P. Talwar and P. A. Lachenbruch. Cluster analysis on age patterns of fertility. *ASA Proceedings of the Social Statistics Section*, pages 808–814, 1976.

[64] J. Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*, chapter 11. Cambridge University Press, 2004.

[65] R. C. Tryon. *Cluster Analysis.* McGraw-Hill, New York, 1939.

[66] E. M. Voorhees. Implementing agglomerative hierarchic clustering algorithms for use in document retrieval. *Information Processing and Management*, 22:465–476, 1986.

[67] P. Willett. Document clustering using an inverted file approach. *Journal of Information Science*, 2:223–231, 1980.

[68] P. Willett. Clustering tendency in chemical classifications. *Journal of Chemical Information and Computer Sciences*, 20:78–80, 1985.

[69] P. Willett. Recent trends in hierarchic document clustering: A critical review. *Information Processing and Management*, 24:577–597, 1988.

[70] W. Wright. A formalization of cluster analysis. *Pattern Recognition*, 5:273–282, 1973.

[71] A. N. Yerkey. A cluster analysis of retrieval patterns among bibliographic databases. *Journal of the American Society for Information Science*, 34:350–355, 1983.

[72] D. Yu, S. Chatterjee, and A. Zhang. Efficiently detecting arbitrary shaped clusters in very large datasets with high dimensions. *11th IEEE International Conference on Tools with Artificial Intelligence*, pages 187–194, 1999.

[73] O. Zamir. Web document clustering: A feasibility demonstration. In *SIGIR 98*, Melbourne, 1998.

[74] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD'96*, Montreal, 1996.