3

Clustering Methods and Algorithms

Cluster analysis is the process of classifying objects into subsets that have meaning in the context of a particular problem. The objects are thereby organized into an efficient representation that characterizes the population being sampled. In this chapter we present the clustering methods themselves and explain algorithms for performing cluster analysis. Section 3.1 lists the factors involved in classifying objects, and in Sections 3.2 and 3.3 we explain the two most common types of classification. Computer software for cluster analysis is described in Section 3.4. Section 3.5 outlines a methodology for using clustering algorithms to the best advantage. This chapter focuses on the act of clustering itself by concentrating on the inputs to and outputs from clustering algorithms. The need for the formal validation methods in Chapter 4 will become apparent during the discussion.

3.1 GENERAL INTRODUCTION

A clustering is a type of classification imposed on a finite set of objects. As explained in Section 2.2, the relationship between objects is represented in a proximity matrix in which rows and columns correspond to objects. If the objects are characterized as patterns, or points in a d-dimensional metric space, the proximities can be distance between pairs of points, such as Euclidean distance. Unless a meaningful measure of distance, or proximity, between pairs of objects has been established, no meaningful cluster analysis is possible. The proximity matrix is the one and only input to a clustering algorithm.

Clustering is a special kind of classification. See Kendall (1966) for discussion on the relationship between classification and clustering. Figure 3.1 shows a tree of classification problems as suggested by Lance and Williams (1967). Each leaf in the tree in Figure 3.1 defines a different genus of classification problem. The nodes in the tree of Figure 3.1 are defined below.

a. *Exclusive versus nonexclusive.* An exclusive classification is a partition of the set of objects. Each object belongs to exactly one subset, or cluster. Nonexclusive, or overlapping, classification can assign an object to several classes. For example, a grouping of people by age or sex is exclusive, whereas a grouping by disease category is nonexclusive because a person can have several diseases simultaneously. Shepard and Arabie (1979) provide a review of nonexclusive or overlapping clustering methods. This chapter treats only exclusive classification. Fuzzy clustering is a type of nonexclusive classification in which a pattern is assigned a degree of belongingness to each cluster in a partition and is explained in Section 3.3.8.

b. *Intrinsic versus extrinsic.* An intrinsic classification uses only the proximity matrix to perform the classification. Intrinsic classification is called "unsupervised learning" in pattern recognition because no category labels denoting an a priori partition of the objects are used. (See Appendix A for an introduction to pattern recognition.) Extrinsic classification uses category labels on the objects as well

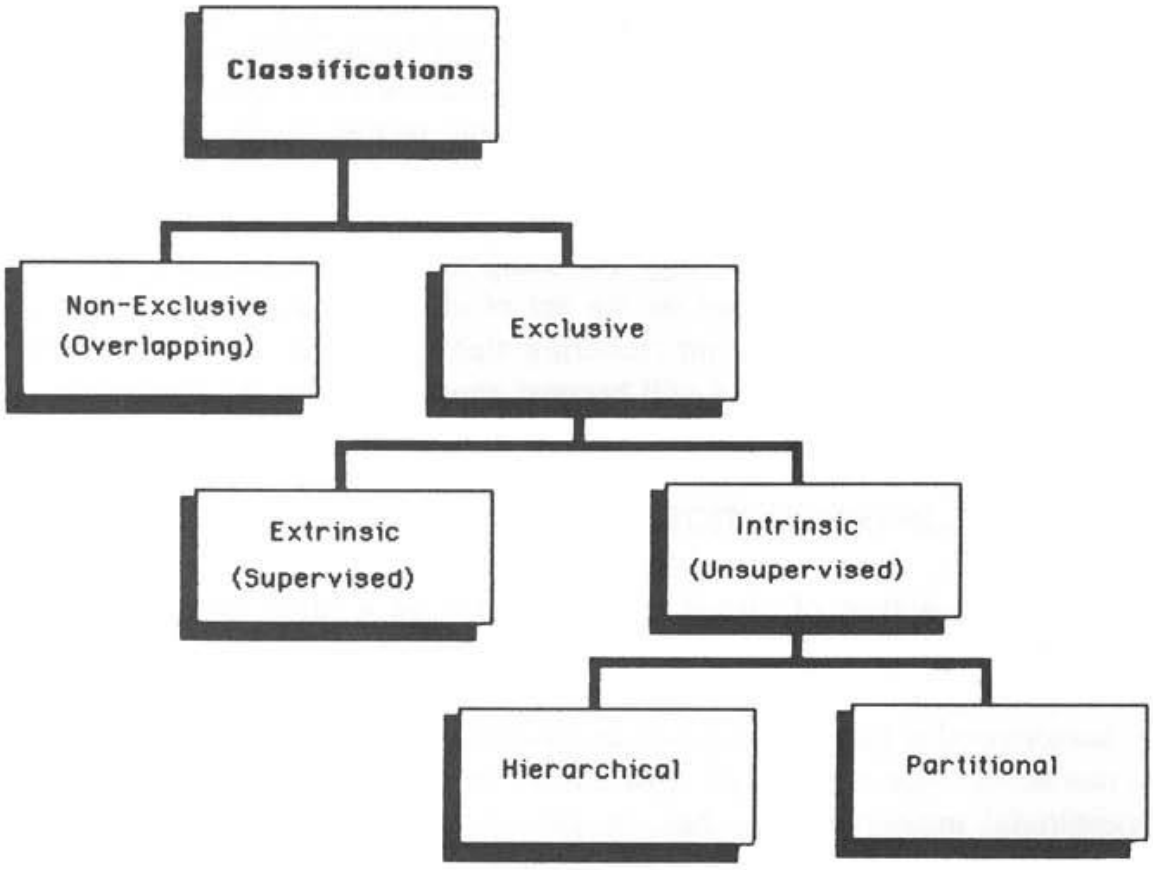


Figure 3.1 Tree of classification types.

as the proximity matrix. The problem is then to establish a discriminant surface that separates the objects according to category. In other words, an extrinsic classifier relies on a "teacher," whereas an intrinsic classifier has only the proximity matrix.

One way to evaluate an intrinsic classification is to see how the cluster labels, assigned to objects during clustering, match the category labels, assigned a priori. For example, suppose that various indices of personal health were collected from smokers and nonsmokers. An intrinsic classification would group the individuals based on similarities among the health indices and then try to determine whether smoking was a factor in the propensity of individuals toward various diseases. An extrinsic classification would study ways of discriminating smokers from nonsmokers based on health indices. We are concerned only with intrinsic classification in this book; intrinsic classification is the essence of cluster analysis.

c. *Hierarchical versus partitional.* Exclusive, intrinsic classifications are subdivided into hierarchical and partitional classifications by the type of structure imposed on the data. A hierarchical classification is a nested sequence of partitions and is explained in Section 3.2, whereas a partitional classification is a single partition and is defined in Section 3.3. Thus a hierarchical classification is a special sequence of partitional classifications. We will use the term *clustering* for an exclusive, intrinsic, partitional classification and the term *hierarchical clustering* for an exclusive, intrinsic, hierarchical classification. Sneath and Sokal (1973) apply the acronym SAHN (Sequential, Agglomerative, Hierarchical, Nonoverlapping) to exclusive, intrinsic, hierarchical, agglomerative algorithms. The differences and similarities between algorithms for generating these two types of classifications are the topics of this chapter.

Several algorithms can be proposed to express the same exclusive, intrinsic classification. One frequently uses an algorithm to express a clustering method, then examines various computer implementations of the method. The primary algorithmic options in common use are explained below.

1. *Agglomerative versus divisive.* An agglomerative, hierarchical classification places each object in its own cluster and gradually merges these atomic clusters into larger and larger clusters until all objects are in a single cluster. Divisive, hierarchical classification reverses the process by starting with all objects in one cluster and subdividing into smaller pieces. Thus this option corresponds to a choice of procedure rather than to a different kind of classification. Partitional classification can be characterized in the same way. A single partition can be established by gluing together small clusters (agglomerative) or by fragmenting a single all-inclusive cluster (divisive).
2. *Serial versus simultaneous.* Serial procedures handle the patterns one by one, whereas simultaneous classification works with the entire set of patterns at the same time (see Clifford and Stephenson, 1975).
3. *Monothetic versus polythetic.* This option is most applicable to problems in taxonomy, where the objects to be clustered are represented as patterns, or

points in a space. A monothetic clustering algorithm uses the features one by one, whereas, a polythetic procedure uses all the features at once. For example, a different feature can be used to form each partition in a hierarchical classification under a monothetic algorithm. We will consider only polythetic algorithms.

4. *Graph theory versus matrix algebra.* What is the appropriate mathematical formalism for expressing a clustering algorithm? We will express some algorithms in terms of graph theory, using properties such as connectedness and completeness to define classifications, and express other algorithms in terms of algebraic constructs, such as mean-square-error. The choice is one of clarity, convenience, and personal choice. When implementing an algorithm on a computer, attention must be paid to questions of computational efficiency. This issue is not related to human understanding of the classification method. Some algorithms have convenient expressions under both options.

3.2 HIERARCHICAL CLUSTERING

A hierarchical clustering method is a procedure for transforming a proximity matrix into a sequence of nested partitions. A hierarchical clustering algorithm is the specification of steps for performing a hierarchical clustering. It is often convenient to characterize a hierarchical clustering method by writing down an algorithm, but the algorithm should be separated from the method itself. In addition to defining algorithms and methods in this section, we define the type of mathematical structure a hierarchical clustering imposes on data and describe ways of viewing that structure.

First comes the notion of a sequence of nested partitions. The n objects to be clustered are denoted by the set $\mathscr{X}$.

$$\mathscr{X} = \{x_1, x_2, \ldots, x_n\}$$

where x_i is the ith object. A partition, $\mathscr{C}$, of $\mathscr{X}$ breaks $\mathscr{X}$ into subsets $\{C_1, C_2, \ldots, C_m\}$ satisfying the following:

$$C_i \cap C_j = \Phi \qquad \text{for } i \text{ and } j \text{ from 1 to } m, \quad i \neq j$$

$$C_1 \cup C_2 \cup \ldots \cup C_m = \mathscr{X}$$

In this notation, "$\cap$" stands for set intersection, "$\cup$" stands for set union, and Φ is the empty set. A clustering is a partition; the components of the partition are called clusters. Partition $\mathscr{B}$ is nested into partition $\mathscr{C}$ if every component of $\mathscr{B}$ is a proper subset of a component of $\mathscr{C}$. That is, $\mathscr{C}$ is formed by merging components of $\mathscr{B}$. For example, if the clustering $\mathscr{C}$ with three clusters and the clustering $\mathscr{B}$ with five clusters are defined as follows, then $\mathscr{B}$ is nested into $\mathscr{C}$. Both $\mathscr{C}$ and $\mathscr{B}$ are clusterings of the set of objects $\{x_1, x_2, \ldots, x_{10}\}$.

$$\mathscr{C} = \{(x_1, x_3, x_5, x_7), (x_2, x_4, x_6, x_8), (x_9, x_{10})\}$$

$$\mathscr{B} = \{(x_1, x_3), (x_5, x_7), (x_2), (x_4, x_6, x_8), (x_9, x_{10})\}$$

Neither $\mathscr{C}$ nor $\mathscr{B}$ is nested into the following partition, and this partition is not nested into $\mathscr{C}$ or $\mathscr{B}$.

$$\{(x_1, x_2, x_3, x_4), (x_5, x_6, x_7, x_8), (x_9, x_{10})\}$$

A hierarchical clustering is a sequence of partitions in which each partition is nested into the next partition in the sequence. An *agglomerative* algorithm for hierarchical clustering starts with the disjoint clustering, which places each of the n objects in an individual cluster. The clustering algorithm being employed dictates how the proximity matrix should be interpreted to merge two or more of these trivial clusters, thus nesting the trivial clustering into a second partition. The process is repeated to form a sequence of nested clusterings in which the number of clusters decreases as the sequence progresses until a single cluster containing all n objects, called the conjoint clustering, remains. A *divisive* algorithm performs the task in the reverse order.

A picture of a hierarchical clustering is much easier for a human being to comprehend than is a list of abstract symbols. A *dendrogram* is a special type of tree structure that provides a convenient picture of a hierarchical clustering. A dendrogram consists of layers of nodes, each representing a cluster. Lines connect nodes representing clusters which are nested into one another. Cutting a dendrogram horizontally creates a clustering. Figure 3.2 provides a simple example. Section 3.2.2 explains the role of dendrograms in hierarchical clustering.

Other pictures can also be drawn to visualize a hierarchical clustering (Kleiner and Hartigan, 1981; Friedman and Rafsky, 1981; Everitt and Nicholls, 1975). Information other than the sequence in which clusterings appear will be of interest. The level, or proximity value, at which a clustering is formed can also be recorded. If objects are represented as patterns, or points in a space, the centroids of the clusters can be important, as well as the spreads of the clusters.

Two specific hierarchical clustering methods are now defined called the *single-link* and the *complete-link* methods. Section 3.2.1 explains algorithms for these two commonly used hierarchical clustering methods. The sequences of clusterings created by these two methods depend on the proximities only through their rank

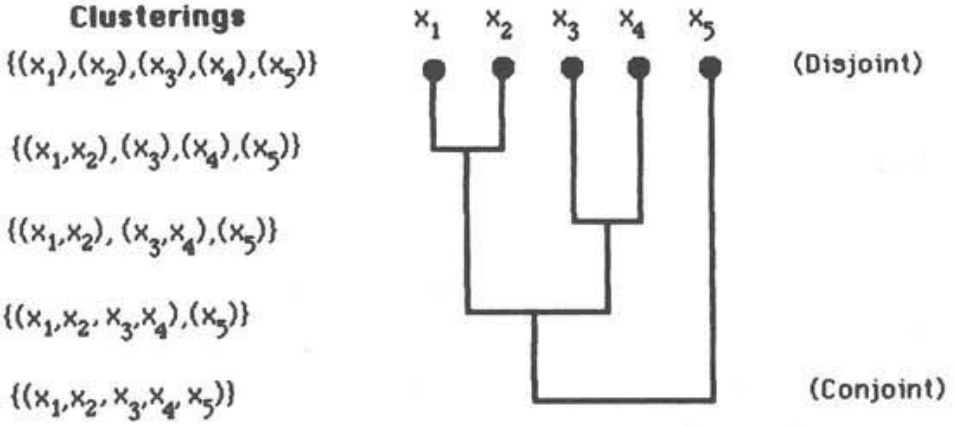


Figure 3.2 Example of dendrogram.

order. Thus we first assume an ordinal scale for the proximities and use graph theory to express algorithms. Single-link and complete-link hierarchical methods are not limited to ordinal data. Sections 3.2.4 and 3.2.5 examine algorithms for these two methods in terms of interval and ratio data. The effects of proximity ties on hierarchical clustering are discussed in Section 3.2.6, while algorithms defined for single-link and complete-link clustering are generalized in Sections 3.2.7 and 3.2.9 to establish new clustering methods. The issues in determining whether or not a hierarchical classification is appropriate for a given proximity matrix are postponed until Chapter 4.

3.2.1 Single-Link and Complete-Link Algorithms from Graph Theory

We begin with a symmetric $n \times n$ proximity matrix $\mathcal{D} = [d(i, j)]$, as defined in Section 2.2. The $n(n-1)/2$ entries on one side of the main diagonal are assumed to contain a permutation of the integers from 1 to $n(n-1)/2$ with no ties. That is, the proximities are on an ordinal scale. We take the proximities to be dissimilarities; $d(1, 2) > d(1, 3)$ means that objects 1 and 3 are more like one another than are objects 1 and 2.

Example 3.1

An example of an ordinal proximity matrix for $n = 5$ is given as matrix $\mathcal{D}_1$.

$$\mathcal{D}_1 = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \\ \begin{bmatrix} 0 & 6 & 8 & 2 & 7 \\ 6 & 0 & 1 & 5 & 3 \\ 8 & 1 & 0 & 10 & 9 \\ 2 & 5 & 10 & 0 & 4 \\ 7 & 3 & 9 & 4 & 0 \end{bmatrix} \end{array}$$

A *threshold graph* is an undirected, unweighted graph on n nodes without self-loops or multiple edges. Each node represents an object. See Appendix G for a brief review of terms in graph theory. A threshold graph $G(v)$ is defined for each dissimilarity level v by inserting an edge (i, j) between nodes i and j if objects i and j are less dissimilar than v. That is,

$$(i, j) \in G(v) \qquad \text{if and only if} \quad d(i, j) \leq v$$

As discussed in Section 2.2, we assume that $d(i, i) = 0$ for all i. Thus $G(v)$ defines a binary relation for any real number v that is reflexive and symmetric. A binary relation is a subset of the product set $\mathcal{X} \times \mathcal{X}$, where $\mathcal{X}$ is the set of objects. Objects x_i and x_j are "related" if their dissimilarity is below the threshold v. Reflexive, symmetric binary relations are pictured in a natural fashion by a threshold graph. Figure 3.3 shows the binary relation obtained from proximity matrix $\mathcal{D}_1$ above for a threshold of 5. The symbol "*" in position (i, j) of the matrix means that the pair (x_i, x_j) belongs to the binary relation.

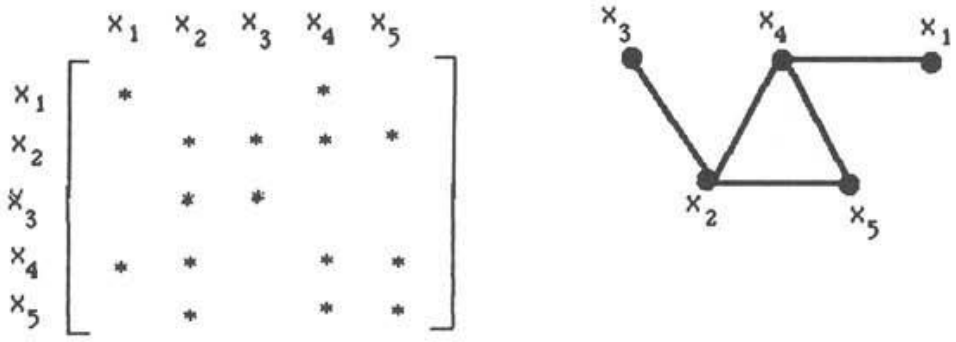


Figure 3.3 Binary relation and threshold graph for threshold 5.

Simple algorithms for the single-link and complete-link clustering methods based on threshold graphs are listed below. These algorithms should help one conceptualize the way in which the two hierarchies are formed and can easily be applied to small problems. Other algorithms are given later in this chapter that are appropriate for computer implementation. Both algorithms assume an ordinal dissimilarity matrix containing no tied entries and produce a nested sequence of clusterings that can be pictured on a dendrogram.

AGGLOMERATIVE ALGORITHM FOR SINGLE-LINK CLUSTERING

Step 1. Begin with the disjoint clustering implied by threshold graph $G(0)$, which contains no edges and which places every object in a unique cluster, as the current clustering. Set $k \leftarrow 1$.

Step 2. Form threshold graph $G(k)$.

If the number of components (maximally connected subgraphs) in $G(k)$ is less than the number of clusters in the current clustering, redefine the current clustering by naming each component of $G(k)$ as a cluster.

Step 3. If $G(k)$ consists of a single connected graph, stop. Else, set $k \leftarrow k + 1$ and go to step 2.

AGGLOMERATIVE ALGORITHM FOR COMPLETE-LINK CLUSTERING

Step 1. Begin with the disjoint clustering implied by threshold graph $G(0)$, which contains no edges and which places every object in a unique cluster, as the current clustering. Set $k \leftarrow 1$.

Step 2. Form threshold graph $G(k)$.

If two of the current clusters form a clique (maximally complete subgraph) in $G(k)$, redefine the current clustering by merging these two clusters into a single cluster.

Step 3. If $k = n(n - 1)/2$, so that $G(k)$ is the complete graph on the n nodes, stop. Else, set $k \leftarrow k + 1$ and go to step 2.

These algorithms can be extended to dissimilarity matrices on interval and ratio scales as long as no entries are tied. Simply view $G(k)$ as the threshold graph containing edges corresponding to the k smallest dissimilarities. A *threshold dendrogram* records the clusterings in the order in which they are formed, irrespective of the dissimilarity level at which the clusterings first appear. A *proximity dendrogram* lists the dissimilarity level at which each clustering forms and, in effect, is a nonlinear transformation of the scale used with a threshold dendrogram. Examples of proximity dendrograms are given in Section 3.2.2.

The single-link clustering on $G(v)$ is defined in terms of connected subgraphs in $G(v)$; the complete-link clustering uses complete subgraphs. However, not all maximally complete subgraphs in a threshold graph need be complete-link clusters. The order in which the clusters are formed is crucial. Figure 3.4 exhibits the single-link and complete-link hierarchical clusterings for the proximity matrix $\mathcal{D}_1$ of Example 3.1. The first seven threshold graphs in the sequence of 10 threshold graphs are shown with nodes labeled so that node j denotes object x_j.

Please note the following peculiarities about forming hierarchical clusterings from threshold graphs. The entire single-link hierarchy is defined by the first

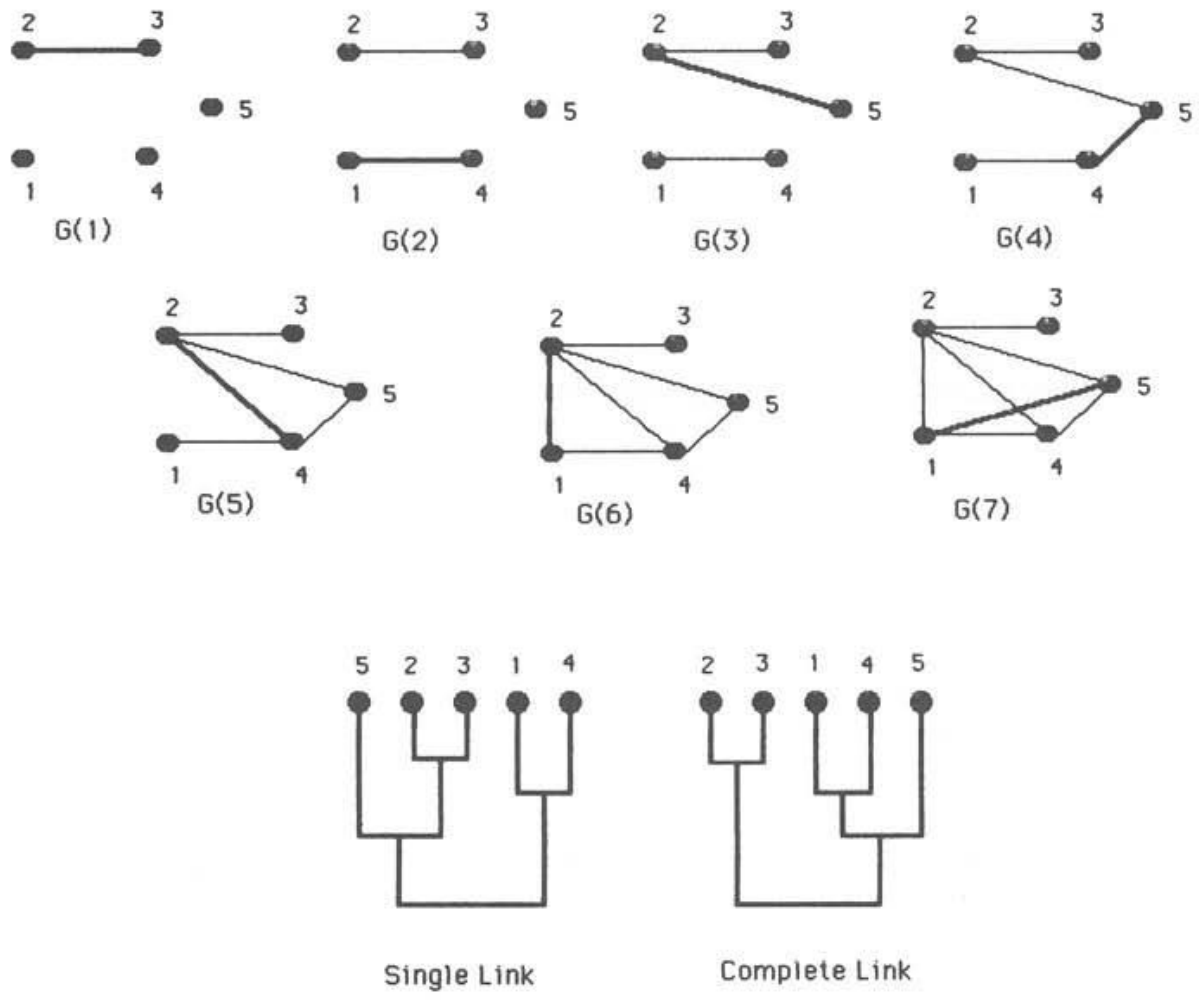


Figure 3.4 Threshold graphs and dendrograms for single-link and complete-link hierarchical clusterings.

four threshold graphs in Figure 3.4. However, the first seven threshold graphs are needed to determine the complete-link hierarchy. Once the two-cluster complete-link clustering has been obtained, no more explicit threshold graphs need be drawn because the two clusters will merge into the conjoint clustering only when all $n(n-1)/2$ edges have been inserted. This example demonstrates the significance of nesting in the hierarchy. Objects $\{x_2, x_5, x_4\}$ form a clique, or maximally complete subgraph, in threshold graph $G(5)$, but the three objects are not a complete-link cluster. Once complete-link clusters $\{x_2, x_3\}$ and $\{x_1, x_4\}$ have been established, object x_5 must merge with one of the two established clusters; once formed, clusters cannot be dissolved and clusters cannot overlap. The dendrograms themselves are drawn with each clustering shown on a separate level, even though, for example, the two-cluster single-link clustering is obtained from $G(3)$ and the two-cluster complete-link clustering is obtained from $G(7)$.

The interpretation of the dendrograms is not under consideration in this chapter, but the two dendrograms in Figure 3.4 do raise a question about object x_5. Does it belong to the cluster $\{x_2, x_3\}$ or to the cluster $\{x_1, x_4\}$? A case can also be made for calling $\{x_2, x_4, x_5\}$ a cluster. Perhaps a hierarchical structure is not appropriate for this proximity matrix. These issues are examined in Chapter 4.

Hubert (1974a) provides the following algorithms for generating hierarchical clusterings by the single-link and complete-link methods. When the proximity matrix contains no ties, clusterings are numbered $0, 1, \ldots, (n-1)$ and the mth clustering, $\mathscr{C}_m$, contains $n - m$ clusters.

$$\mathscr{C}_m = \{C_{m1}, C_{m2}, \ldots, C_{m(n-m)}\}$$

HUBERT'S ALGORITHM FOR SINGLE-LINK AND COMPLETE-LINK METHODS

Step 1. Set $m \leftarrow 0$. Form the disjoint clustering with clustering number m.

$$\mathscr{C}_0 = \{(x_1), (x_2), \ldots, (x_n)\}$$

Step 2a. To find the next clustering (with clustering number $m + 1$) by the *single-link* method, define the function Q_s for all pairs (r, t) of clusters in the current clustering as follows.

$$Q_s(r, t) = \min \{d(i, j) : \text{the maximal subgraph of } G(d(i, j)) \text{ defined by } C_{mr} \cup C_{mt} \text{ is connected}\}$$

Clusters C_{mp} and C_{mq} are merged to form the next clustering in the single-link hierarchy if

$$Q_s(p, q) = \min \{Q_s(r, t)\}$$

Step 2b. The function Q_c is used to find clustering number $m + 1$ by the *complete-link* method and is defined for all pairs (r, t) of clusters in the current clustering.

$$Q_c(r, t) = \min \{d(i, j) : \text{the maximal subgraph of } G(d(i, j)) \text{ defined by } C_{mr} \cup C_{mt} \text{ is complete}\}$$

Cluster C_{mp} is merged with cluster C_{mq} under the complete-link method if

$$Q_c(p, q) = \min \{Q_c(r, t)\}$$

Step 3. Set $m \leftarrow m + 1$ and repeat step 2. Continue until all objects are in a single cluster.

The word "maximal" in the definitions of functions Q_c and Q_s means that all nodes of the two clusters C_{mr} and C_{mt} must be considered when establishing connectedness or completeness. Only existing clusters can be merged at the next level.

Example 3.2

One way to understand the functions Q_s and Q_c is to consider the sequence of threshold graphs even though the threshold graphs are not necessary to the evaluation of these functions. For example, the first seven threshold graphs for the proximity matrix $\mathcal{D}_1$ (Example 3.1) are given in Figure 3.4. The third clustering ($m = 2$) can be numbered as follows.

$$\mathcal{C}_2 = \{C_{21}, C_{22}, C_{23}\}$$

The three clusters are defined as

$$C_{21} = \{x_5\}, C_{22} = \{x_2, x_3\}, C_{23} = \{x_1, x_4\}$$

To evaluate Q_s when m is 2, find the smallest proximity that will connect two of the existing clusters. Clusters C_{21} and C_{22} become connected in threshold graph $G(3)$. Therefore, (p, q) is (1, 2) and $Q_s(p, q)$ is 3. Another way of understanding this function is to realize that $Q_s(r, t)$ is the smallest dissimilarity that connects clusters C_{mr} and C_{mt} and the smallest of the dissimilarities so found defines the next clustering. In this case, clusters C_{21} and C_{22} first connect at level 3, or in $G(3)$, clusters C_{21} and C_{23} first connect in $G(4)$ and clusters C_{22} and C_{23} first form a connected subgraph in $G(5)$. The minimum of the levels (3, 4, 5) is 3.

The interpretation of Q_c is much the same, with completeness replacing connectedness. For example, Q_c is found when $m = 2$ by searching the threshold graphs in sequence until one is found that merges existing clusters from $\mathcal{C}_2$ into a complete subgraph. This does not happen until $G(7)$, so (p, q) is (1, 3) and $Q_c(p, q)$ is 7. Clusters C_{21} and C_{22} first form a complete subgraph in threshold graph $G(9)$. Clusters C_{21} and C_{23} first merge into a complete subgraph in $G(7)$ and clusters C_{22} and C_{23} first form a complete subgraph in $G(10)$. The minimum of the levels (7, 9, 10) is 7, so the fourth complete link clustering ($m = 3$) is achieved at threshold 7 and merges clusters C_{21} and C_{23}. The fact that other complete subgraphs are formed in the process, such as $\{x_2, x_4, x_5\}$, is immaterial.

Single-link clusters are characterized as maximally connected subgraphs, whereas complete-link clusters are cliques, or maximally complete subgraphs. Jardine and Sibson (1971) have demonstrated several desirable theoretical properties of single-link clusterings, but several authors (e.g., Wishart, 1969; Hubert, 1974b)

have objected to certain practical difficulties with clusters formed by the single-link method. For example, single-link clusters easily chain together and are often "straggly." Only a single edge between two large clusters is needed to merge the clusters. On the other hand, complete-link clusters are conservative. All pairs of objects must be related before the objects can form a complete-link cluster. Completeness is a much stronger property than connectedness. Perceived deficiencies in these two clustering methods have led to a large number of alternatives, some of which are explained in Sections 3.2.7 and 3.2.9. For example, Hansen and DeLattre (1978) noted that single-link clusters may chain and have little homogeneity, while complete-link clusters may not be well separated.

Every connected subgraph of a threshold graph is a single-link cluster but not every clique is a complete-link cluster. Peay (1975) proposed an exclusive, overlapping, hierarchical clustering method based on cliques and extended it to asymmetric proximity matrices. Matula (1977) noted that the number of possible cliques is huge, so clustering based on cliques is practical only for small n.

Suppose that the latest clustering of $\{x_1, x_2, \ldots, x_n\}$ in one of the hierarchies has been formed by merging clusters C_{mp} and C_{mq} in the clustering

$$\{C_{m1}, C_{m2}, \ldots, C_{m(n-m)}\}$$

The following characterizations may help to distinguish the two clustering methods. If the clustering was by the single-link method, we would know that

$$\min_{x_i \in C_{mp}, x_j \in C_{mq}} \{d(i, j)\} = \min_{r \neq s} \{ \min_{x_i \in C_{mr}, x_j \in C_{ms}} \{d(i, j)\}\}$$

If the clustering was by the complete-link method, we have that

$$\max_{x_i \in C_{mp}, x_j \in C_{mq}} \{d(i, j)\} = \min_{r \neq s} \{ \max_{x_i \in C_{mr}, x_j \in C_{ms}} \{d(i, j)\}\}$$

These characterizations show why the single-link method has been called the "minimum" method and the complete-link method has been named the "maximum" method (Johnson, 1967). However, if the proximities are similarities instead of dissimilarities, this terminology would be confusing. This characterization also explains why the complete-link method is referred to as the "diameter" method. The diameter of a complete subgraph is the largest proximity among all proximities for pairs of objects in the subgraph. Although the complete-link method does not generate clusters with minimum diameter, the diameter of a complete-link cluster is known to equal the level at which the cluster is formed. By contrast, single-link clusters are based on connectedness and are characterized by minimum path length among all pairs of objects in the cluster.

3.2.2 Dendrograms and Recovered Structure

An important objective of hierarchical cluster analysis is to provide a picture of the data that can easily be interpreted, such as the dendrograms in Figure 3.4. Dendrograms list the clusterings one after another. Cutting a dendrogram at any

level defines a clustering and identifies clusters. The level itself has no meaning in terms of the scale of the proximity matrix.

A *proximity graph* is a threshold graph in which each edge is weighted according to its proximity. The proximities used in Section 3.2.1 are ordinal, so the weights are integers from 1 to $n(n-1)/2$. The dendrogram drawn from a proximity graph is called a *proximity dendrogram* and records both the clusterings and the proximities at which they are formed. Proximity dendrograms are especially useful when the proximities are on an interval or ratio scale.

Example 3.3

A ratio proximity matrix is given below as $\mathcal{D}_2$. The threshold and proximity dendrograms are given in Figure 3.5. Also shown is the sequence of proximity graphs which provides the actual dissimilarity values at which clusters are formed.

$$\mathcal{D}_2 = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_2 & x_3 & x_4 & x_5 \end{array} \\ \begin{bmatrix} 5.8 & 4.2 & 6.9 & 2.6 \\ & 6.7 & 1.7 & 7.2 \\ & & 1.9 & 5.6 \\ & & & 7.6 \end{bmatrix} \end{array}$$

A proximity dendrogram is drawn on a proximity scale from a sequence of proximity graphs and highlights clusters that are "born" early and "last" a long time in the dendrogram. These observations are the basis for formal measures of cluster validity in Chapter 4.

Any hierarchical clustering algorithm can be seen as a way of transforming a proximity matrix into a dendrogram. Only the single-link and complete-link methods of clustering have been discussed so far, but the statement applies to hierarchical clustering methods defined in Sections 3.2.7 and 3.2.9 as well. Threshold and proximity dendrograms represent the structure that the hierarchical clustering method is imposing on the data. This imposed structure can be captured in another proximity matrix called the cophenetic matrix. The agreement between the given proximity matrix and the cophenetic matrix measures the degree to which the hierarchical clustering method captures the actual structure of the data. Formal methods for measuring this agreement are discussed in Chapter 4. Here the cophenetic matrix is defined to help explain the difference between the single-link and complete-link methods.

We begin with a hierarchical clustering:

$$\{\mathscr{C}_0, \mathscr{C}_1, \ldots, \mathscr{C}_{n-1}\}$$

where the mth clustering contains $n - m$ clusters:

$$\mathscr{C}_m = \{C_{m1}, C_{m2}, \ldots, C_{m(n-m)}\}$$

A level function, L, records the proximity at which each clustering is formed. For a threshold dendrogram $L(k) = k$, because the levels in the dendrogram are evenly spaced. In general,

$$L(m) = \min \{d(x_i, x_j) : \mathcal{C}_m \text{ is defined}\}$$

The *cophenetic proximity* measure d_C on the n objects is the level at which objects x_i and x_j are first in the same cluster.

$$d_C(i, j) = L(k_{ij})$$

where

$$k_{ij} = \min \{m : (x_i, x_j) \in C_{mq}, \text{ some } q\}$$

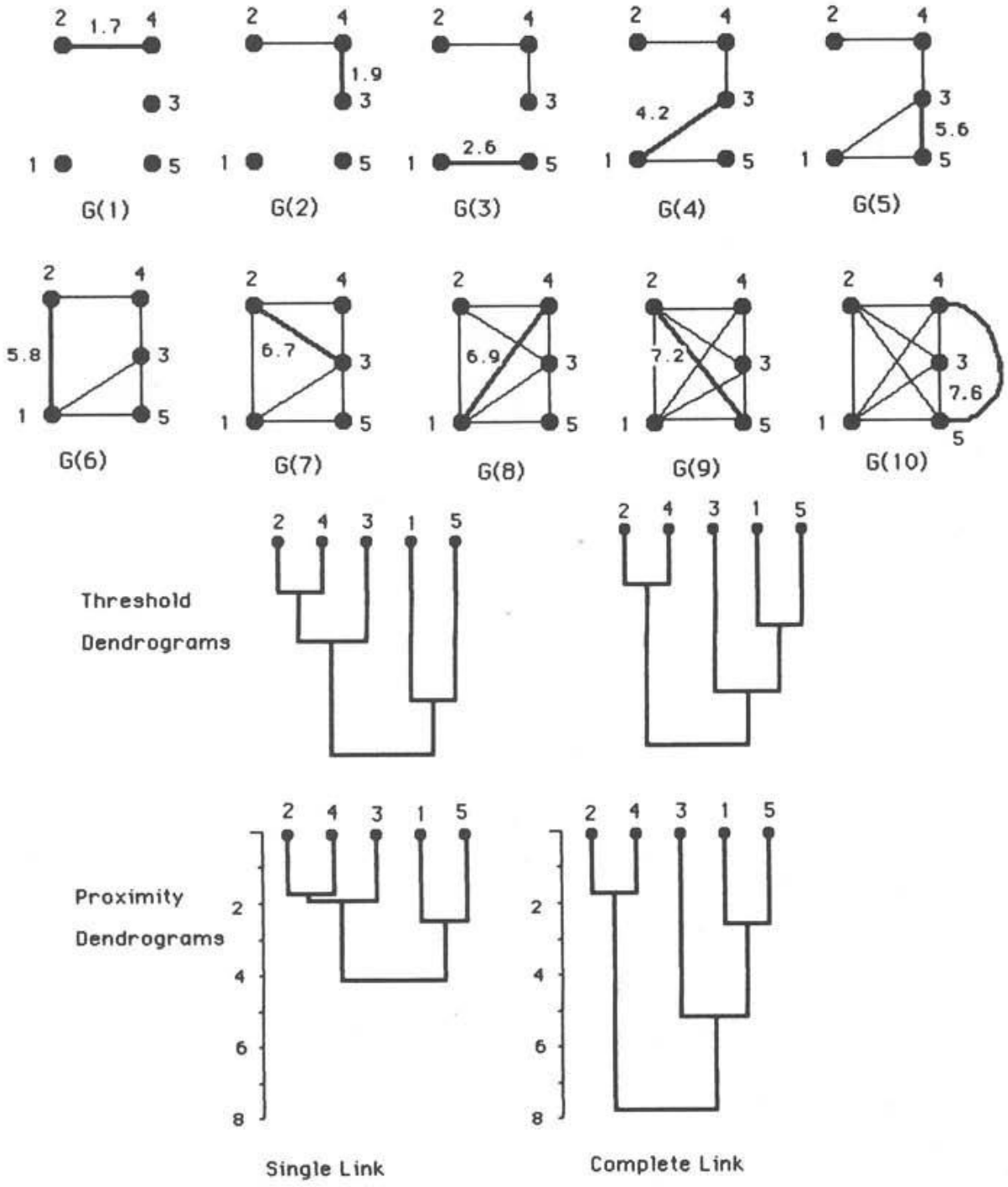


Figure 3.5 Examples of threshold and proximity graphs with corresponding dendrograms.

The matrix of values $[d_C(x_i, x_j)]$ is called the *cophenetic matrix*. The closer the cophenetic matrix and the given proximity matrix, the better the hierarchy fits the data. There can be no more than $(n - 1)$ levels in a dendrogram, so there can be no more than $(n - 1)$ distinct cophenetic proximities. Since the cophenetic matrix has $n(n - 1)/2$ entries, it must contain many ties.

The cophenetic matrix for the single-link dendrogram in Figure 3.5 is shown below as $\mathcal{D}_{Cs}$ and will be used to demonstrate some interesting properties of cophenetic matrices.

$$\mathcal{D}_{Cs} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_2 & x_3 & x_4 & x_5 \end{array} \\ \begin{bmatrix} 4.2 & 4.2 & 4.2 & 2.6 \\ & 1.9 & 1.7 & 4.2 \\ & & 1.9 & 4.2 \\ & & & 4.2 \end{bmatrix} \end{array}$$

Applying the single-link clustering method to $\mathcal{D}_{Cs}$ reproduces the single-link dendrogram in Figure 3.5. This might be expected. However, applying the complete-link method to $\mathcal{D}_{Cs}$ generates the same (single-link) dendrogram. The complete-link method is usually ambiguous when the proximity matrix contains ties, as discussed in Section 3.2.6. However, the cophenetic matrix is so arranged that tied proximities form complete subgraphs and no ambiguity occurs under complete-link clustering. A cophenetic matrix is an example of a proximity matrix with perfect hierarchical structure. Both the single-link and the complete-link methods generate exactly the same dendrogram when applied to a cophenetic matrix. Repeating this exercise by starting with the complete-link dendrogram generates the cophenetic matrix $\mathcal{D}_{Cc}$.

$$\mathcal{D}_{Cc} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_2 & x_3 & x_4 & x_5 \end{array} \\ \begin{bmatrix} 7.6 & 5.6 & 7.6 & 2.6 \\ & 7.6 & 1.7 & 7.6 \\ & & 7.6 & 5.6 \\ & & & 7.6 \end{bmatrix} \end{array}$$

The cophenetic matrix $\mathcal{D}_{Cc}$ also has perfect hierarchical structure. The complete-link and single-link clustering methods will produce exactly the same dendrogram when applied to $\mathcal{D}_{Cc}$, and that dendrogram will be identical to the complete-link dendrogram in Figure 3.5. An important question in applications is: Which dendrogram better describes the true structure of the data?

3.2.3 Hierarchical Structure and Ultrametricity

The fact that both the single-link and the complete-link methods generate exactly the same proximity dendrogram when applied to a cophenetic matrix suggests that the cophenetic matrix captures "true" or "perfect" hierarchical structure. Whether or not the hierarchical structure is appropriate for a given data set has yet to be determined, but the type of structure exemplified by the cophenetic

matrix is very special. The justification for calling the cophenetic matrix "true" hierarchical structure comes from the fact that a cophenetic proximity measure d_C defines the following equivalence relation, denoted R_C, on the set of objects:

$$R_C(a) = \{(x_i, x_j) : d_C(i, j) \leq a\}$$

Relation $R_C(a)$ can be shown to be an equivalence relation for any $a \geq 0$ by checking the three conditions necessary for an equivalence relation. Since $d_C(i, i) = 0$ for all i,

$$(x_i, x_i) \in R_C(a) \qquad \text{for all } a \geq 0$$

so $R_C(a)$ is reflexive. Since $d_C(i, j) = d_C(j, i)$ for all (i, j),

$$(x_j, x_i) \in R_C(a) \quad \text{if} \quad (x_i, x_j) \in R_C(a) \qquad \text{for all } a \geq 0$$

so $R_C(a)$ is symmetric. The final condition, transitivity, requires that for all $a \geq 0$,

$$\text{if } (x_i, x_k) \in R_C(a) \quad \text{and if} \quad (x_k, x_j) \in R_C(a), \qquad \text{then } (x_i, x_j) \in R_C(a)$$

This condition must be satisfied for all triples (x_i, x_j, x_k) of objects and all a. It can also be restated as

$$d_C(i, j) \leq \max \{d_C(i, k), d_C(k, j)\} \qquad \text{for all } (i, j, k)$$

When stated in this way, the requirement is called the *ultrametric inequality*. A close inspection of the cophenetic matrices for Figure 3.5 shows that they satisfy the ultrametric inequality, so $R_C(a)$ is, indeed, an equivalence relation for any $a \geq 0$. The nesting of the clusterings forming the hierarchy assures transitivity. The only way that the very restrictive ultrametric inequality can be satisfied is to have many ties in the cophenetic proximity. Recall that, at most, only $n - 1$ of the $n(n - 1)/2$ cophenetic proximities can be distinct. Since cophenetic proximity measures represent perfect hierarchical structure, proximity measures without ties seldom reflect true hierarchical structure. The concept of ultrametricity has been developed separately in mathematics and has applications in physics. See Rammal et al. (1986) for an excellent review of ultrametricity and Schikhof (1984) for a mathematical treatment of ultrametricity in the realm of "p-adic" analysis.

Two items should be noted with regard to the ultrametric inequality. First, the cophenetic proximities derived from single-link and complete-link clusterings always satisfy the ultrametric inequality. However, Section 3.2.7 will introduce some hierarchical clustering methods whose cophenetic proximities are not ultrametric. Second, a geometric interpretation of the ultrametric inequality demonstrates why proximities measured in applications are very seldom ultrametric. Suppose that each object is a pattern in a d-dimensional space. If Euclidean distance is the measure of proximity and if the proximity matrix is to be ultrametric, the triangles formed by *all* triples of points must be isosceles triangles with the unequal leg no longer than the two legs of equal length.

Jardine and Sibson (1971) characterize hierarchical clustering methods as

mappings from the class of proximity matrices to the class of ultrametric proximity measures. That is, a hierarchical clustering method imposes a dendrogram on the given proximity matrix and this establishes the cophenetic proximity matrix, which satisfies the ultrametric inequality. Measures of fit between proximity measures and cophenetic proximity measures are discussed in Chapter 4. The property of ultrametricity is also called *monotonicity*; a cophenetic proximity measure satisfies the ultrametric inequality only if the clusters form in a monotonic manner as dissimilarity increases. In other words, the clusterings are nested in the hierarchy. Single-link and complete-link clusterings are always monotonic, but other common clustering methods defined in Section 3.2.7 can create the next clustering at a *smaller* dissimilarity than the present one. This issue is discussed in Section 3.2.8.

3.2.4 Other Graph Theory Algorithms for Single-Link and Complete-Link

The algorithms for single-link and complete-link hierarchical clusterings described thus far establish step-by-step procedures for forming dendrograms. In this section we present other algorithms for these clustering methods that provide insight into the clustering methods and can be computationally attractive.

An algorithm for single-link clustering begins with the minimum spanning tree (MST) for $G(\infty)$, which is the proximity graph containing all $n(n-1)/2$ edges. Although the single-link hierarchy can be derived from the MST, the MST cannot be found from a single-link hierarchical clustering. For convenience, we assume that no two edges in the MST have the same weight, even though Section 3.2.6 shows that ties in proximity pose no problem with single-link clustering. An agglomerative algorithm for single-link clustering is given below that assumes a dissimilarity matrix.

GRAPH THEORY ALGORITHM FOR SINGLE-LINK CLUSTERING

Step 1. Begin with the disjoint clustering, which places each object in its own cluster. Find an MST on $G(\infty)$.

Repeat steps 2 and 3 until all objects are in one cluster.

Step 2. Merge the two clusters connected by the MST edge with the smallest weight to define the next clustering.

Step 3. Replace the weight of the edge selected in step 2 by a weight larger than the largest proximity.

This algorithm follows from the characterization for single-link clustering given in Section 3.2.1 and the definition of MST. A divisive algorithm is just as simple. Cut the edges in the MST in the order of weight, cutting the largest first. Each cut defines a new clustering, with those objects connected in the MST at any stage belonging to the same cluster. As long as no proximity ties occur,

these algorithms generate the same single-link clusterings as the algorithms presented earlier. Gower and Ross (1969) first proposed this algorithm. Rohlf (1973) provided an implementation that examines each proximity value only once.

Example 3.4

Examples of the two algorithms are given in Figure 3.6 for the proximity matrix $\mathcal{D}_3$ defined below.

$$\mathcal{D}_3 = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_2 & x_3 & x_4 & x_5 \end{array} \\ \begin{bmatrix} 2.3 & 3.4 & 1.2 & 3.7 \\ & 2.6 & 1.8 & 4.6 \\ & & 4.2 & 0.7 \\ & & & 4.4 \end{bmatrix} \end{array}$$

A node coloring of a threshold graph $G(v)$ is an assignment of "colors," or labels, to the n nodes in such a way that no two nodes connected by an edge

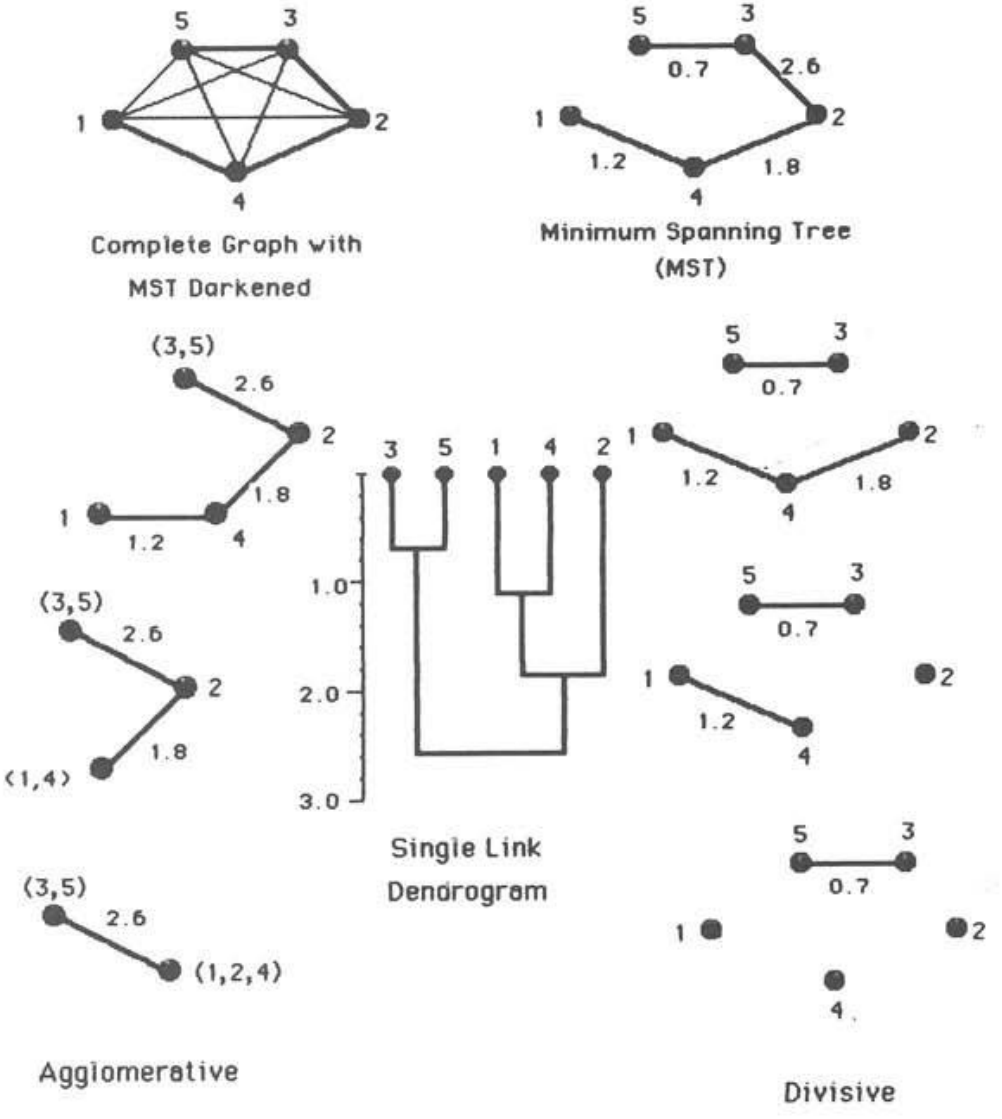


Figure 3.6 Examples of agglomerative and divisive single-link algorithms based on the MST.

in $G(v)$ are colored the same. Baker and Hubert (1976) show how the set of node colorings is related to hierarchical clustering. The connection between node coloring and complete-link clustering is not as simple as is the relation between single-link clustering and the MST. The last complete-link clustering achieved for a given threshold graph $G(v)$ corresponds to a coloring of the nodes of the complement of $G(v)$. Hansen and DeLattre (1978) provide other algorithms from graph coloring.

3.2.5 Matrix Updating Algorithms for Single-Link and Complete-Link

In this section we discuss algorithms for single-link and complete-link clustering in terms of a scheme for updating the proximity matrix. This approach was suggested by King (1967) and popularized by Johnson (1967), who formalized the procedure. The algorithm is an agglomerative scheme that erases rows and columns in the proximity matrix as old clusters are merged into new ones. We again simplify the algorithm by assuming no ties in the proximity matrix. Figure 3.7 provides examples of this algorithm for the proximity matrix $\mathcal{D}_3$ (Example 3.4).

The $n \times n$ proximity matrix is $\mathcal{D} = [d(i, j)]$. The clusterings are assigned sequence numbers 0, 1, . . . , $(n - 1)$ and $L(k)$ is the level of the kth clustering. A cluster with sequence number m is denoted (m) and the proximity between clusters (r) and (s) is denoted $d[(r), (s)]$.

JOHNSON'S ALGORITHM FOR SINGLE-LINK AND COMPLETE-LINK CLUSTERING

Step 1. Begin with the disjoint clustering having level $L(0) = 0$ and sequence number $m = 0$.

Step 2. Find the least dissimilar pair of clusters in the current clustering, say pair $\{(r), (s)\}$, according to

$$d[(r), (s)] = \min \{d[(i), (j)]\}$$

where the minimum is over all pairs of clusters in the current clustering.

Step 3. Increment the sequence number: $m \leftarrow m + 1$. Merge clusters (r) and (s) into a single cluster to form the next clustering m. Set the level of this clustering to

$$L(m) = d[(r), (s)]$$

Step 4. Update the proximity matrix, $\mathcal{D}$, by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column corresponding to the newly formed cluster. The proximity between the new cluster, denoted (r, s) and old cluster (k) is defined as follows. For the single-link method,

$$d[(k), (r, s)] = \min \{d[(k), (r)], d[(k), (s)]\}$$

For the complete-link method,

$$d[(k), (r, s)] = \max \{d[(k), (r)], d[(k), (s)]\}$$

Step 5. If all objects are in one cluster, stop. Else, go to step 2.

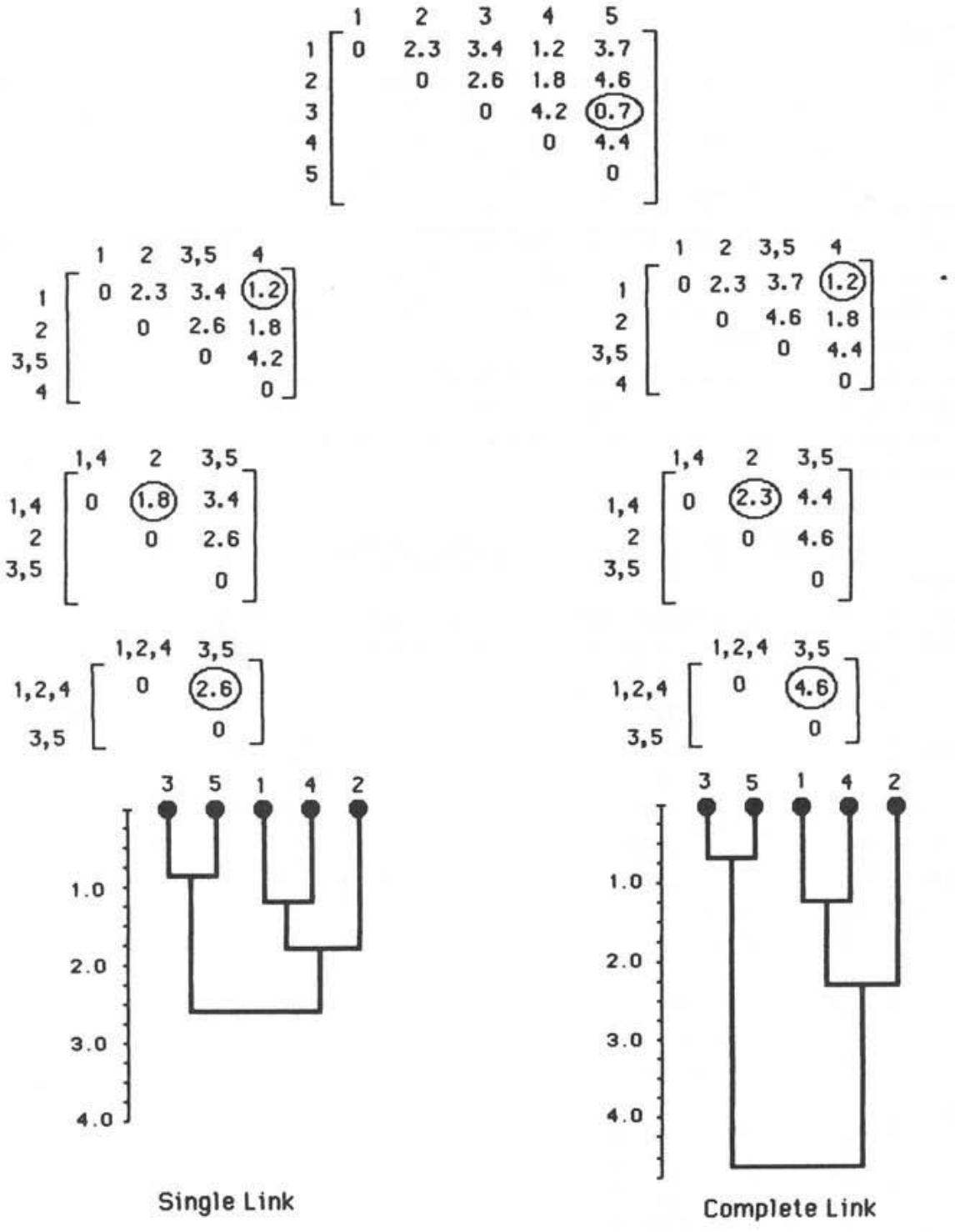


Figure 3.7 Examples of matrix updating algorithms for single-link and complete-link clusterings.

Anderberg (1973) discusses three computational approaches to implementing the algorithm above, called the stored matrix, sorted matrix, and stored data approaches. The three approaches differ as to whether the pattern matrix or the dissimilarity matrix is stored in random access memory or in auxiliary storage, such as disk. Note that the dissimilarity matrix requires more storage than the pattern matrix when $n \gg d$. The stored matrix approach, where the entire dissimilarity matrix is stored in random access memory, is fastest.

Example 3.5

The computational examples in Figure 3.7 demonstrate the construction of single-link and complete-link hierarchies. This example demonstrates the qualitative differences between the single-link and complete-link hierarchies for the two artificial data sets defined in Section 2.4. The first data set, called DATA1, consists of 100 patterns in a four-dimensional pattern space generated so as to have four categories, or true clusters. Patterns 1 through 24 were generated from category 1, patterns 25 through 59 from category 2, patterns 60 through 80 from category 3, and patterns 81 through 100 were generated in category 4. An eigenvector projection is given in Figure 2.9. The proximity measure is squared Euclidean distance in the pattern space. Since the proximity measure is Euclidean distance and since the data were generated to several decimal places on a computer, we feel safe in assuming that no proximity ties exist, so the hierarchies are both unique (see Section 3.2.6). Figures 3.8 and 3.9 show the proximity dendrograms for the single-link and complete-link hierarchies, respectively.

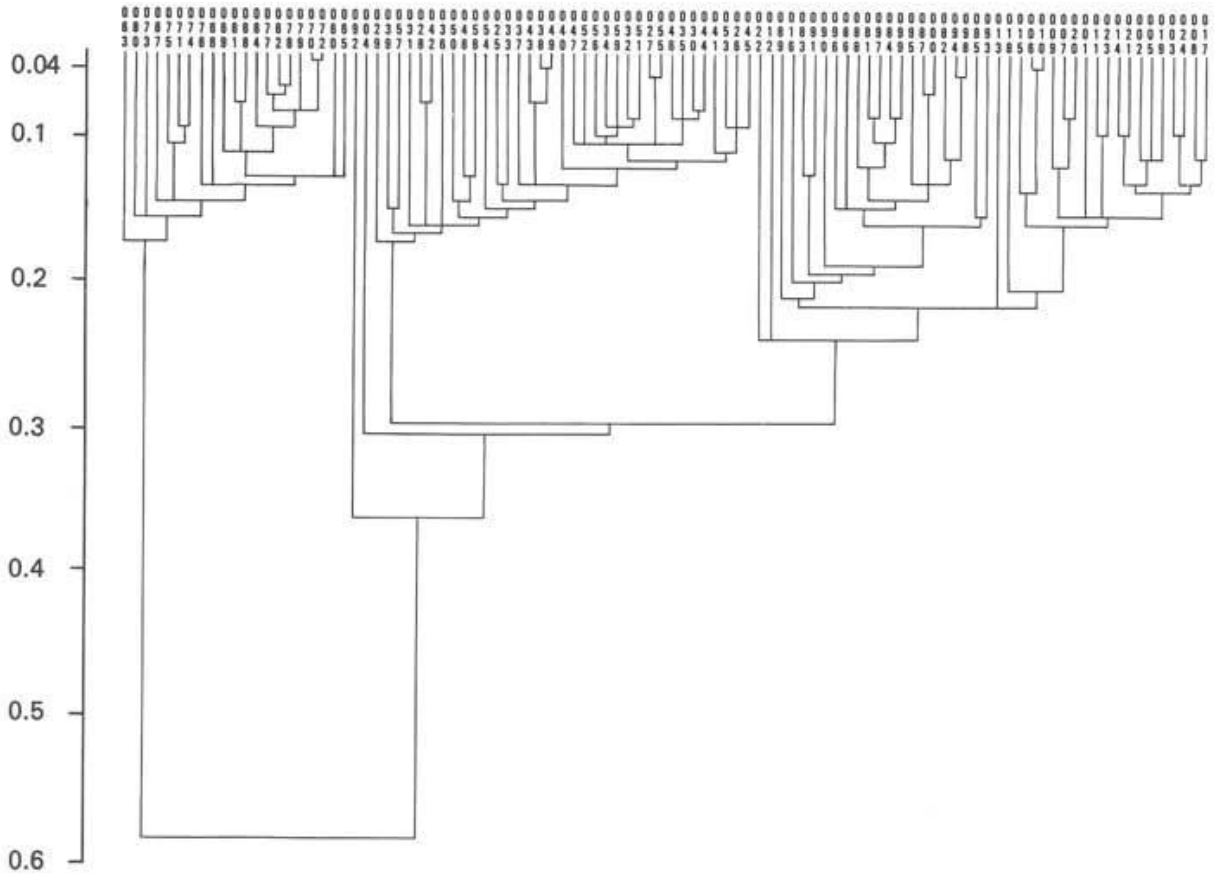


Figure 3.8 Single-link hierarchy for 100 clustered patterns in four dimensions.

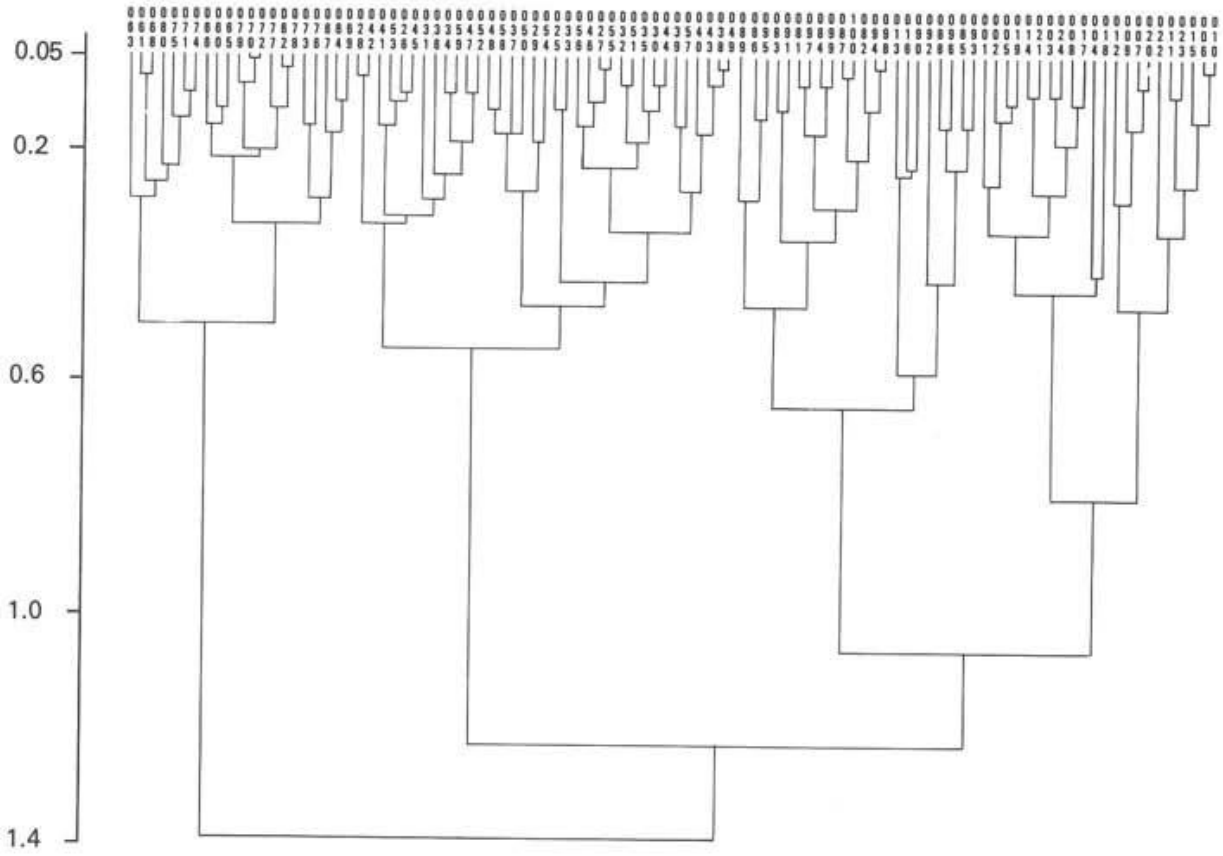


Figure 3.9 Complete-link hierarchy for 100 clustered patterns in four dimensions.

This example demonstrates the difficulty in comparing two dendrograms and motivates the development of methods for automatically isolating significant clusters that are presented in Chapter 4. The complete-link dendrogram in Figure 3.9 can be cut at level 1.0 to generate four clusters. These clusters recover the original four categories in the data perfectly. The four-category structure is not at all apparent in the single-link hierarchical clustering of Figure 3.8.

Clustering methods have the nasty habit of creating clusters in data even when no natural clusters exist, so hierarchies and clusterings must be viewed with extreme suspicion. Figures 3.10 and 3.11 demonstrate this statement on the two hierarchies for a data set, called DATA2, consisting of 100 points uniformly distributed over a unit hypercube in six dimensions (see Section 2.4). The patterns are positioned at random, so it is barely possible that they have arranged themselves into meaningful clusters; however, it is unlikely that real clusters exist, especially considering the two-dimensional projections in Figure 2.11. We thus interpret Figures 3.10 and 3.11 as hierarchies in which no true clusters exist. The single-link dendrogram in Figure 3.10 exhibits the chaining that is characteristic of single-link hierarchies. This chaining can occur even when valid clusters exist, as in Figure 3.8. The complete-link hierarchy in Figure 3.11 suggests some meaningful clusters; it looks more clustered than the single-link hierarchy, and this is the lure of complete-link clustering. It tends to produce dendrograms that form small clusters which combine nicely into larger clusters even when such a hierarchy is not warranted, as with random data. This example should demonstrate the difficulties inherent in letting the human eye scan over the dendrogram to pick out believable clusters and clusterings.

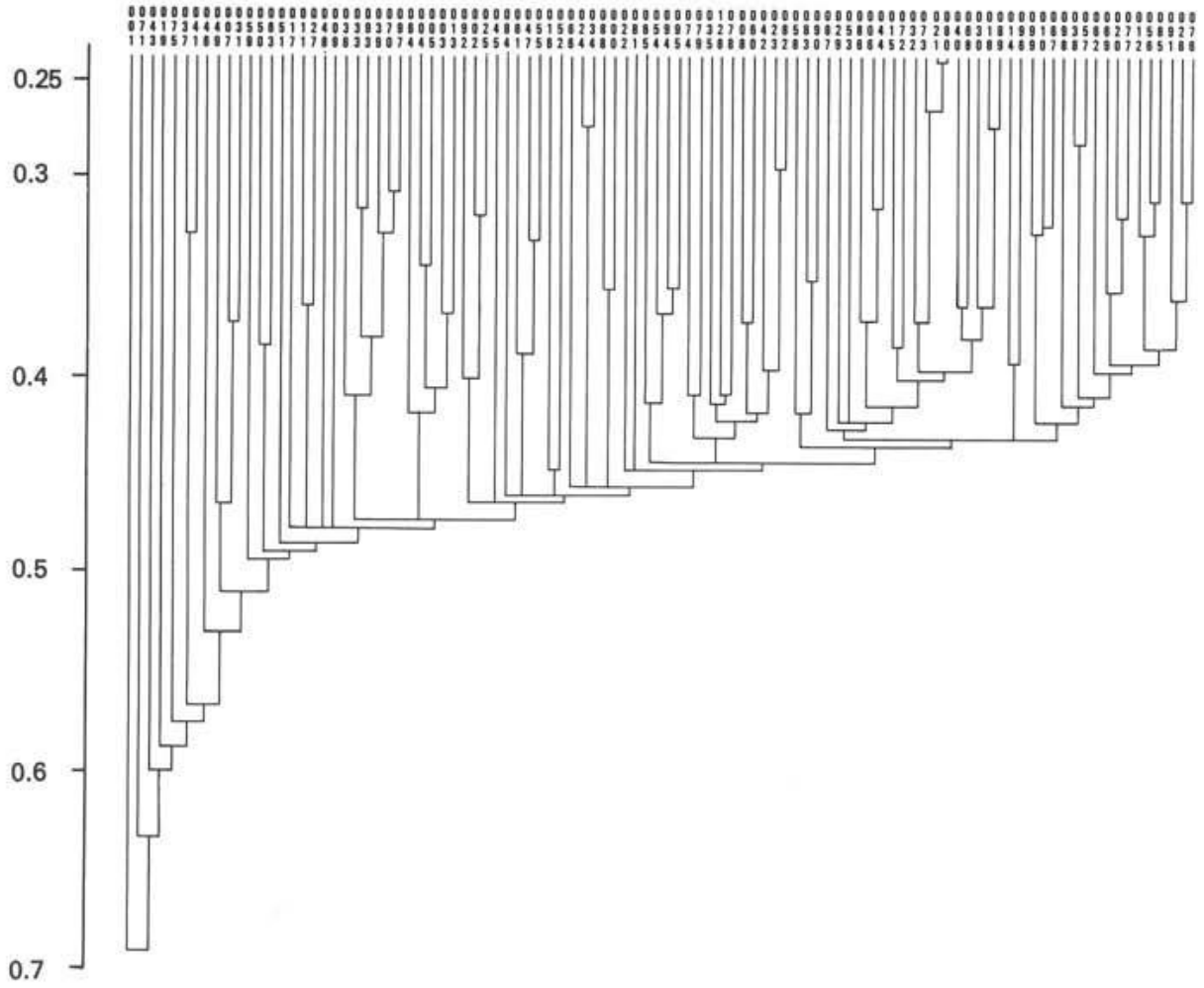


Figure 3.10 Single-link hierarchy for 100 random patterns in six dimensions.

3.2.6 Ties in Proximity

The computational complexity of competing algorithms for implementing a particular clustering method and the availability of software should determine which algorithm is appropriate for a given application. The problem of choosing between the single-link and complete-link methods is much more difficult than choosing an algorithm for one of the methods. No list of characteristics exists that lets us choose between the two methods in a calm, rational manner. Some theoretical and practical information about the two methods is summarized in this section, especially the effects of ties in the proximity matrix.

The single-link and complete-link methods differ in many respects, such as in the graph structures recognized and the updating procedure. The two methods produce the same clusterings when the proximity matrix satisfies the ultrametric inequality, as discussed in Section 3.2.3. This section demonstrates that the two methods differ in the way they treat ties in the proximity matrix. Up to now, we have assumed that the proximity matrix contains no ties so that two new clusters are never formed at the same level and the algorithms defined thus far produce unique dendrograms. A tie implies that two or more edges are added to the proximity

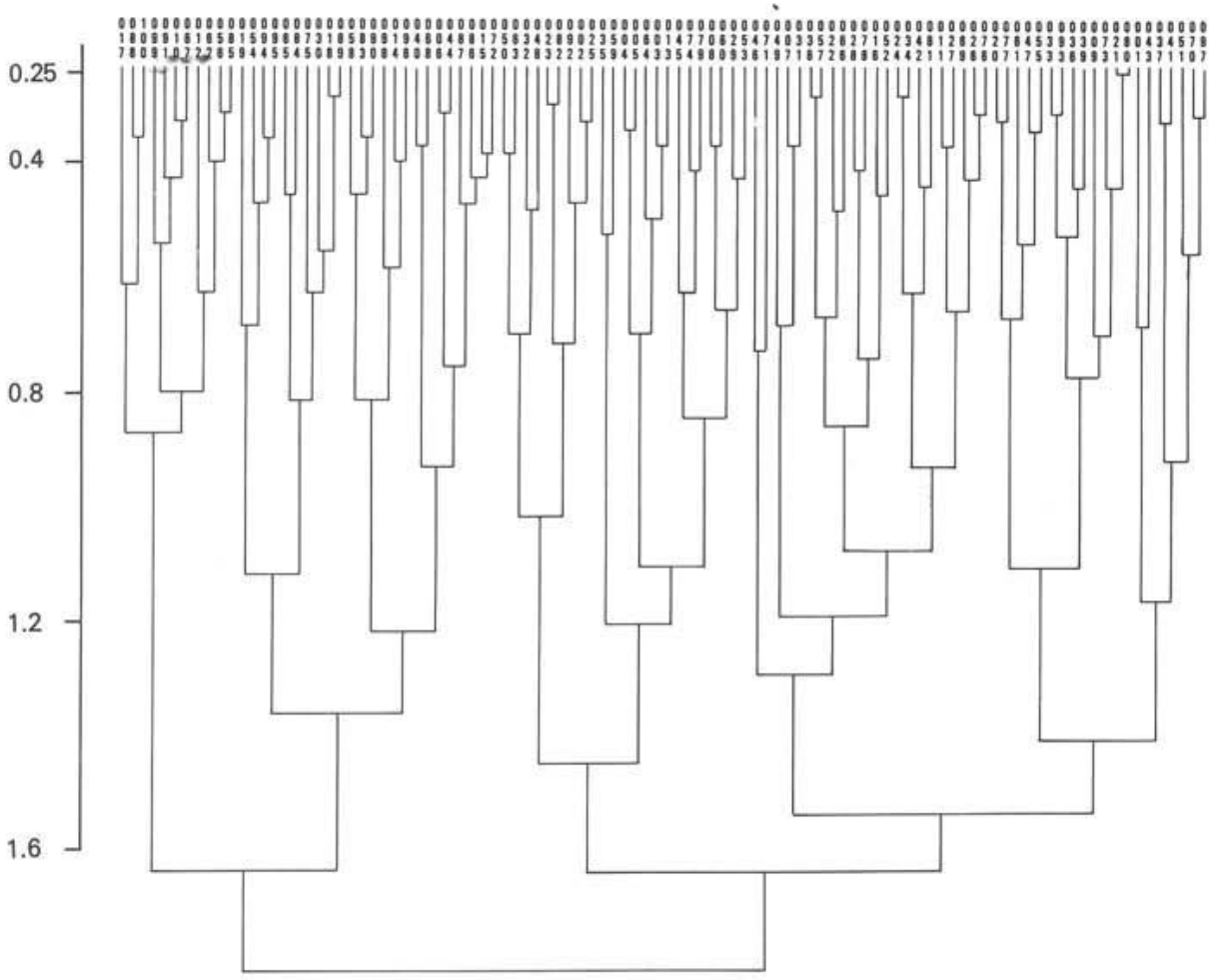


Figure 3.11 Complete-link hierarchy for 100 random patterns in six dimensions.

graph at once and that the minimum and maximum functions required in matrix updating are not unique.

Jardine and Sibson (1971) showed that the single-link method does not suffer from ambiguities due to ties because it has a continuity property. If the ties are broken in the proximities by adding or subtracting a small amount from the tied proximities, the resulting single-link dendrograms will merge smoothly into the same dendrogram as the added amount tends to zero, no matter how the ties are broken. This statement applies to all single-link algorithms as long as the rank orders of the proximities are not changed by the added amounts. By contrast, several complete-link dendrograms can be obtained by breaking ties in this way, as demonstrated in Figure 3.12.

Figure 3.12(a) shows the first three threshold graphs for the given proximity matrix. Two edges are added at once in $G(3)$. The single-link hierarchy is the same whether edge (2, 3) is inserted first or edge (3, 4) is inserted first. The proximity dendrogram for the single-link method in Figure 3.12(b) is unique even though more than one cluster can be formed at the same level. Algorithms based on the MST or on matrix updating produce the same results. The situation is very different with complete link clustering. Figure 3.12(b) shows the hierarchy defined by adding edge (2, 3) first and that formed when adding edge (3, 4)

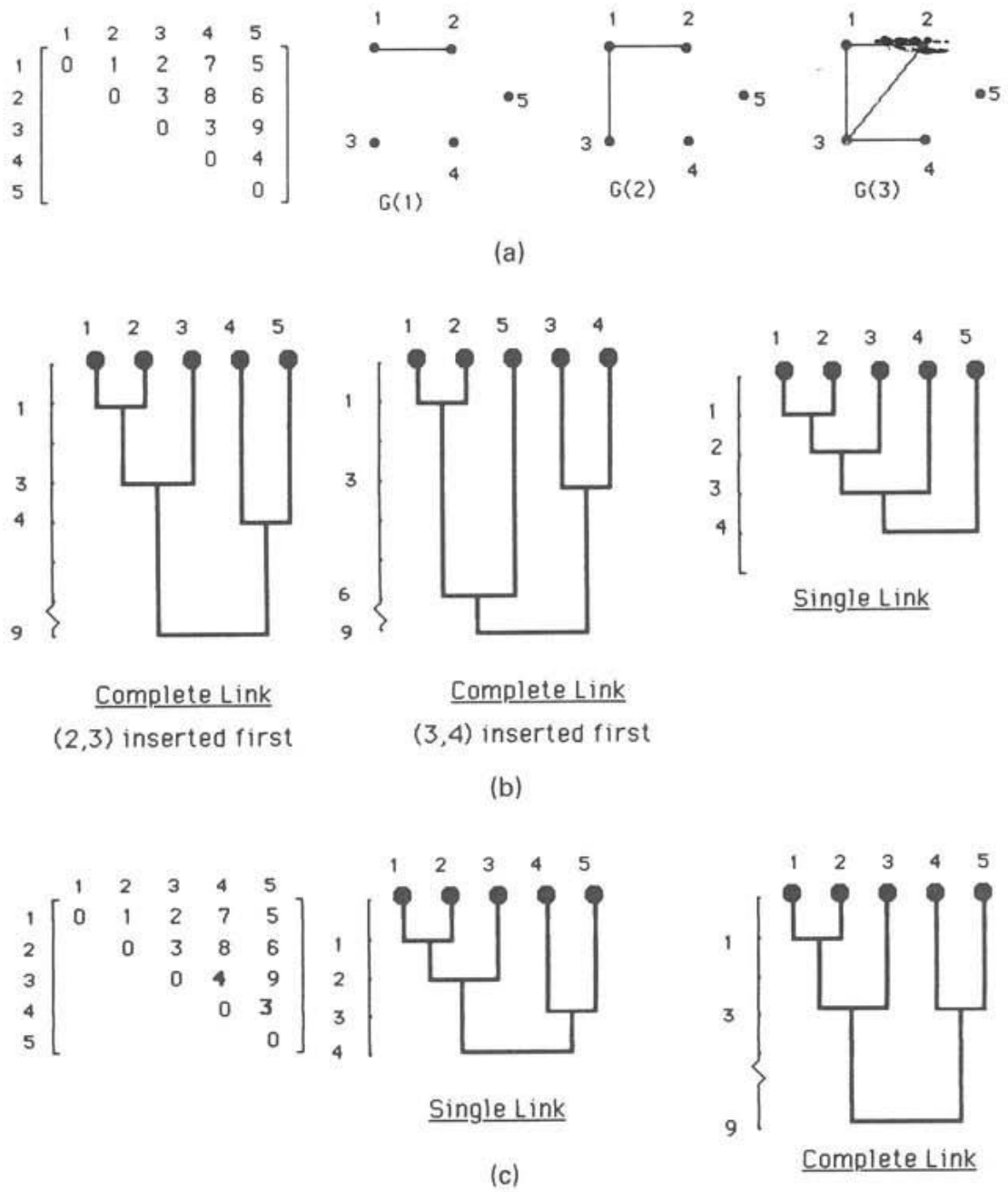


Figure 3.12 Effects of ties in proximity on single-link and complete-link clustering: (a) proximity matrix and threshold graphs; (b) proximity dendrograms; (c) altered proximity matrix and dendrograms.

first. The two clustering structures are very different. This effect can also be observed on the matrix updating algorithm and with Hubert's algorithm (Section 3.2.1). Adding the two edges (2, 3) and (3, 4) simultaneously does not solve the problem because the resulting four-edge threshold graph is not a complete graph. In fact, the next complete graph would be the one on all five nodes and the hierarchical clustering would have only three levels.

Figure 3.12(c) emphasizes the seriousness of ties. The given proximity matrix differs from that in Figure 3.12(a) in only two entries; the (3, 4) and (4, 5) entries are interchanged, as might occur through a typing error when entering

data. In this case a unique complete-link hierarchy is obtained because the two edges with the same proximity can be added in arbitrary order. However, the hierarchy has two clusters forming at level 3. The single-link hierarchy is also shown. The single- and complete-link dendrograms in Figure 3.12(c) resemble one another much more closely than do those in Figure 3.12(b), which might lead one to believe that the proximity matrix in Figure 3.12(c) had a good hierarchical structure, whereas that in Figure 3.12(a) has a poor hierarchical structure. This example raises the issue of sensitivity. It appears that the hierarchical structure can change dramatically with small changes in the rank orders of the proximities.

The havoc that ties can create in a complete-link hierarchy has been noted by several researchers. Sibson (1971) and Williams et al. (1971) argue against the complete-link method as a feasible clustering procedure (see also Hubert, 1974a). The practical problem of ties is subtle. Software packages do not typically check for ties. The order in which an edge is added from a set of edges with the same proximity is at the whim of the programmer. The program will generate only one complete-link clustering, even though a number of clusterings might be equally justifiable. This problem is compounded when the proximity matrix contains several ties. The comparative studies in Section 3.5.2 suggest that the complete-link method produces more useful hierarchies in many applications than does the single-link method, even though proximity ties make it ambiguous.

3.2.7 General Matrix Updating Algorithms and Monotonicity

This section generalizes the algorithms in Section 3.2.5 and discusses issues in the computation and application of these algorithms. Questions of the validity of cluster structures are taken up in Chapter 4. The general paradigm for expressing SAHN (Sequential, Agglomerative, Hierarchical, Nonoverlapping) clustering methods is given in Section 3.2.5. Step 4 of that algorithm specifies how the dissimilarity matrix is to be updated by defining the formula for the dissimilarity between a newly formed cluster, (r, s), and an existing cluster, (k) with n_k objects. The single-link and complete-link algorithms use the minimum and maximum, respectively, of the dissimilarities between the pairs $\{(k), (r)\}$ and $\{(k), (s)\}$. Other clustering methods can be defined by specifying different combinations of the distances involved. A general formula for step 4 that includes most of the commonly referenced hierarchical clustering methods is given below.

$$d[(k), (r, s)] = \alpha_r d[(k), (r)] + \alpha_s d[(k), (s)] + \beta d[(r), (s)] + \gamma |d[(k), (r)] - d[(k), (s)]|$$

This formula was first proposed by Lance and Williams (1967). Table 3.1 shows the parameter values for the most common algorithms. This table is also given in Milligan (1979) and in Day and Edelsbrunner (1984).

The acronym "PGM" refers to the "pair group method"; the prefixes "U" and "W" refer to unweighted and weighted, respectively. An "unweighted" method

TABLE 3.1 Coefficient Values for SAHN Matrix Updating Algorithms

Clustering Method	α_r	α_s	β	γ
Single-link	1/2	1/2	0	$-1/2$
Complete-link	1/2	1/2	0	1/2
UPGMA (group average)	$\frac{n_r}{n_r + n_s}$	$\frac{n_s}{n_r + n_s}$	0	0
WPGMA (weighted average)	1/2	1/2	0	0
UPGMC (unweighted centroid)	$\frac{n_r}{n_r + n_s}$	$\frac{n_s}{n_r + n_s}$	$\frac{-n_r n_s}{(n_r + n_s)^2}$	0
WPGMC (weighted centroid)	1/2	1/2	$-1/4$	0
Ward's method (minimum variance)	$\frac{n_r + n_k}{n_r + n_s + n_k}$	$\frac{n_s + n_k}{n_r + n_s + n_k}$	$\frac{-n_k}{n_r + n_s + n_k}$	0

treats each object in a cluster equally, regardles of the structure of the dendrogram. A "weighted" method weights all clusters the same, so objects in small clusters are weighted more heavily than objects in large clusters. The suffixes "A" and "C" refer to "arithmetic averages" and "centroids." Thus "UPGMA" stands for "unweighted pair group method using arithmetic averages" and "WPGMC" refers to "weighted pair group method using centroids." Rohlf (1970) and Sneath and Sokal (1973) have used this terminology. The UPGMC method has also been called, simply, the centroid method, while the WPGMC method has been called the median method (see Lance and Williams, 1967).

Sneath and Sokal (1973) provide a good discussion of the backgrounds of these methods and define other SAHN algorithms. Arithmetic averaging attempts to avoid the extremes of the single-link and complete-link methods. When measuring the dissimilarity between an existing cluster and a prospective cluster, the single-link method finds the closest pair of objects in the two clusters, the complete-link method finds the most distant pair, and the UPGMA and WPGMA methods use arithmetic averages of the dissimilarities. The arithmetic averaging methods have no simple geometric interpretation. In contrast, the UPGMC and WPGMC methods have direct geometric interpretations when the objects are represented as patterns in a d-dimensional space. The centroid methods assess the dissimilarity between two clusters by the distance between centroids. The UPGMC method measures distance in terms of the centroid computed from all patterns in each cluster. The WPGMC method computes centroids from the centroids of the two clusters that merge to form a new cluster. The UPGMA weights the contribution of each pattern equally by taking into account the sizes of the clusters, while the WPGMC weights the patterns in small clusters more heavily than the patterns in large clusters. Centroid methods should only be used when the objects are represented as patterns and the proximity measure is squared Euclidean distance. An important distinction between centroid methods and other SAHN algorithms is in monotonicity, as explained later in this section.

Example 3.6

Dendrograms for the seven algorithms in Table 3.1 are drawn in Figure 3.13. The six objects involved are the six pattern vectors defined below in a three-dimensional space.

$$\mathbf{x}_1 = (1.0 \quad 2.0 \quad 2.0)^T \qquad \mathbf{x}_4 = (3.0 \quad 4.0 \quad 3.0)^T$$

$$\mathbf{x}_2 = (2.0 \quad 1.0 \quad 2.0)^T \qquad \mathbf{x}_5 = (0.0 \quad 3.5 \quad 3.5)^T$$

$$\mathbf{x}_3 = (0.0 \quad 1.0 \quad 3.0)^T \qquad \mathbf{x}_6 = (2.0 \quad 2.5 \quad 2.5)^T$$

Under a squared Euclidean distance measure of dissimilarity, the proximity matrix is given below.

	1	2	3	4	5	6
1	0	2.0	3.0	9.0	5.5	1.5
2		0	5.0	11.0	12.5	2.5
3			0	18.0	6.5	6.5
4				0	9.5	3.5
5					0	6.0
6						0

The tie in proximity between pairs of patterns (x_3, x_5) and (x_3, x_6) causes no ambiguity in any of the dendrograms. The dendrograms for Ward's method, the two arithmetic average methods, and the two centroid methods all have the same topology and differ only in levels. They suggest that x_4 is an outlier because it joins the cluster of the other five patterns last and the gap between the formation of the five-pattern cluster and the singleton cluster is large. The single-link dendrogram has much the same topology, except that x_5 now appears to be the outlier. The complete-link dendrogram establishes cluster (x_4, x_5). All dendrograms agree that (x_1, x_2, x_6) is a strong cluster. Quantitative measures of the strength and quality of clusters and clusterings, defined in Chapter 4, should help answer such questions as: If one of the dendrograms were to be cut to define a partition, where is the best cutting level? Is (x_1, x_2, x_6, x_3) a good cluster?

Several of the comparative studies discussed in Section 3.5.2 conclude that Ward's method (Ward, 1963), also called the minimum variance method, outperforms other hierarchical clustering methods. This method is based on notions of square error popularized in analysis-of-variance and other statistical procedures (Wilks, 1963; Cooley and Lohnes, 1971). Square-error criteria are also used in partitional clustering algorithms (Section 3.3.1). Ward's method is implemented by the standard algorithm using the constants in Table 3.1. These constants are derived below to see how square error is minimized.

Suppose that a clustering has been achieved with Ward's method and that the next clustering in the hierarchy is to be obtained with the matrix updating algorithm. Ward's method is designed for the situation when the data appear as patterns. Thus we begin with a set of n patterns in a d-dimensional space. Let $x_{ij}^{(k)}$ be the value for feature j of pattern i when pattern i is in cluster k for i from 1 to n_k and j from 1 to d. The centroid of cluster k, denoted $[m_1^{(k)}, \ldots, m_d^{(k)}]$, is the cluster center, or the average of the n_k patterns in cluster k.

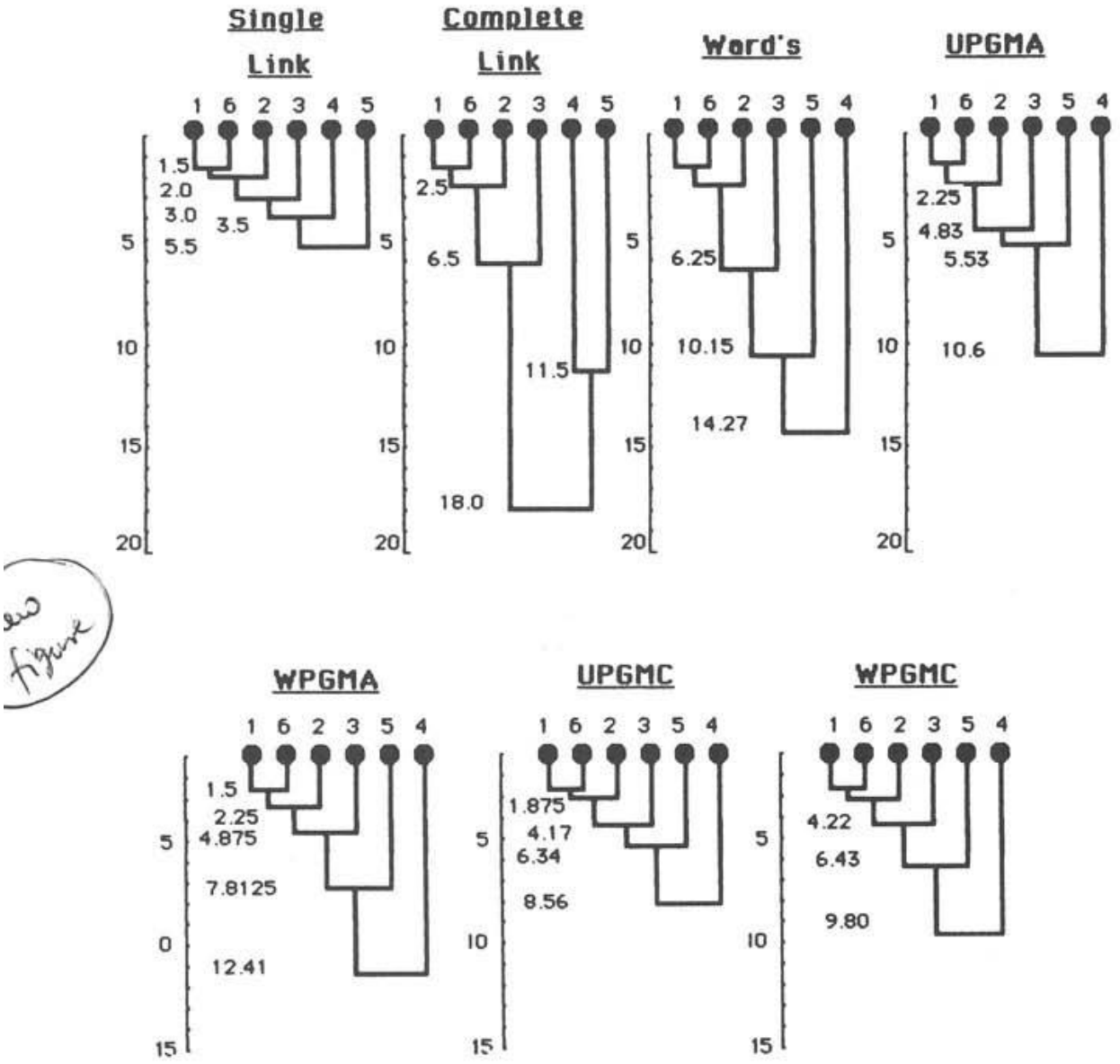


Figure 3.13 Examples of dendrograms for matrix updating algorithms.

$$m_j^{(k)} = (1/n_k) \sum_{i=1}^{n_k} x_{ij}^{(k)}$$

The square-error for cluster k is the sum of squared distances to the centroid for all patterns in cluster k.

$$e_k^2 = \sum_{i=1}^{n_k} \sum_{j=1}^{d} [x_{ij}^{(k)} - m_j^{(k)}]^2$$

The square-error for the entire clustering, which contains K clusters, is the sum of the square-errors for the individual clusters.

$$E_K^2 = \sum_{k=1}^{K} e_k^2$$

Ward's method merges the pair of clusters that minimizes ΔE_{pq}^2, the change in E_K^2 caused by the merger of clusters p and q into cluster t to form the next clustering.

Since the square-errors for all clusters except for the three clusters involved remain the same,

$$\Delta E_{pq}^2 = e_t^2 - e_p^2 - e_q^2$$

After a bit of algebra, we find that the change in square-error depends only on the centroids.

$$\Delta E_{pq}^2 = \frac{n_p n_q}{n_p + n_q} \sum_{j=1}^{d} [m_j^{(p)} - m_j^{(q)}]^2$$

The clusters p and q selected for merger are the clusters that minimize this quantity. The square-error must increase as the number of clusters decreases, but the increase is as small as possible in Ward's method. Once clusters p and q are merged into cluster t, the proximity between all other clusters and the new cluster t must be updated. Letting cluster r represent a cluster other than p, q, or t, the following formula can be applied to find $d[(r), (t)]$:

$$d[(r), (t)] = \frac{n_r + n_p}{n_r + n_t} d[(r), (p)] + \frac{n_r + n_q}{n_r + n_t} d[(r), (q)] - \frac{n_r}{n_r + n_t} d[(p), (q)]$$

The choice of a suitable hierarchical clustering method is an important matter in applications, but theory provides few guidelines for optimizing the choice. Square-error is a familiar criterion in engineering, so one might feel comfortable with a procedure that minimizes square-error, such as Ward's method. However, the objective of cluster analysis is to investigate the structure of the data, so the imposition of an apriori criterion, such as square-error, might not be appropriate. All data do not occur as patterns, so we cannot limit our thinking to geometrical constructs. Section 3.5.2 reviews several empirical studies that compare hierarchical clustering methods and that guide the choice of a clustering method.

3.2.8 Crossovers and Monotonicity in Dendrograms

Section 3.2.2 defined perfect hierarchical structure as a proximity matrix that satisfies the ultrametric inequality. The rationale was that the single-link and complete-link methods produced the same dendrograms for an ultrametric proximity matrix, and since these two methods search for very different types of structure, the fact that they exhibit the same exact structure is meaningful. It is clear from the single-link and complete-link algorithms based on threshold and proximity graphs that these methods are *monotonic*. That is, the level at which the next cluster forms is always larger, on a dissimilarity scale, than the level of the current clustering. Monotone methods induce ultrametric cophenetic matrices. Monotonicity can be expressed in mathematical terms by referring to the matrix updating formula in Section 3.2.7. If a clustering method merges clusters (r) and (s) into cluster (r, s), monotonicity demands that

$$d[(k), (r, s)] \geq d[(r), (s)]$$

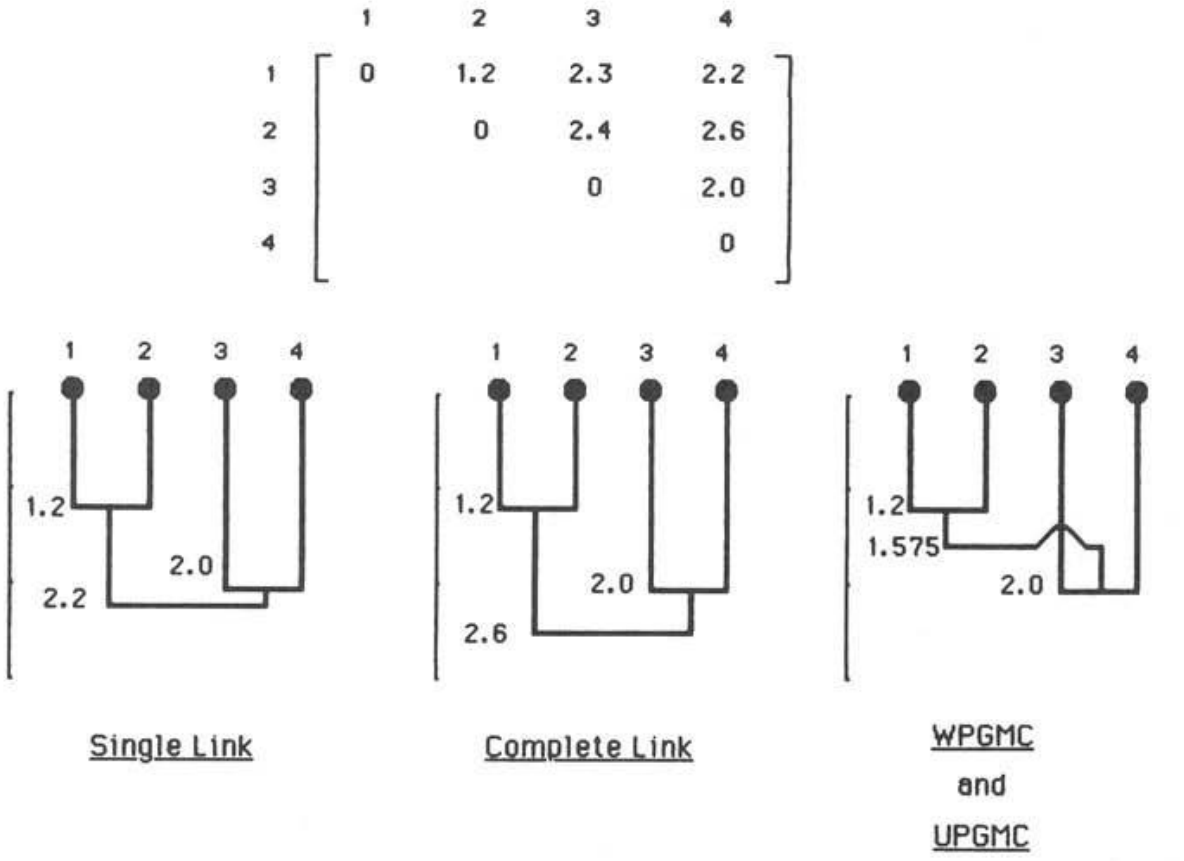


Figure 3.14 Examples of crossover or reversal in dendrogram.

for all clusters (k) distinct from (r) and (s). That is, no dissimilarity in the updated matrix can be smaller than the smallest entry in the previous matrix. Another way of saying this is that the cophenetic matrix generated by these two methods satisfies the ultrametric inequality.

What can be said about the monotonicity of SAHN algorithms expressed through matrix updating, especially those defined by the matrix updating algorithm in Table 3.1? Figure 3.14 provides a simple example of a dissimilarity matrix and the dendrograms generated by the single-link, complete-link, UPGMC, and WPGMC methods. The dendrograms from the centroid methods (UPGMC and WPGMC) are not monotone and exhibit what is called a "crossover" or a "reversal" since clusters (x_1, x_2) and (x_3, x_4) merge at a level lower than the level at which (x_3, x_4) is first defined.

Monotonicity is clearly a property of the clustering method and has nothing to do with the proximity matrix. The advantage of the matrix updating formula is that the monotonicity of any SAHN algorithm that can be expressed in terms of this updating formula can be predicted from the coefficients. Assuming that $\alpha_r > 0$, and $\alpha_s > 0$, Milligan (1979) provided the following results. The matrix updating formula for step 4 of the SAHN algorithm is repeated below for easy reference. Clusters (r) and (s) are being merged into cluster (r, s) and the dissimilarity between distinct cluster (k) and the newly formed cluster is being established.

$$d[(k), (r, s)] = \alpha_r d[(k), (r)] + \alpha_s d[(k), (s)] + \beta d[(r), (s)] + \gamma |d[(k), (r)] - d[(k), (s)]|$$

Result 1. If $\alpha_r + \alpha_s + \beta \geq 1$ and $\gamma \geq 0$, the clustering method is monotone.

This result is easily demonstrated. The first inequality can be rewritten as

$$\beta \geq 1 - \alpha_r - \alpha_s$$

and substituting in the matrix updating formula shows that

$$d[(k), (r, s)] \geq d[(r), (s)] + \alpha_r\{d[(k), (r)] - d[(r), (s)]\} + \alpha_s\{d[(k), (s)] - d[(r), (s)]\} + \gamma |d[(k), (r)] - d[(k), (s)]|$$

SAHN algorithms require that $d[(r), (s)]$ be no greater than either $d[(k), (r)]$ or $d[(k), (s)]$ for any distinct (k). Thus the condition that γ be nonnegative implies that

$$d[(k), (r, s)] \geq d[(r), (s)]$$

for all clusters (k) other than (r) and (s), which implies monotonicity.

Result 2. If $\alpha_r + \alpha_s + \beta \geq 1$ and $0 > \gamma \geq \max\{-\alpha_r, -\alpha_s\}$, the clustering method is monotone.

To demonstrate this result, first consider the case when $d[(k), (r)] > d[(k), (s)]$. Using the first inequality and recalling that γ is negative, the matrix updating equation can be rewritten as

$$d[(k), (r, s)] \geq d[(r), (s)] + (\alpha_r - |\gamma|)\{d[(k), (r)] - d[(r), (s)]\} + (\alpha_s + |\gamma|)\{d[(k), (s)] - d[(r), (s)]\}$$

Since $\alpha_r \geq |\gamma|$, the second term on the right is nonnegative for the case under consideration. The last term on the right is nonnegative because of the way SAHN algorithms are defined. Thus

$$d[(k), (r, s)] \geq d[(r), (s)]$$

as in Result 1, and the clustering method is monotone. The case when $d[(k), (s)] > d[(k), (r)]$ can be proved in a similar fashion.

The inequality in Result 1 is not satisfied for either of the centroid methods (UPGMC and WPGMC). It is easy to create examples for which these methods do not produce monotone hierarchies, as indicated in Figure 3.14. However, all other methods in Table 3.1 are monotone. Note that only single-link and complete-link clusterings are invariant under monotone transformation of the dissimilarities.

Figure 3.13 demonstrates that nonmonotone clustering methods do not necessarily produce crossovers. ~~The discussion of the centroid methods suggests that they should only be applied when objects are patterns in a pattern space so crossovers~~ Crossovers can occur even when the objects are patterns in a pattern space and Euclidean distance is the measure of proximity.

will be eliminated. One is tempted to reject nonmonotone methods out of hand. Williams and Lance (1977) call them obsolete. Anderberg (1973) claims that they lack a useful interpretation for general proximities. Sneath and Sokal (1973) claim that "the frequency of reversals and the relatively high degree of its distortion of the original [proximity] matrix has led to the abandonment of this [UPGMC] technique." On the other hand, the performances of nonmonotone methods in several comparative studies of clustering methods discussed in Section 3.5.2 do not suggest that such methods be abandoned. Williams et al. (1971) argue that monotonicity is not essential for the proper performance of hierarchical clustering.

3.2.9 Clustering Methods Based on Graph Theory

The statements of the single-link and complete-link algorithms in terms of graph theory in Section 3.2.1 suggest that properties other than connectedness and completeness can be used to define clustering methods. The idea is to watch the sequence of threshold graphs or proximity graphs for the appearance of a suitable property. Hubert (1974a) suggests the following expression of algorithms that define hierarchical clustering methods. Ties in the proximities can affect the clusterings in unexpected ways, so we assume that no ties exist in the proximity matrix.

New hierarchical clustering algorithms are formed by changing step 2 in the algorithm of Section 3.2.1. The function $Q_{\rho(k)}$ is defined as follows for all pairs of clusters $\{C_{mr}, C_{mt}\}$ in the clustering $\{C_{m1}, \ldots, C_{m(n-m)}\}$:

$$Q_{\rho(k)}(r, t) = \min \{d(i, j) : \text{the maximal subgraph of } G[d(i, j)] \text{ defined by } C_{mr} \cup C_{mt} \text{ is connected and either has property } \rho(k) \text{ or is complete}\}$$

Following the algorithm, clusters C_{mp} and C_{mq} are merged to form the next clustering in the sequence if

$$Q_{\rho(k)}(p, q) = \min \{Q_{\rho(k)}(r, t)\}$$

Some examples of property ρ are given below. Integer k is a parameter, so, for example, $\rho(k)$ could mean a node connectivity of k or a node degree of k.

Node connectivity. The node connectivity of a connected subgraph is the largest number n_c such that all pairs of nodes are joined by at least n_c paths having no nodes in common.

Edge connectivity. The edge connectivity of a connected subgraph is the largest integer n_e such that all pairs of nodes are joined by at least n_e paths having no edges in common.

Node degree. The degree of a connected subgraph is the largest integer n_d such that each node has at least n_d incident edges.

Diameter. The diameter of a connected subgraph is the maximum "distance" between two nodes in the subgraph. The distance between two nodes is the number of edges in the shortest path joining them.

Radius. The radius of a connected subgraph is the smallest integer n_r such that at least one node is within distance n_r of all other nodes in the subgraph.

Specifying parameter k and property ρ defines a new clustering method. Every cluster must at least be connected. Once all the edges have been inserted into the subgraph, it is complete and no further properties can be applied. Certain practical difficulties arise when trying to select a suitable property. Few guidelines exist other than intuition and experience. Theorems from mathematics provide some insight into these methods. For example, a node connectivity of k implies an edge connectivity of k, but the reverse is not true. Similarly, an edge connectivity of k implies a minimum degree of k, but the reverse does not hold. A compelling reason must appear before one of these methods is used in place of the single-link, complete-link, or other SAHN algorithms.

Example 3.7

Figure 3.15 demonstrates hierarchical clustering methods defined by graph properties. An ordinal proximity matrix is given below on eight objects. Threshold graph $G(13)$ is pictured in Figure 3.15(a) to help in establishing the dendrograms for several methods, as well as for the single-link and complete-link methods. Proximity dendrograms are shown in Figure 3.15(b)–(h). A simple way to find these hierarchies with pencil and paper is first to list the pairs of objects in rank order by proximity. Then construct a sequence of threshold graphs and find the first threshold graph at which a property is satisfied. It is important to check the property only on the subgraph formed by the union of the subgraphs for two *existing* clusters.

	1	2	3	4	5	6	7	8
1	0	13	21	18	4	8	7	28
2	—	0	9	19	15	14	10	16
3	—	—	0	22	20	12	11	17
4	—	—	—	0	3	23	27	1
5	—	—	—	—	0	5	24	2
6	—	—	—	—	—	0	6	25
7	—	—	—	—	—	—	0	26
8	—	—	—	—	—	—	—	0

Ling (1972) examined hierarchical clustering based on notions of connectivity and compactness that are particularly appropriate for ordinal proximity matrices. He assumed ordinal proximities with no ties, but his clustering method can also be applied to interval and ratio proximity matrices. The properties ρ that Ling proposed are defined below.

Consider a proximity graph on n nodes. A subgraph is *r-connected* if all pairs of nodes in the subgraph are connected by r-chains. An *r-chain* between two nodes is a sequence of nodes having $d(i, j) \leq r$ for all pairs (i, j) of nodes in the sequence. A subgraph is (k, r)-*bonded* if every node in the subgraph is directly connected to at least k nodes and if $d(i, j) \leq r$ for all k connections. Finally, a subgraph is (k, r)-*connected* if it is both r-connected and (k, r)-bonded.

A subgraph becomes a (k, r)-*cluster* in the algorithm of Section 3.2.1 as soon as it becomes (k, r)-connected. Note that $(1, r)$ clusters are single-link clusters. Ling (1972, 1973a) proposed the (k, r)-cluster as a way of identifying significant clusters. Given any proximity graph, a (k, r)-cluster can be defined independent of the hierarchical clustering algorithm. A subgraph is a (k, r)-cluster if r is the smallest value of s for which the subgraph is (k,s)-connected for some s and the subgraph is not properly contained in any other (k, t)-connected subgraph for $t > r$. Such clusters have several attractive mathematical properties and are intuitively appealing since both connectedness and compactness are involved in the definition.

We emphasize that no theory exists for choosing among the various properties of graphs to select the "best" clustering method for a particular application. Section 3.5 provides some guidance, but familiarity with a method and confidence in the results of previous applications of the method are the only practical ways of choosing a method.

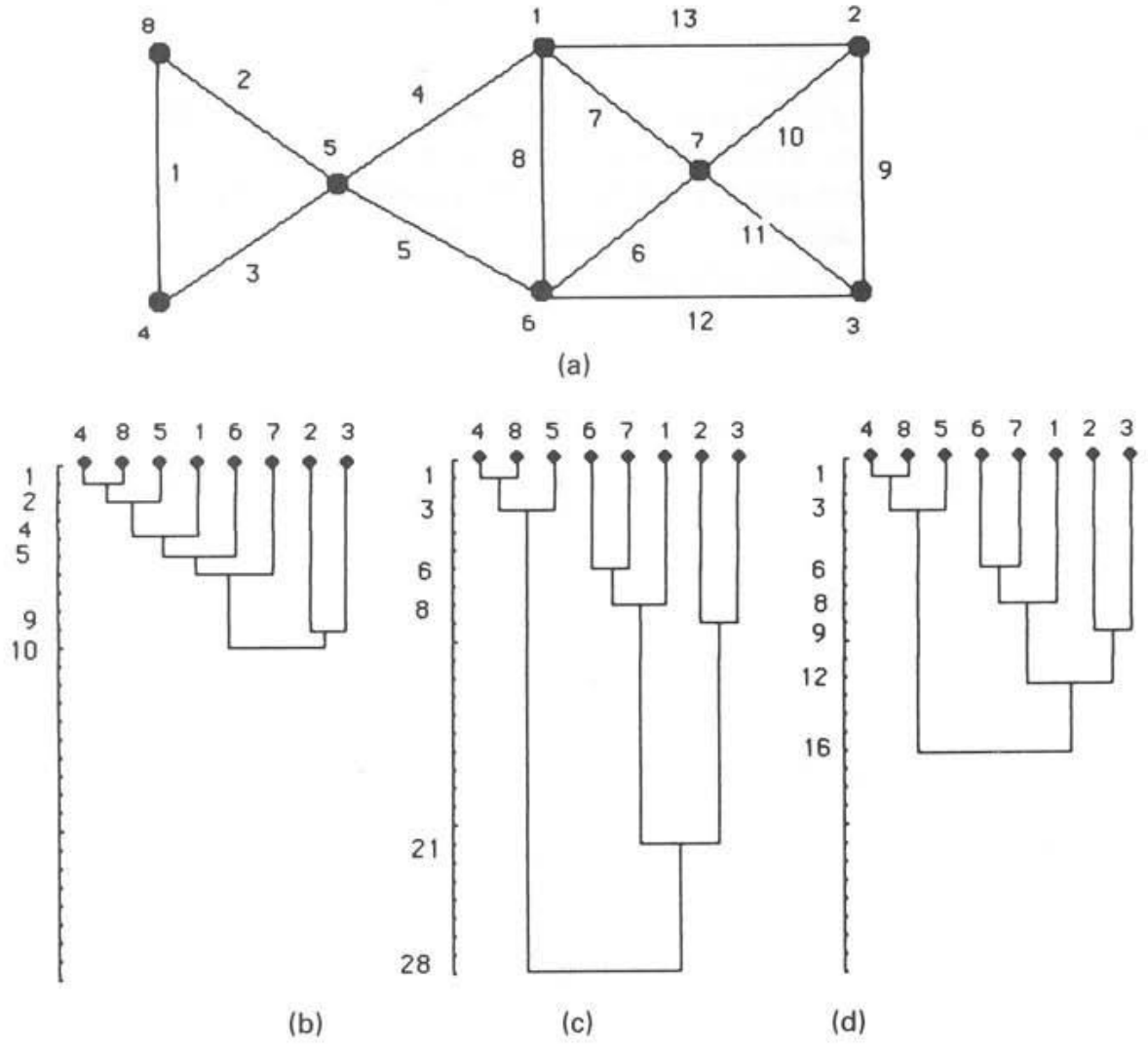


Figure 3.15 Examples of dendrograms from graph theory: (a) threshold graph G(13) for proximity matrix in example 3.7; (b) single-link; (c) complete-link; (d) 2-node connected; (e) 2-edge connected; (f) 2-degree; (g) 2-diameter; (h) 2-radius.

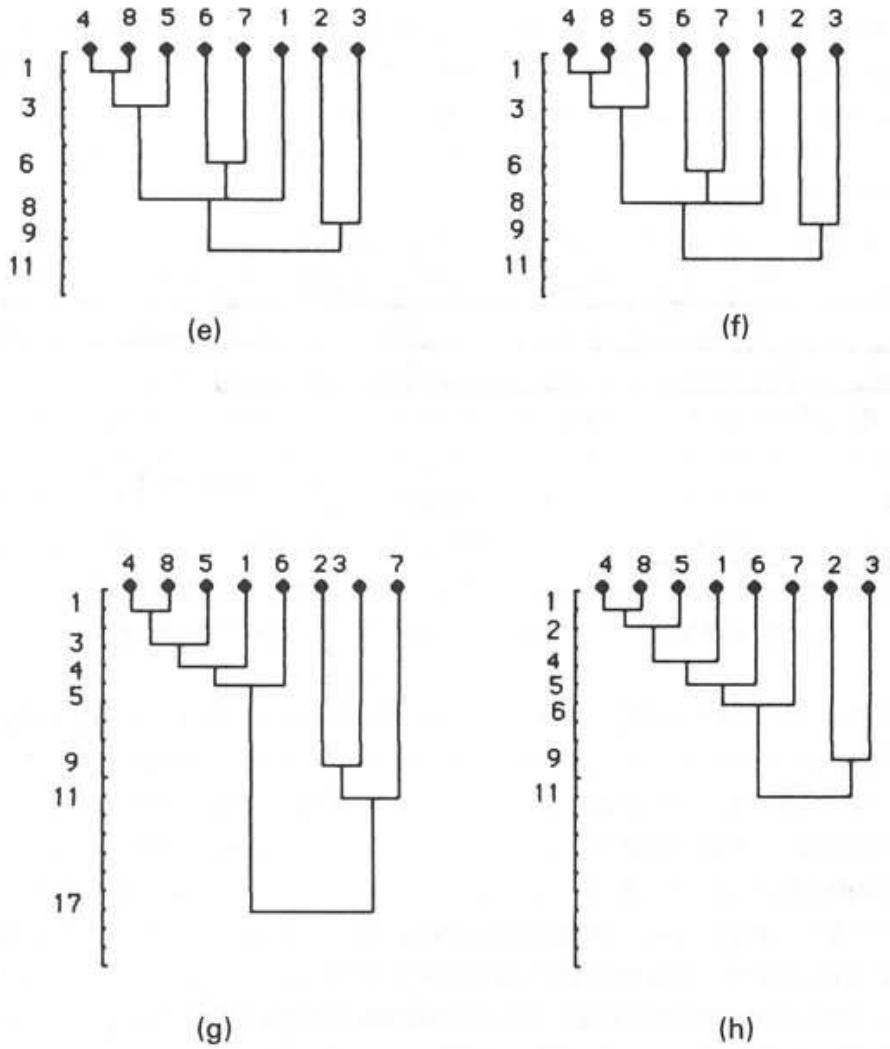


Figure 3.15 (*continued*)

3.3 PARTITIONAL CLUSTERING

Hierarchical clustering techniques organize the data into a nested sequence of groups. An important characteristic of hierarchical clustering methods is the visual impact of the dendrogram, which enables a data analyst to see how objects are being merged into clusters or split at successive levels of proximity. The data analyst can then try to decide whether the entire dendrogram describes the data or can select a clustering, at some fixed level of proximity, which makes sense for the application in hand. We refer to nonhierarchical clustering methods as *partitional* clustering methods. They generate a single partition of the data in an attempt to recover natural groups present in the data. Both clustering strategies have their appropriate domains of applications. Hierarchical clustering methods generally require only the proximity matrix among the objects, whereas partitional techniques expect the data in the form of a pattern matrix. It is generally assumed that the features have been measured on a ratio scale.

Hierarchical techniques are popular in biological, social, and behavioral sci-

ences because of the need to construct taxonomies. Partitional techniques are used frequently in engineering applications where single partitions are important. Partitional clustering methods are especially appropriate for the efficient representation and compression of large data bases. Dendrograms are impractical with more than a few hundred patterns.

The problem of partitional clustering can be formally stated as follows. Given n patterns in a d-dimensional metric space, determine a partition of the patterns into K groups, or clusters, such that the patterns in a cluster are more similar to each other than to patterns in different clusters. The value of K may or may not be specified. A clustering criterion, such as square-error, must be adopted. Criteria can be classified as global or local. A *global* criterion represents each cluster by a prototype and assigns the patterns to clusters according to most similar prototypes. A *local* criterion forms clusters by utilizing local structure in the data. For example, clusters can be formed by identifying high-density regions in the pattern space or by assigning a pattern and its k nearest neighbors to the same cluster.

The theoretical solution to this partitional problem is straightforward. Simply select a criterion, evaluate it for all possible partitions containing K clusters, and pick the partition that optimizes the criterion. The first difficulty encountered is selecting a criterion that translates one's intuitive notions about "cluster" into a reasonable mathematical formula. Criteria are highly dependent on problem parameters and must be simple for computational reasons but complex enough to reflect various data structures. The second difficulty with this approach is that the number of partitions is astronomical, even for moderate numbers of patterns, so evaluating even the simplest criterion over all partitions is impractical.

Let $S(n, K)$ denote the number of clusterings of n objects into K clusters. The order of the objects in each cluster and the order of the clusters themselves are immaterial. Empty clusters are not counted. A partial difference equation can be written for $S(n, K)$ as follows. Suppose that all clusterings of $n - 1$ objects have been listed. A clustering of n objects can be formed from this list in two ways.

1. The nth object can be added as a singleton cluster to each member of the list with exactly $(K - 1)$ clusters.
2. The nth object can be added to each cluster of any member of the list with exactly K clusters.

Thus

$$S(n, K) = S(n - 1, K - 1) + KS(n - 1, K)$$

The boundary conditions on this equation are

$$S(n, 1) = 1, S(n, n) = 1, S(n, K) = 0 \qquad \text{if } K > n$$

The solution to this equation for $S(n, K)$ requires that values $\{S(j, p)\}$ be known for the set $\{(j, p) : 1 \leq j \leq n - 2, 1 \leq p \leq K\}$.

Solutions to the partial difference equation are called Stirling numbers of the second kind (Fortier and Solomon, 1966; Jensen, 1969):

$$S(n, K) = \frac{1}{K!} \sum_{i=1}^{K} (-1)^{K-i} \binom{K}{i} (i)^n$$

There are only 34,105 distinct partitions of 10 objects into four clusters, but this number explodes to approximately 11,259,666,000 if 19 objects are to be partitioned into four clusters. Clearly, exhaustive enumeration of all possible partitions is not computationally feasible even for small numbers of patterns. In addition, K, the number of clusters, must be selected a priori. Chapter 4 discusses this problem.

To avoid this combinatorial explosion, a criterion function is evaluated only for a small set of "reasonable" partitions. How to identify a small subset of partitions that has a good chance of containing the optimal partition? The most common approach is to optimize the criterion function using an iterative, hill-climbing technique. Starting with an initial partition, objects are moved from one cluster to another in an effort to improve the value of the criterion function. Thus each successive partition is a perturbation of the previous one and, therefore, only a small number of partitions is examined. Algorithms based on this technique are computationally efficient but often converge to local minima of the criterion function. Several heuristics for choosing the initial partition, moving objects from one cluster to the other, and for merging and splitting clusters will be discussed later.

Another way of avoiding the combinatorial explosion is somehow to identify and reject a large number of partitions which are not likely to be of interest. Jensen (1969) uses a dynamic programming approach to eliminate many partitions and is still able to achieve an optimal solution. A significant computational savings is realized, especially for large clustering problems, at the expense of algorithmic complexity. For example, to partition 19 objects into four clusters, less than 2% of the total number of partitions need to be evaluated using the dynamic programming formulation. Even this reduction is not enough to make this approach computationally feasible for practical clustering problems. Several related approaches are described in the literature (Edwards and Cavalli-Sforza, 1965; Vinod, 1969; Rao, 1971; Koontz et al., 1975; Lefkovitch, 1980).

There is no single "best" criterion for obtaining a partition because no precise and workable definition of "cluster" exists. Clusters can be of arbitrary shapes and sizes in a multidimensional pattern space. Each clustering criterion imposes a certain structure on the data, and if the data happen to conform to the requirements of a particular criterion, the true clusters are recovered. Only a small number of independent clustering criteria can be understood both mathematically and intuitively. Thus the hundreds of criterion functions proposed in the literature are related and the same criterion appears in several disguises. Shaffer et al. (1979) demonstrate the similarity of a mode-seeking partitional algorithm (Kittler, 1976) and the MST-based algorithm of Zahn (1971). Similarly, Urquhart (1982) shows that partitions obtained from a relative neighborhood graph are identical to those

generated by mutual near-neighbor clustering (Gowda and Krishna, 1978). The literature of cluster analysis is spread so widely and over so many areas of science that a single criterion function is rediscovered repeatedly.

In this section we present some of the most common partitional clustering methods. Several popular criteria are versions of square-error and are discussed in Sections 3.3.1 to 3.3.3. Another global criterion obtains a partition by fitting a mixture density model to the patterns (Section 3.3.4). These clustering criteria create clusters having hyperellipsoidal shapes. Sections 3.3.5 to 3.3.7 describe several partitional clustering methods based on local criteria of density or mode estimation, graph connectivity, and near-neighbor relationships. Finally, Section 3.3.8 briefly covers the topic of fuzzy clustering, where each object is permitted to belong to more than one cluster with a grade of membership.

Most of the partitional clustering techniques presented here implicitly assume continuous-valued feature vectors so that the patterns can be viewed as being embedded in a metric space. If the features are on a nominal or ordinal scale, Euclidean distances and cluster centers are not very meaningful, so hierarchical clustering methods are normally applied. Wong and Wang (1979) proposed a clustering algorithm for discrete-valued data. The approach is similar to the mode estimation procedure for continuous data but approximates the high-order discrete probability distribution by a second-order product and uses Hamming distance (Section 2.2) between patterns.

The technique of conceptual clustering or learning from examples (Michalski and Stepp, 1983) can be used with objects represented by nonnumeric or symbolic descriptors. The objective here is to group objects into conceptually simple classes. Clustering of trains using attributes such as number of cars, number of wheels, colors of wheels, and number of items carried, and the clustering of microcomputers using attributes such as CPU speed, memory size, and type of processor are more appropriately handled by associating each cluster with a simple "concept." Concepts are defined in terms of attributes. For example, in the train classification problem, trains with two red cars is a concept. Objects are arranged into a hierarchy of classes described by concepts.

3.3.1 Square-Error Clustering Criteria

The most commonly used partitional clustering strategy is based on the square-error criterion. The general objective is to obtain that partition which, for a fixed number of clusters, minimizes the square-error. Ward's method of hierarchical clustering (Section 3.2.7) uses square-error in a different way. Minimizing square-error, or within-cluster variation, will be shown to be equivalent to maximizing the between-cluster variation.

Suppose that the given set of n patterns in d dimensions has somehow been partitioned into K clusters $\{C_1, C_2, \ldots, C_K\}$ such that cluster C_k has n_k patterns and each pattern is in exactly one cluster, so that

$$\sum_{k=1}^{K} n_k = n$$

The mean vector, or center, of cluster C_k is defined as the centroid of the cluster, or

$$\mathbf{m}^{(k)} = (1/n_k) \sum_{i=1}^{n_k} \mathbf{x}_i^{(k)}$$

where $\mathbf{x}_i^{(k)}$ is the ith pattern belonging to cluster C_k. The square-error for cluster C_k is the sum of the squared Euclidean distances between each pattern in C_k and its cluster center $\mathbf{m}^{(k)}$. This square-error is also called the within-cluster variation.

$$e_k^2 = \sum_{i=1}^{n_k} (\mathbf{x}_i^{(k)} - \mathbf{m}^{(k)})^{\mathrm{T}} (\mathbf{x}_i^{(k)} - \mathbf{m}^{(k)})$$

The Mahalanobis distance (Section 2.2) can also be used to define square-error.

The square-error for the entire clustering containing K clusters is the sum of the within-cluster variations:

$$E_K^2 = \sum_{k=1}^{K} e_k^2$$

The objective of a square-error clustering method is to find a partition containing K clusters that minimizes E_K^2 for fixed K. The resulting partition has also been referred to as the minimum variance partition. Figure 3.16 illustrates that the square-error criterion views the centroids of clusters as prototypes. The error represents deviations of the patterns from the centroids. In other words, the patterns are viewed as a collection of K spherically shaped swarms. Square-error clustering tries to make the K swarms as compact and separated as possible.

Gordon and Henderson (1977) also define the clustering problem in terms of minimizing the within-cluster sum of square distances. However, they write their criterion function in such a way that the clustering problem can be formulated as a nonlinear programming problem. Let x_{ij} denote the jth feature of the ith

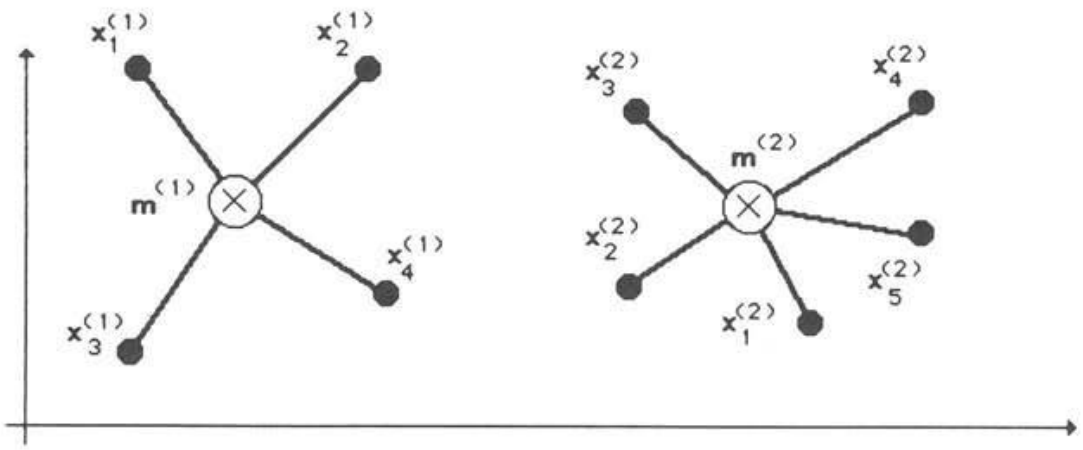


Figure 3.16 Distances used in computing square-error.

pattern, $i = 1, \ldots, n$; $j = 1, \ldots, d$. Let $y_{ik} = 1$ if the ith pattern belongs to the kth cluster, and 0 if the ith pattern does not belong to the kth cluster, $k = 1, \ldots, K$. The centroid of the kth cluster, $\mathbf{z}_k$, is written as $\mathbf{z}_k = (z_{k1}, \ldots, z_{kd})$, where

$$z_{kj} = \sum_{i=1}^{n} (y_{ik} x_{ij}) \Big/ \sum_{i=1}^{n} y_{ik}$$

The total within-cluster variation, denoted as S_T, can be written as

$$E_K^2 = S_T = \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \sum_{j=1}^{d} (x_{ij} - z_{kj})^2$$

Gordon and Henderson (1977) proposed the following two formulations for minimizing S_T.

1. Given the pattern matrix, $\mathcal{X}$, and the number of clusters, K, find the $n \times K$ matrix $\mathcal{Y} = [y_{ik}]$ that minimizes S_T. Note that $\mathcal{Y}$ is an $n \times K$ matrix of 0's and 1's with exactly one 1 in each row and at least one 1 in each column.
2. A more general formulation minimizes S_T under the assumption that $y_{ik} \in [0, 1]$ subject to the constraints

$$\sum_{k=1}^{K} y_{ik} = 1 \qquad \text{and} \qquad y_{ik} \geq 0$$

The term y_{ik} denotes the fraction of the ith pattern that is assigned to the kth cluster. This concept is similar to fuzzy clustering (Section 3.3.8), where a pattern belongs to a cluster with a "grade of membership." A lemma by Gordon and Henderson shows that the matrix $\mathcal{Y}$ which minimizes S_T under these constraints must contain only 0's and 1's. Thus the first formulation is a special case of the second formulation. The algorithm used to minimize S_T is based on the method of steepest descent.

A number of clustering criteria related to square-error have been derived from scatter matrices used in discriminant analysis. They are based on the following decomposition of the total scatter matrix, $\mathcal{S}$, into the within-cluster scatter matrix, $\mathcal{S}_W$, and the between-cluster scatter matrix, $\mathcal{S}_B$ (see Appendix D).

$$\mathcal{S} = \mathcal{S}_B + \mathcal{S}_W$$

The total scatter matrix $\mathcal{S}$ is fixed, no matter how the given patterns are partitioned. These matrices are scalars in one dimension, so it should be clear that increasing the between-cluster scatter $\mathcal{S}_B$ decreases the within-cluster scatter $\mathcal{S}_W$, and vice versa. To define clustering criteria in terms of scatter matrices, we need to represent the "size" of clusters by the trace and the determinant operators. A number of clustering criteria are described below in terms of scatter (Friedman and Rubin, 1967).

We first show that a clustering criterion defined by either the trace of $\mathcal{S}_W$ or $\mathcal{S}_B$ is identical to the square-error criterion.

$$\text{tr}\,(\mathcal{S}_W) = \sum_{k=1}^{K} \text{tr}\,(\mathcal{S}^{(k)}) = \sum_{k=1}^{K} \sum_{i=1}^{n_k} (\mathbf{x}_i^{(k)} - \mathbf{m}^{(k)})^{\mathrm{T}}(\mathbf{x}_i^{(k)} - \mathbf{m}^{(k)}) = E_K^2$$

Note that tr $(\mathcal{S}^{(k)})$ is the sum of variances along the feature directions for cluster k and measures the compactness of cluster C_k. Minimizing tr $(\mathcal{S}_W)$ is identical to maximizing tr $(\mathcal{S}_B)$ because

$$\text{tr}\,(\mathcal{S}) = \text{tr}\,(\mathcal{S}_W) + \text{tr}\,(\mathcal{S}_B)$$

and tr $(\mathcal{S})$ is the same for any partition. The trace criterion, and the equivalent square-error criterion, are invariant under orthogonal transformations (Appendix D) of the pattern space, such as rotations, but are not invariant under nonsingular linear transformations. That is, the minimum square-error partition may change if the coordinate axes are scaled.

The determinant of $\mathcal{S}_B$ is not a good clustering criterion because $\mathcal{S}_B$ becomes singular when the number of clusters is less than the number of features, $K < d$. Minimizing the determinant of $\mathcal{S}_W$ has an advantage over the square-error criterion in that $\mathcal{S}_W$ is invariant to nonsingular linear transformations (Appendix D) of the patterns. However, $\mathcal{S}_W$ becomes singular if $(n - K) < d$ or if the patterns lie in a subspace of the feature space. A linear transformation (Sections 2.4.1 and 2.4.3) can be used to reduce the dimensionality of the data when $\mathcal{S}_W$ is singular.

Section 2.4.3 showed that the eigenvectors of $\mathcal{S}_W^{-1}\mathcal{S}_B$ define a projection of the patterns to a space of $K - 1$ dimensions. These eigenvectors are invariant under nonsingular linear transformations of the pattern matrix. Further, since the eigenvalues of $\mathcal{S}_W^{-1}\mathcal{S}_B$ determine the ratio of between-cluster to within-cluster scatter, we can define two additional clustering criterion. Assuming that there are $m \leq \min\{d, K - 1\}$ significant eigenvalues, $\{\zeta_1, \zeta_2, \ldots, \zeta_m\}$ of $\mathcal{S}_W^{-1}\mathcal{S}_B$, these criteria can be expressed as

$$\text{tr}\,(\mathcal{S}_W^{-1}\mathcal{S}_B) = \sum_{i=1}^{m} \zeta_i \qquad \text{and} \qquad |\mathcal{S}|/|\mathcal{S}_W| = \prod_{i=1}^{m} (1 + \zeta_i)$$

Clustering methods based on these criteria choose that partition for which tr $(\mathcal{S}_W^{-1}\mathcal{S}_B)$ or $|\mathcal{S}|/|\mathcal{S}_W|$ is maximized. Note that maximizing $|\mathcal{S}|/|\mathcal{S}_W|$ is the same as minimizing $|\mathcal{S}_W|$ because $|\mathcal{S}|$ is independent of the partition of the data.

The clustering criteria above look for globular or hyperellipsoidal clusters. Different criterion lead to different clusterings. The square-error criterion is less demanding computationally than the criteria based on scatter ratios because the latter require the computation of eigenvalues after every partition. Unfortunately, there is no general guideline available for choosing one criterion over the other. In practice, one should generate partitions using different criterion functions and then choose the "best" one under some validation scheme (Chapter 4).

Example 3.8

Suppose that the four two-dimensional pattern vectors shown in Figure 3.17 are to be clustered (Duda and Hart, 1973). It is easy to compute the scatter matrices for the following three partitions (Duda and Hart, 1973).

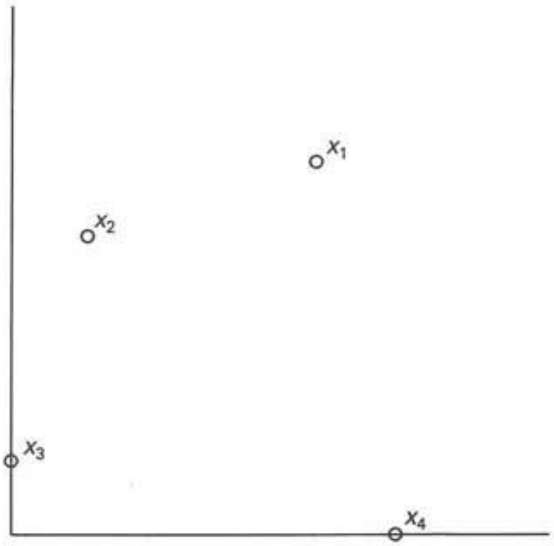


Figure 3.17 Clustering of four patterns in two dimensions.

$(\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4\})$	tr $(\mathcal{S}_W) = 18$	$\lvert\mathcal{S}_W\rvert = 16$
$(\{\mathbf{x}_1, \mathbf{x}_4\}, \{\mathbf{x}_2, \mathbf{x}_3\})$	tr $(\mathcal{S}_W) = 18$	$\lvert\mathcal{S}_W\rvert = 16$
$(\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}, \{\mathbf{x}_4\})$	tr $(\mathcal{S}_W) = 52/3$	$\lvert\mathcal{S}_W\rvert = 64/3$

Thus the third partition is best of the three according to the tr $(\mathcal{S}_W)$ criterion, which is equivalent to the square-error criterion. However, the first two partitions are selected by the $|\mathcal{S}_W|$ criterion.

3.3.2 Square-Error Clustering Methods

The basic idea of an iterative clustering algorithm is to start with an initial partition and assign patterns to clusters so as to reduce square-error. The square-error tends to decrease as the number of clusters increases and can be minimized only for a fixed number of clusters. An iterative partitional clustering method can be implemented in several different ways. Different implementations can lead to different partitions. Dubes and Jain (1976) emphasize the distinction between clustering methods and clustering algorithms. A clustering method specifies the general strategy for grouping the patterns into clusters such as minimizing square-error or maximizing tr $(\mathcal{S}_W^{-1}\mathcal{S}_B)$. A clustering algorithm, on the other hand, is a computer program that implements a strategy and incorporates various heuristics. A general algorithm for iterative partitional clustering method is given below. Anderberg (1973) provides an extensive discussion of several details of this approach.

ALGORITHM FOR ITERATIVE PARTITIONAL CLUSTERING

Step 1. Select an initial partition with K clusters.

Repeat steps 2 through 5 until the cluster membership stabilizes.

Step 2. Generate a new partition by assigning each pattern to its closest cluster center.

Step 3. Compute new cluster centers as the centroids of the clusters.

Step 4. Repeat steps 2 and 3 until an optimum value of the criterion function is found.

Step 5. Adjust the number of clusters by merging and splitting existing clusters or by removing small, or outlier, clusters.

The details of the steps in this algorithm must either be supplied by the user as parameters or be implicitly hidden in the computer program. However, these details are crucial to the success of the program. A big frustration in using clustering programs is the lack of guidelines available for choosing details. We briefly review some of the crucial parameters and the options available (Anderberg, 1973; Dubes and Jain, 1980).

Initial partition. An initial partition can be formed by first specifying a set of K seed points. Seed points can be the first K patterns or K patterns chosen randomly from the pattern matrix. A set of K patterns that are well separated from each other can be obtained by taking the centroid of the data as the first seed point and selecting successive seed points which are at least a certain distance away from the seed points already chosen. The initial partition or clustering is formed by assigning each pattern to the closest seed point. The centroids of the resulting clusters are the initial cluster centers. Hierarchical clustering of the data can also be used to select an initial partition.

Different initial partitions can lead to different final clusterings because algorithms based on square-error can converge to local minima. This is especially true if the clusters are not separated well. One way to overcome local minima is to run the partitional algorithm with several different initial partitions. If they all lead to the same final partition, we have some confidence that the global minimum of square-error has been achieved.

Updating the partition. Partitions are updated by reassigning patterns to clusters in an attempt to reduce the square-error. The term "pass" or "cycle" refers to the process of examining the cluster label of every pattern once. McQueen (1967) defined a *K-means pass* as an assignment of all patterns to the closest cluster center. The center of the gaining cluster is recomputed after each new assignment in McQueen's K-means method. Forgy's method (Forgy, 1965) recomputes cluster centers after all patterns have been examined. The Euclidean metric is the most common metric for computing the distance between a pattern and a cluster center but Mahalanobis distance (Section 2.2.1) is also used. However, Mahalanobis distance requires computation of the inverse of the sample covariance matrix every time a pattern changes its cluster label.

Friedman and Rubin (1967) define a *hill-climbing pass* and a *forcing pass* in their clustering algorithm based on an invariant criteria using scatter matrices. A hill-climbing pass changes the cluster label of a pattern only to improve the

criterion function. Remember that a K-means pass assigns every pattern to its closest cluster center. A forcing pass perturbs the partition to avoid getting trapped at a local minimum of the criterion function. A forcing pass tries each pattern of a cluster in a different cluster. The criterion function is recalculated after each test, the best partition found is retained, and the forcing pass is repeated for the next cluster. These passes are applied repeatedly until convergence is obtained.

Adjusting the number of clusters. Some clustering algorithms can create new clusters or merge existing clusters if certain conditions are met. This capability allows an algorithm to recover from poor initial partitions and lets it select a "natural" or "suitable" number of clusters, especially if the number of clusters desired is not appropriate. In one of the popular partitional clustering algorithms called ISODATA (Ball and Hall, 1964), these conditions are determined from parameters specified by the user of the program. A cluster is split if it has too many patterns and an unusually large variance along the feature with largest spread. Two clusters are merged if their cluster centers are sufficiently close, again based on a parameter supplied by the user.

An outlier is a pattern that is sufficiently far removed from the rest of the data to suspect that it was included by error, such as a mistake in data entry. Quite often an outlier is due to noise in the measurement process or error in data coding. Outliers can provide useful information about the underlying data generation process, but forcing an outlier to belong to a cluster distorts the shape of that cluster. Figure 3.18 demonstrates that an outlier can force a partitional clustering

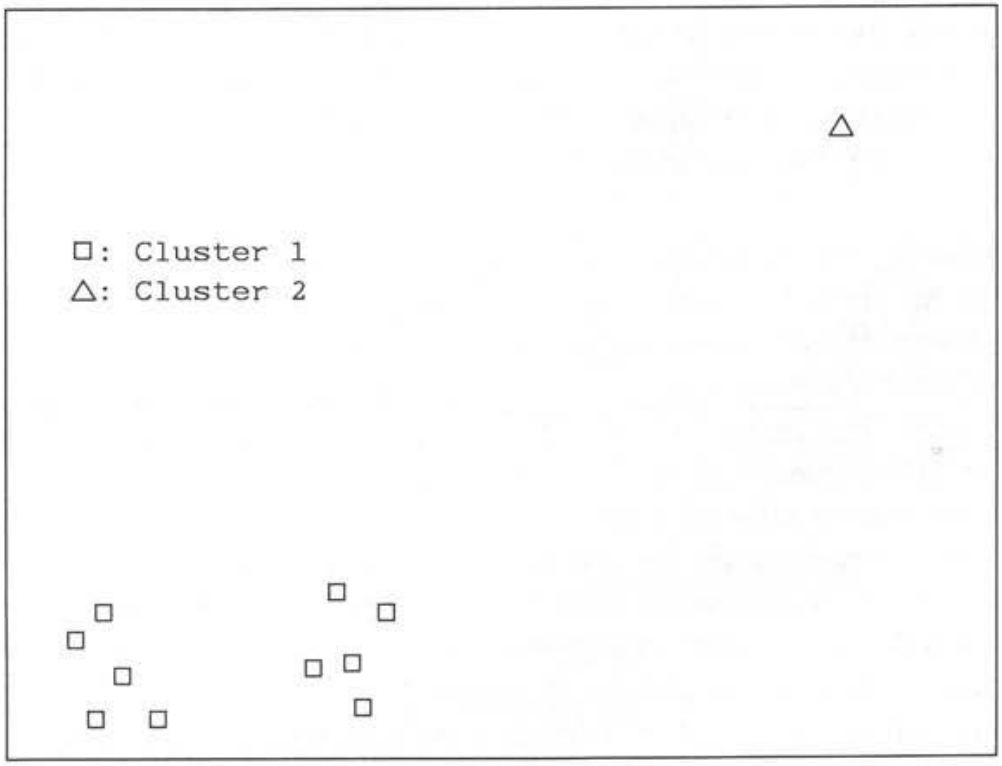


Figure 3.18 Effect of outlier in distorting a clustering.

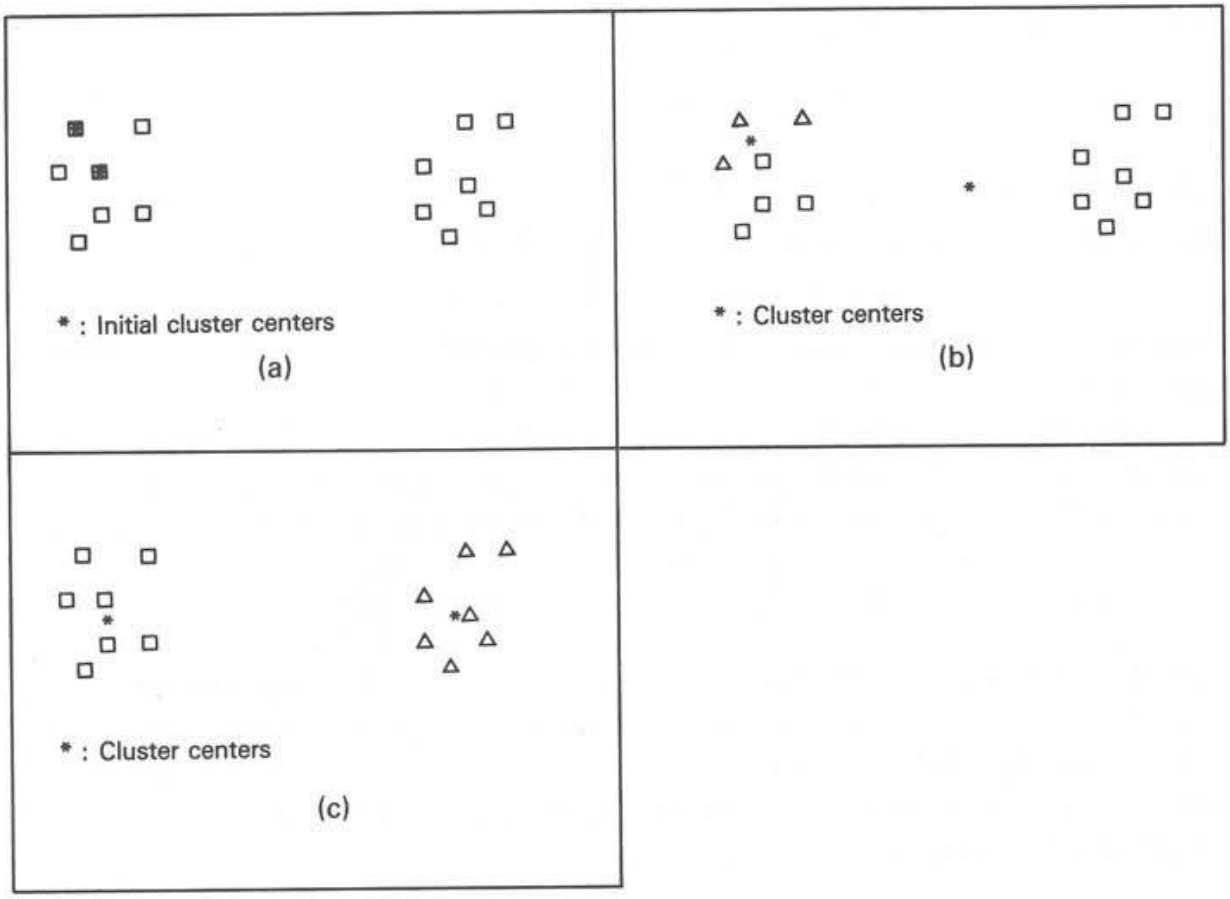


Figure 3.19 Convergence of K-means clustering: (a) initial data; (b) cluster membership after first loop; (c) cluster membership after second loop.

algorithm to put two compact and well-separated groups into the same cluster. Thus it is best to identify an outlier and remove it from further consideration. Some clustering algorithms also treat "small" clusters as outliers.

Convergence. When does the algorithm stop? Partitional algorithms terminate when the criterion function cannot be improved. There is no guarantee that an iterative algorithm will reach a global minimum. Some algorithms stop when the cluster labels for all the patterns do not change between two successive iterations. A maximum number of iterations can be specified to prevent endless oscillations. In practice, K-means type algorithms converge rapidly. Figure 3.19 shows two well-separated clusters in two dimensions. Even though the two initial seed points belong to the same cluster, the convergence of the K-means algorithm to the correct partition requires only two iterations.

Selim and Ismail (1984) rigorously prove convergence of the K-means algorithm. The problem of partitioning n d-dimensional patterns into K clusters can be formulated as the following mathematical programming problem. Minimize the weighted sum of Euclidean distances between patterns and cluster centers,

$$f(\mathcal{W}, \mathcal{M}) = \sum_{k=1}^{K} \sum_{i=1}^{n} w_{ki} d(\mathbf{x}_i, \mathbf{m}^{(k)})$$

subject to the constraint

$$\sum_{k=1}^{K} w_{ki} = 1, \qquad i = 1, 2, \ldots, n \quad \text{and} \quad w_{ki} \in \{0, 1\}$$

The matrix $\mathcal{W} = [w_{ki}]$ is a $K \times n$ matrix of weights for each pattern in each cluster and $\mathcal{M}$ is the $d \times K$ matrix of cluster centers.

$$\mathcal{M} = [\mathbf{m}^{(1)} \quad \mathbf{m}^{(2)} \quad \cdots \quad \mathbf{m}^{(K)}]$$

This is similar to the formulation of Gordon and Henderson (1977) discussed in Section 3.3.1.

The function $f(\mathcal{W}, \mathcal{M})$ is nonconvex and its local minimum need not be a global minimum. Frieze (1980) has also studied similar optimization problems. Consider the data set shown in Figure 3.20. If the two initial cluster centers are

$$\mathbf{m}^{(1)} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \qquad \text{and} \qquad \mathbf{m}^{(2)} = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$$

then the K-means algorithm will stop after one iteration and converge to the solution $w_{11} = w_{12} = w_{23} = w_{24} = 1$, $w_{ij} = 0$ otherwise, which yields two clusters $\{\mathbf{x}_1, \mathbf{x}_2\}$ and $\{\mathbf{x}_3, \mathbf{x}_4\}$ and $f(\mathcal{W}, \mathcal{M}) = 8$. Selim and Ismail show that this solution is not even a local minimum because by slightly perturbing $\mathcal{M}$, $f(\mathcal{W}, \mathcal{M}) = 6$ can be achieved with the cluster centers

$$\mathbf{m}^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \text{and} \qquad \mathbf{m}^{(2)} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

Pollard (1981) also provides conditions for the almost-sure convergence of the cluster centers in K-means clustering as the number of patterns increases.

Computation. The computational complexity of this algorithm is of the order $O(ndKT)$, where n is the number of patterns, d the number of features, K

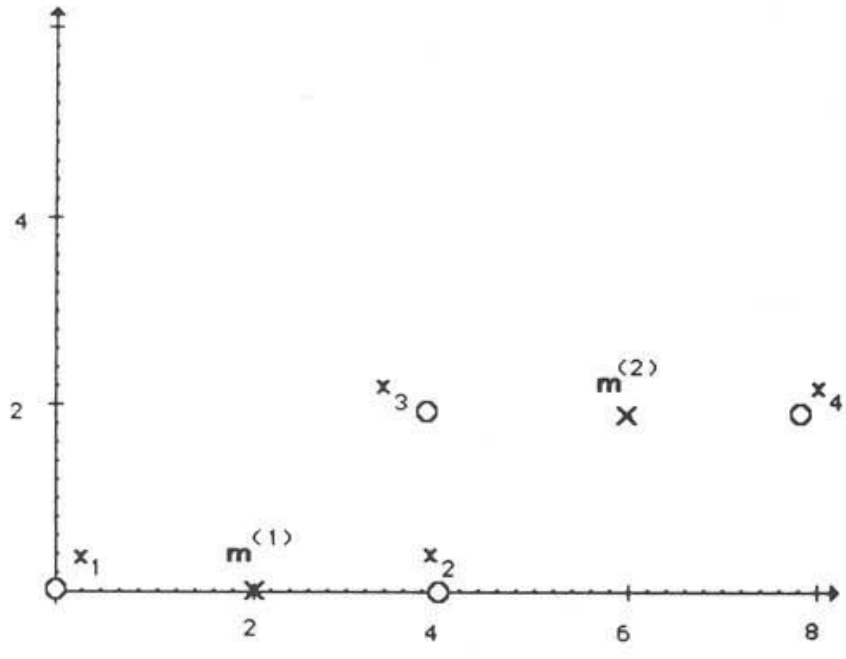


Figure 3.20 Convergence of square-error clustering algorithm.

the number of clusters desired, and T the number of iterations. The value of T depends on the initial cluster centers, distribution of patterns, and the size of the clustering problem. In practice, however, the user specifies an upper bound on the value of T. The iterative nature of the square-error clustering methods demands substantial processing time, even for a few hundred patterns. Two approaches have recently been taken to speed up square-error clustering algorithms that utilize advances in microelectronics technology: parallel processing and VLSI architecture. Tilton and Strong (1984) describe the performance of the ISODATA clustering algorithm when implemented on the MPP (Massively Parallel Processor), which contains an array of 128 × 128 microprocessors. A clustering problem involving a 512 × 512 digital image and 16 clusters required 20 seconds on the MPP compared to 7 hours on a VAX-11/780 superminicomputer. (See Chapter 5 for a brief review of digital image processing and image segmentation.) Ni and Jain (1985) present a systolic architecture for a square-error pattern clustering whose design has a potential performance gain of 1300 times over a serial processor.

3.3.3 Square-Error Clustering Programs

We now explain two examples of square-eror clustering programs, called FORGY and CLUSTER and compare their performance on two data sets. Clustering software is discussed in Section 3.4.

FORGY is the simplest and most straightforward square-error clustering program (Forgy, 1965). It uses only the K-means pass. The cluster centers are updated by recomputing the centroids of all patterns having the same cluster label at the end of the pass. The seed points are K patterns chosen at random, where K is specified by the user as the number of clusters desired. Our implementation of FORGY allows the user to specify a heuristic that creates additional clusters (Dubes and Jain, 1976). After the square-error has converged for a fixed K, a new cluster is created when a pattern is found that is sufficiently far removed from the existing cluster centers. The average distance between pattern $\mathbf{x}_i$ and the K cluster centers is given by

$$\bar{d}_i = (1/K) \sum_{k=1}^{K} d(\mathbf{x}_i, \mathbf{m}^{(k)})$$

A new cluster is created centered at pattern $\mathbf{x}_i$ if

$$|d(\mathbf{x}_i, \mathbf{m}^{(q)}) - \bar{d}_i| \leq \bar{d}_i T_1$$

where the qth cluster center is the cluster center closest to pattern $\mathbf{x}_i$ and T_1 is a user-specified parameter, $0 < T_1 < 1$. The left side of the above inequality is roughly $\bar{d}_i$ for patterns close to an existing cluster and is small for patterns far removed from all existing clusters. The larger T_1, the more new clusters are created. FORGY also detects outliers. If the number of patterns in any cluster falls below another user-specified parameter, T_2, then all the patterns belonging to that cluster are considered as outliers and are ignored.

To run FORGY, a user specifies threshold T_1 for creating new clusters,

threshold T_2 for deleting clusters, and the maximum allowable number of iterations. The most crucial parameter is T_1 and a few runs of the program with different values of T_1 and T_2 may be necessary to obtain a reasonable grouping of the data. FORGY should be run several times with different starting configurations.

The output of FORGY provides cluster labels for all patterns and several statistics such as the square error for each cluster, a table of distances between cluster centers, and the ratio of within-cluster to between-cluster distance. A table showing the number of patterns in each category is also printed when a priori category labels for all patterns are available. Since FORGY is based on square-error, it also generates some statistics that can be applied to analyze the separations among patterns. Since FORGY tries to group the patterns, an analysis of variance cannot be applied to these statistics. This point is treated at length in Chapter 4. Example 3.9 demonstrates the expected output of FORGY. The statistics are explained below.

The square-error for a clustering, E_K^2, can be decomposed into a feature-by-feature sum as shown below.

$$E_K^2 = \sum_{j=1}^{d} f_j^2$$

The contribution of feature j to the square-error, f_j^2, is the sum of the squared differences between each pattern and its cluster center using only feature j.

$$f_j^2 = \sum_{k=1}^{K} \sum_{i=1}^{n_k} [x_{ij}^{(k)} - m_j^{(k)}]^2$$

In terms of the standard decomposition (Appendix D), f_j^2 is the "within-cluster" variation in feature j. The "between-cluster" component, b_j^2, can be expressed in terms of the cluster centers and m_j, the jth coordinate of the centroid of all patterns.

$$b_j^2 = \left[\sum_{k=1}^{K} n_k (m_j^{(k)})^2 \right] - n(m_j)^2$$

$$m_j = \frac{1}{n} \sum_{k=1}^{K} \sum_{i=1}^{n_k} x_{ij}^{(k)} = \frac{1}{n} \sum_{k=1}^{K} n_k m_j^{(k)}$$

Discriminant analysis (Appendix D) shows that the total square-error contribution from feature j, a_j^2, can be written as

$$a_j^2 = f_j^2 + b_j^2$$

The F-ratio is defined as

$$F\text{-ratio} = \frac{b_j^2/(K-1)}{f_j^2/(n-K)}$$

The name "F-ratio" comes from analysis of variance (Appendix F). When the samples are independent and come from Gaussian distributions and when the vari-

ances of all groups are the same and when the group labels are assigned a priori, the F-ratio has an F distribution with $K - 1$ and $n - K$ degrees of freedom. Large values of the F-ratio, when measured on the scale of an F distribution, indicate a grouping in which the separation among clusters is significantly large with respect to the separations among patterns in individual clusters. Unfortunately, this distribution cannot be applied to determine whether a particular feature contributes significantly to the clustering because the cluster labels are assigned *after* looking at the data. For example, a clustering algorithm labels the patterns so as to separate the clusters maximally. The null distribution of the F-ratio printed out by the program is not the standard distribution published in textbooks.

A few other quantities defined in FORGY's output need clarification. For cluster number k, the "squared error" is e_k^2, "S.E./$(N(k) - 1)$" is $e_k^2/(n_k - 1)$, and "CLAVGD(k)" is $(e_k^2/n_k)^{1/2}$. Other terms are self-explanatory. Note that "distance" means "squared Euclidean distance."

Example 3.9

FORGY was applied to data sets DATA1 and DATA2 (Example 2.6). DATA1 consists of 100 patterns arranged into four distinct clusters in a four-dimensional unit hypercube. Patterns are arranged by category. The first 24 patterns are from category 1, the next 35 from category 2, the next 21 from category 3, and the last 20 from category 4. DATA2 consists of 100 patterns uniformly generated in a six-dimensional unit hypercube. The following parameter settings were tried for both sets of data:

$K = 2, 4, 6$ $\quad T_1 = 0.5, \quad T_2 = 1$ $\quad$ maximum number of iterations = 20

The objective here was not to finely tune the parameters to get the best clustering, but to see if reasonable clusterings can be obtained. FORGY converged in fewer than 10 iterations on both data sets. The two-cluster solution took about 12 seconds of CPU time on a Harris 500 superminicomputer. The execution time increased to 35 seconds when the number of clusters was increased to six.

We first summarize the results for DATA1, where the true cluster numbers generated by computer are treated as category information. Part of the output for the two-cluster solution is shown in Figure 3.21. The output shows that categories 1 and 2 are grouped into one cluster and categories 3 and 4 in the other cluster. This grouping is surprising in light of the two-dimensional representation in Figure 2.10 and might be explained by the observation that the clusters are hyperellipsoidal with nearly identical covariance structure. The F-ratio for feature 4 is largest, so the patterns cluster better in feature 4 than in the other 3 features. We cannot say whether the clustering in feature 4 is significantly best. Only patterns 13, 16, and 89 fail to cluster properly. The main output of the program is the locations of the two cluster centers. The total square error for the two-cluster solution is 17.406 and seems evenly divided between the two clusters.

Only the most useful information in FORGY output is displayed for the four-cluster (Figure 3.22) and six-cluster (Figure 3.23) solutions. The total square-error reduces to 6.8396 for the four-cluster solution and to 3.4096 for the six-cluster solution. Categories 1 and 4 are confused in the four-cluster solution, whereas they are separated in the two-cluster solution. ~~Feature 4 is not as important to the four-cluster solution as it was to the two-cluster solution according to the F-ratio.~~ The six-cluster solution separates patterns

Feature 4 is more important to the two-cluster solution than feature 2 according to the F-ratio.

Results of FORGY algorithm after 6 iterations

0 Patterns were removed.
2 Clusters were obtained.

Cluster No. for each pattern—LABEL array

1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	2	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2

Variation about cluster means

Feature	Squared error	Between clusters	F-Ratio
1	2.5197	18.456	717.80
2	7.3108	41.203	552.32
3	5.1391	37.489	714.89
4	2.4365	35.665	1434.5

Squared error per cluster—K is the cluster No.

K	N(K)	Squared error	S.E./(N(K)-1)	CLAVGD(K)
1	58	8.1296	0.14262	0.37439
2	42	9.2766	0.22626	0.46997
TOTAL	100	17.406		

Squared error divided by (No. of patterns in clustering—No. of clusters) = 0.17761

Average error or root of (Squared error divided by NPAT-NDELE) = 0.41721

Distances between cluster centers

	1	2
1	0.0000	0.51911
2	0.51911	0.0000

Average distance between each cluster center and the other centers
0.51911 0.51911

Average of the Avg. distances listed directly above =0.51911

Minimum of the distances between each cluster center and the other cluster centers
0.51911 0.51911

For each cluster, the ratio of the average distance between that cluster center and all others to the average within-cluster distance
1.3866 1.1046

For each cluster, the ratio of the minimum of the distances between that cluster center and all others to the average within-cluster distance
1.3866 1.1046

Cluster membership according to category:
Rows are clusters and columns are categories

	1	2	3	4
1	22	35	0	1
2	2	0	21	19

CLUSTER CENTERS

1	0.509	0.727	0.736	0.771
2	0.301	0.520	0.402	0.204

Figure 3.21 FORGY clustering on DATA1: two clusters.

```
Results of  FORGY  algorithm after  6  iterations

  0 Patterns were removed.
  4 Clusters were obtained.


Cluster No. for each pattern - LABEL array

  2  2  2  2  2  2  2  2  2  2        2  2  2  2  2  2  2  2  2  2
  2  2  2  2  3  3  3  3  3  3        3  3  3  3  3  3  3  3  3  3
  3  3  3  3  3  3  3  3  3  3        3  3  3  3  3  3  3  3  3  1
  4  1  4  1  1  4  1  4  1  1        4  1  1  4  4  1  1  1  1  4
  2  2  2  2  2  2  2  2  2  2        2  2  2  2  2  2  2  2  2  2


Squared error per cluster - K is the cluster No.

      K   N(K)     Squared error     S.E./(N(K)-1)     CLAVGD(K)
      1     13       0.20735          0.17279E-01       0.12629
      2     44        5.1956          0.12083           0.34363
      3     35        1.2852          0.37799E-01       0.19162
      4      8       0.15144          0.21634E-01       0.13759
  TOTAL    100        6.8396

Squared error divided by (No  of patterns in  clustering - No. of clusters) =
    0.71246E-01


Distances between cluster centers

        1             2            3            4
1    0.0000       0.76497       1.0242      0.53347E-01
2   0.76497        0.0000      0.52729      0.73402
3    1.0242       0.52729       0.0000       1.2203
4   0.53347E-01  0.73402       1.2203       0.0000


Cluster membership according to category:
Rows are clusters and columns are categories
        1    2    3    4
1       0    0   13    0
2      24    0    0   20
3       0   35    0    0
4       0    0    8    0
```

Figure 3.22 FORGY clustering on DATA1: four clusters.

into clusters according to category except for patterns 13 and 16. The origins of these two patterns should probably be inspected. Categories 2 and 4 are split into two clusters, so a bimodal structure might be appropriate for describing these categories.

How many clusters are appropriate for these data? One of the heuristics for deciding the number of clusters is to look for a "knee" in the plot of total square-error versus the number of clusters (see Section 4.4.2). This heuristic suggests that DATA1 has four clusters. The problem of fixing the "correct" number of clusters is one of the most fundamental and unsolved problems in cluster analysis.

One useful statistic printed by FORGY is the ratio of the distances between a cluster center and all others to the average within-cluster distance. Large values of this statistic suggest that the clusters are compact and well separated. These values are substantially higher for both the four-cluster and the six-cluster solution than for the two-cluster solution, although these statistics naturally increase with the number of clusters. In the four-cluster solution, only cluster 2, which is a mixture of categories 1 and 4, has a low value for this statistic.

```
Results of  FORGY  algorithm after  9  iterations

  0 Patterns were removed.
  6 Clusters were obtained.


Cluster No. for each pattern - LABEL array

  6  6  6  6  6  6  6  6  6  6        6  6  4  6  6  4  6  6  6  6
  6  6  6  6  2  5  2  5  5  5        5  5  2  5  5  5  5  2  2  2
  5  5  2  5  5  2  5  5  2  5        5  5  5  5  2  2  2  5  5  3
  3  3  3  3  3  3  3  3  3  3        3  3  3  3  3  3  3  3  3  3
  1  1  1  1  4  1  1  1  1  4        1  4  4  1  1  4  1  1  1  1


Squared error per cluster - K is the cluster No.

      K   N(K)     Squared error     S.E./(N(K)-1)     CLAVGD(K)
      1     15       0.43799          0.31285E-01       0.17088
      2     12       0.25361          0.23056E-01       0.14538
      3     21       0.62299          0.31149E-01       0.17224
      4      7       0.30889          0.51482E-01       0.21006
      5     23       0.71969          0.32713E-01       0.17689
      6     22        1.0664          0.50783E-01       0.22017
  TOTAL    100        3.4096

Squared error divided by (No  of patterns in  clustering - No. of clusters) =
    0.36272E-01


Distances between cluster centers

        1             2             3             4             5             6
  1   0.0000       0.88037       0.80797      0.85551E-01   0.84843       0.32988
  2   0.88037       0.0000       0.97198       0.74127      0.39550E-01   0.39932
  3   0.80797      0.97198        0.0000       0.61857       1.1596       0.88723
  4   0.85551E-01  0.74127       0.61857        0.0000       0.70367       0.20117
  5   0.84843      0.39550E-01    1.1596       0.70367       0.0000       0.41891
  6   0.32988      0.39932       0.88723       0.20117      0.41891        0.0000


Cluster membership according to category:
Rows are clusters and columns are categories

        1    2    3    4
  1     0    0    0   15
  2     0   12    0    0
  3     0    0   21    0
  4     2    0    0    5
  5     0   23    0    0
  6    22    0    0    0
```

Figure 3.23 FORGY clustering on DATA1: six clusters.

The patterns in DATA2 have no category information, so FORGY cannot construct cluster by category tables to judge the performance of the clustering program. We can contrast the output of FORGY for these random data with the clustered data in DATA1 but must be aware that the numbers of features is not the same. Figures 3.24 to 3.26 show that the square-error for DATA2 is substantially higher than for DATA1, and the total square-error for DATA2 does not fall as sharply as for DATA1 when the number of clusters is increased. Both sets of data have the same number of patterns, but DATA2 is in six dimensions. Again, the complete output for FORGY on DATA2 is shown only for the two-cluster solution (Figure 3.24).

Another noticeable difference between clustering for the two data sets appears in

Results of FORGY algorithm after 6 iterations

0 Patterns were removed.

2 Clusters were obtained.

Cluster No. for each pattern—LABEL array

1	1	2	2	2	1	2	1	1	1	1	1	2	2	1	1	2	1	2	2
2	1	1	2	1	1	1	1	2	1	2	1	1	2	1	1	1	1	1	1
1	1	1	1	2	2	2	1	1	1	1	1	1	1	2	1	1	2	1	1
2	2	1	2	1	1	1	1	1	1	2	1	2	2	2	1	2	1	2	2
1	1	2	1	1	1	1	2	1	2	1	1	1	2	1	2	1	2	1	2

Variation about cluster means

Feature	Squared error	Between clusters	F-Ratio
1	7.5990	22.963	296.14
2	7.4116	22.134	292.67
3	5.9815	28.352	464.52
4	7.1100	26.380	363.61
5	8.4855	27.928	322.54
6	6.4006	23.071	353.24

Squared error per cluster—K is the cluster No.

K	N(K)	Squared error	S.E./(N(K)-1)	CLAVGD(K)
1	64	27.509	0.43665	0.65561
2	36	15.479	0.44226	0.65572
TOTAL	100	42.988		

Squared error divided by (No. of patterns in clustering—No. of clusters) = 0.43865

Average error or root of (Squared error divided by NPAT-NDELE) = 0.65565

Distances between cluster centers

	1	2
1	0.0000	0.26452
2	0.26452	0.0000

Average distance between each cluster center and the other centers

0.26452 0.26452

Average of the Avg. distances listed directly above =0.26452

Minimum of the distances between each cluster center and the other cluster centers

0.26452 0.26452

For each cluster, the ratio of the average distance between that cluster center and all others to the average within-cluster distance

0.40348 0.40341

For each cluster, the ratio of the minimum of the distances between that cluster center and all others to the average within-cluster distance

0.40348 0.40341

CLUSTER CENTERS

1	0.368	0.525	0.405	0.564	0.558
	0.551				
2	0.640	0.369	0.714	0.426	0.487
	0.336				

Figure 3.24 FORGY clustering on DATA2: two clusters.

```
Results of  FORGY  algorithm after  10  iterations

  0 Patterns were removed.
  4 Clusters were obtained.


Cluster No. for each pattern - LABEL array

  1  4  4  1  1  2  1  4  3  4        3  1  4  1  2  4  1  2  3  1
  3  4  4  1  4  2  1  4  4  2        1  2  3  4  4  3  2  1  3  2
  2  4  2  2  4  1  1  2  4  2        2  4  2  3  1  1  1  1  2  4
  1  3  2  4  3  1  4  2  1  3        1  2  3  4  3  1  1  4  4  3
  4  4  1  3  1  2  2  1  2  1        1  4  3  3  3  1  3  1  4  1

Squared error per cluster - K is the cluster No.

      K   N(K)     Squared error     S.E./(N(K)-1)     CLAVGD(K)
      1     32        12.332           0.39782          0.62079
      2     22        6.1898           0.29475          0.53043
      3     19        5.1512           0.28618          0.52069
      4     27        9.6708           0.37195          0.59848
  TOTAL    100        33.344

Squared error divided by (No  of patterns in  clustering - No. of clusters) =
    0.34733

Distances between cluster centers

         1          2          3          4
1    0.0000     0.44283    0.39219    0.42436
2   0.44283      0.0000    0.30128    0.31992
3   0.39219     0.30128     0.0000    0.66478
4   0.42436     0.31992    0.66478     0.0000
```

Figure 3.25 FORGY clustering on DATA2: four clusters.

the ratio of the average distance between a cluster center and all others to the average within-cluster distance. This statistic has substantially lower values for random data than for the clustered data. Clustering tendency (Section 4.6) deals with the issue of whether a given data set is random.

Example 3.10

Figure 3.27(a) demonstrates a difficulty with square-error clustering. It shows the two-cluster solution generated by FORGY for a set of patterns in the plane which clearly contains two well-separated groups. Unfortunately, the partition boundary between the two clusters does not lie in the sparse region as expected but cuts one of the two "natural" clusters in half. Similarly, the cigar-shaped data in Figure 3.27(b) are not clustered correctly by FORGY. These examples demonstrate that the square-error criterion, which seeks compact hyperellipsoidal clusters, can produce misleading results when the data do not occur in compact, hyperellipsoidal boundaries.

The second square-error clustering program to be examined is called CLUSTER (Dubes and Jain, 1976). It has the same objective as FORGY but generates a nonhierarchical sequence of clusterings rather than a single clustering. This program utilizes both a K-means pass and a forcing pass (Friedman and Rubin, 1967). CLUSTER attempts to find the "best" clusterings containing 1, 2, . . . ,

```
Results of  FORGY  algorithm after  6  iterations

  0 Patterns were removed.
  6 Clusters were obtained.

Cluster No. for each pattern - LABEL array

  1  3  1  1  1  4  1  3  5  6        1  6  1  5  4  3  6  4  5  1
  5  1  3  6  3  4  1  3  3  4        6  2  5  2  3  5  4  1  5  4
  4  2  4  4  3  1  6  2  3  2        1  3  2  5  1  6  1  4  4  3
  5  6  2  3  5  1  6  2  1  5        1  4  5  3  4  6  4  3  3  5
  3  3  4  5  6  4  4  6  4  4        6  3  5  5  2  6  5  4  6  6

Squared error per cluster - K is the cluster No.

      K   N(K)     Squared error     S.E./(N(K)-1)     CLAVGD(K)
      1     18         5.7175           0.33633          0.56360
      2      9         1.6167           0.20209          0.42383
      3     19         6.0932           0.33851          0.56630
      4     21         6.9902           0.34951          0.57694
      5     17         4.5053           0.28158          0.51480
      6     16         3.8947           0.25965          0.49338
  TOTAL    100         28.818

Squared error divided by (No  of patterns in  clustering - No. of clusters) =
    0.30657

Distances between cluster centers

        1          2          3          4          5          6
  1   0.0000     0.65279    0.41074    0.44736    0.46324    0.49531
  2   0.65279    0.0000     0.32856    0.39610    0.34615    0.51358
  3   0.41074    0.32856    0.0000     0.46140    0.83177    0.61105
  4   0.44736    0.39610    0.46140    0.0000     0.36814    0.39783
  5   0.46324    0.34615    0.83177    0.36814    0.0000     0.60731
  6   0.49531    0.51358    0.61105    0.39783    0.60731    0.0000
```

Figure 3.26 FORGY clustering on DATA2: six clusters.

K clusters and prints a history of the K clusterings achieved, one for each number of clusters.

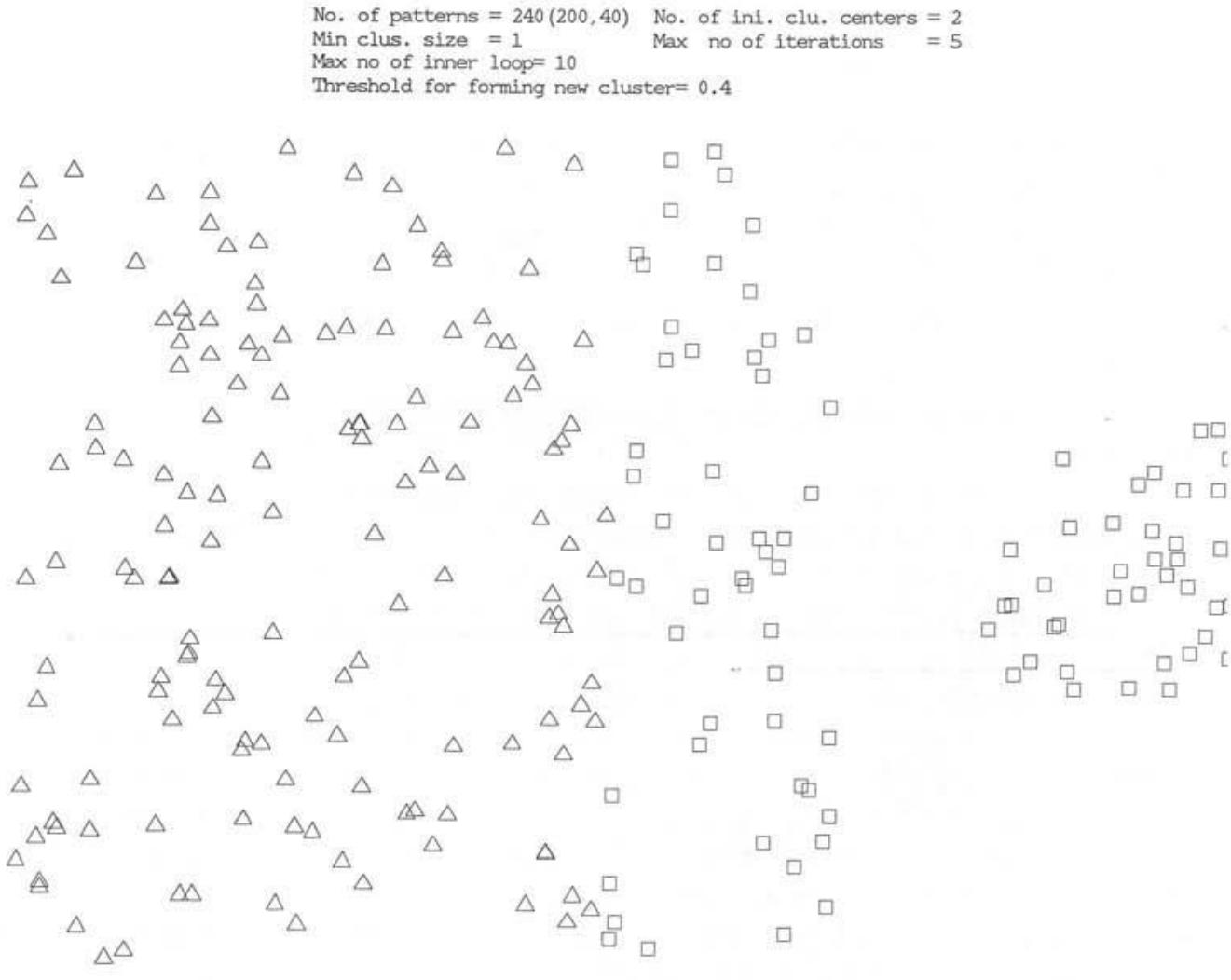
CLUSTER involves two phases which are repeated until a pass through both phases does not decrease the square-error. Phase 1 creates a sequence of clusterings containing 2, 3, . . . , K clusters, where K is specified by the user. The initial two cluster centers are the centroid of the patterns and the pattern farthest removed from the centroid, not counting the outliers. Given a clustering with k clusters, the pattern farthest removed from the existing clustering is identified as the $(k + 1)$st cluster center. The K-means pass is repeated until no patterns change clusters or until a maximum number of iterations have been completed.

The first pass through phase 1 gives a set of K clusterings, each containing a different number of clusters. Phase 2 then creates another set of clusterings by merging existing clusters two at a time to see if a better clustering can be achieved (forcing pass). After each pass through phases 1 and 2, the square-errors of the clusterings are compared with those of the clusterings (having the same number of clusters) that existed before that pass. If any of the square-errors are smaller than before, another pass through phases 1 and 2 is initiated. This continues until square-error cannot be decreased.

One of the main advantages of CLUSTER over other clustering programs is that the user need not specify parameters. Only the maximum number of clusters desired is required. The clustering statistic computed in CLUSTER is a purely heuristic number.

$$(1/K) \sum_{k=1}^{K} [(n_k/e_k^2)^{1/2} (n - n_k)^{-1} \sum_{\substack{r=1 \\ k \neq r}}^{K} n_r d^2(\mathbf{m}^{(k)}, \mathbf{m}^{(r)})]$$

That is, the ratio of the "average" distance from cluster k to all other clusters is divided by the average within-cluster distance for cluster k. This quantity is averaged over all clusters. Intuition dictates that the larger this number, the better the clustering. But how large is large? How does this statistic depend on problem parameters? These questions have not been answered. We now present the results of CLUSTER on DATA1 and DATA2 and compare these results with those given by program FORGY.



(a)

Figure 3.27 Inadequacies of square-error clustering.

(b)

Figure 3.27 (*continued*)

Example 3.11

The partitions obtained from CLUSTER for data sets DATA1 and DATA2 are given in Figures 3.28 and 3.29. They appear to be better than those generated by FORGY. For example, the four-cluster solution from CLUSTER for DATA1 has a total square-error of 4.0677 (Figure 3.28) compared to 6.8396 by FORGY. The partition from CLUSTER uniquely associates a cluster with each category with the exception of a single pattern from category 1. FORGY's four-cluster solution merged categories 1 and 4 in the same cluster. The performance of CLUSTER on the random data of DATA2 is comparable to that of FORGY.

In summary, clustering programs that minimize square-error are very practical. They try to define clusters that are hyperellipsoidal in shape. The square-error criterion is equivalent to several other criteria involving the scatter matrices used in discriminant analysis. The numerous square-error programs available differ both in computational details and in the approach taken to minimize the square error. Square-error clustering methods do exhibit inadequacies as when the Euclidean metric is used to measure distance but the features are not on comparable scales.

Program CLUSTER will group 100 patterns using
4 features. There are 4 categories.

Mean vector for raw data:

0.4214	0.6402	0.5956	0.5330

Standard deviations for raw data:

0.1890	0.2890	0.2805	0.3204

Begin output for clustering number 1

Clusters by category:

CLUSTER	1	2	3	4
1	24	35	21	20

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	100	30.052	0.30355	0.54819

Squared error divided by (No. patterns − 1) = 0.30355

Begin output for clustering number 2

2 Clusters: Cluster Membership Table

1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	2
2	2	2	2	2	2	2	2	2	2		2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1

Clusters by category:

CLUSTER	1	2	3	4
1	24	35	0	20
2	0	0	21	0

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	79	16.760	0.21487	0.46059
2	21	0.62299	0.31149E-01	0.17224
Total	100	17.383		

Sq. error divided by (No. patterns − No. of clusters) = 0.17737

Table of distances between cluster centers

CLUS	1	2
1	0.00	0.874
2	0.874	0.00

The clustering statistic: 3.0112

Begin output for clustering number 3

3 Clusters: Cluster Membership Table

3	3	3	3	3	3	3	3	3	3		3	3	3	3	3	3	3	3	3	3
3	3	3	3	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	2
2	2	2	2	2	2	2	2	2	2		2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3		3	3	3	3	3	3	3	3	3	3

Clusters by category:

CLUSTER	1	2	3	4
1	0	35	0	0
2	0	0	21	0
3	24	0	0	20

Figure 3.28 Results of CLUSTER on DATA1.

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	35	1.2852	0.37799E-01	0.19162
2	21	0.62299	0.31149E-01	0.17224
3	44	5.1956	0.12083	0.34363
Total	100	7.1038		

Sq. error divided by (No. patterns − No. of clusters) = 0.73235E-01

Table of distances between cluster centers

CLUS	1	2	3
1	0.00	1.04	0.726
2	1.04	0.00	0.861
3	0.726	0.861	0.00

The clustering statistic: 1.5205

Begin output for clustering number 4

4 Clusters: Cluster Membership Table

4	4	4	4	4	4	4	4	4	4		4	4	4	4	4	3	4	4	4	4
4	4	4	4	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	2
2	2	2	2	2	2	2	2	2	2		2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3		3	3	3	3	3	3	3	3	3	3

Clusters by category:

CLUSTER	1	2	3	4
1	0	35	0	0
2	0	0	21	0
3	1	0	0	20
4	23	0	0	0

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	35	1.2852	0.37799E-01	0.19162
2	21	0.62299	0.31149E-01	0.17224
3	21	0.99892	0.49946E-01	0.21810
4	23	1.1606	0.52756E-01	0.22464
Total	100	4.0677		

Sq. error divided by (No. patterns − No. of clusters) = 0.42372E-01

Table of distances between cluster centers

CLUS	1	2	3	4
1	0.00	1.04	0.900	0.633
2	1.04	0.00	0.867	0.929
3	0.900	0.867	0.00	0.526
4	0.633	0.929	0.526	0.00

The clustering statistic: 1.0908

Begin output for clustering number 5

5 Clusters: Cluster Membership Table

4	4	4	4	4	4	4	4	4	4		4	4	5	4	4	5	4	4	4	4
4	4	4	4	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	2
2	2	2	2	2	2	2	2	2	2		2	2	2	2	2	2	2	2	2	2
3	3	3	3	5	3	3	3	3	5		3	5	5	3	3	5	3	3	3	3

Clusters by category:

CLUSTER	1	2	3	4
1	0	35	0	0
2	0	0	21	0
3	0	0	0	15
4	22	0	0	0
5	2	0	0	5

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	35	1.2852	0.37799E-01	0.19162
2	21	0.62299	0.31149E-01	0.17224
3	15	0.43799	0.31285E-01	0.17088
4	22	1.0664	0.50783E-01	0.22017
5	7	0.30889	0.51482E-01	0.21006
Total	100	3.7215		

Sq. error divided by (No. patterns − No. of clusters) = 0.39174E-01

Figure 3.28 ***(continued)***

Table of distances between cluster centers

CLUS	1	2	3	4	5
1	0.00	1.04	0.922	0.635	0.841
2	1.04	0.00	0.899	0.942	0.786
3	0.922	0.899	0.00	0.574	0.292
4	0.635	0.942	0.574	0.00	0.449
5	0.841	0.786	0.292	0.449	0.00

The clustering statistic: 0.78831

Begin output for clustering number 6

6 Clusters: Cluster Membership Table

6	4	6	6	4	4	4	6	4	4		4	4	5	6	4	5	6	6	6	4
6	4	4	6	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	2
2	2	2	2	2	2	2	2	2	2		2	2	2	2	2	2	2	2	2	2
3	3	3	3	5	3	3	3	3	5		3	5	5	3	3	5	3	3	3	3

Clusters by category:

CLUSTER	1	2	3	4
1	0	35	0	0
2	0	0	21	0
3	0	0	0	15
4	12	0	0	0
5	2	0	0	5
6	10	0	0	0

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	35	1.2852	0.37799E-01	0.19162
2	21	0.62299	0.31149E-01	0.17224
3	15	0.43799	0.31285E-01	0.17088
4	12	0.36424	0.33113E-01	0.17422
5	7	0.30889	0.51482E-01	0.21006
6	10	0.33110	0.36789E-01	0.18196
Total	100	3.3504		

Sq. error divided by (No. patterns − No. of clusters) = 0.35642E-01

Table of distances between cluster centers

CLUS	1	2	3	4	5	6
1	0.00	1.04	0.922	0.674	0.841	0.615
2	1.04	0.00	0.899	0.961	0.786	0.939
3	0.922	0.899	0.00	0.606	0.292	0.568
4	0.674	0.961	0.606	0.00	0.431	0.261
5	0.841	0.786	0.292	0.431	0.00	0.506
6	0.615	0.939	0.568	0.261	0.506	0.000

The clustering statistic: 0.68259

Figure 3.28 ***(continued)***

Program CLUSTER will group 100 patterns using
6 features. There are 0 categories.

Mean vector for raw data:

0.4657	0.4692	0.5166	0.5145	0.5327
0.4738				

Standard deviations for raw data:

0.3051	0.2824	0.2860	0.2747	0.2933
0.2733				

Begin output for clustering number 1

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	100	49.083	0.49578	0.70059

Squared error divided by (No. patterns − 1) = 0.49578

Begin output for clustering number 2

2 Clusters: Cluster Membership Table

2	1	2	2	2	1	2	2	1	2	1	2	2	2	1	2	2	1	1	2
1	2	1	2	1	1	2	2	1	1	2	1	1	2	2	1	1	2	1	1
1	1	2	1	2	2	2	1	2	1	1	2	1	1	2	2	2	2	2	1
2	1	1	2	1	2	2	1	2	1	2	1	1	2	1	2	2	2	2	1
2	1	2	1	2	1	1	2	1	2	2	1	1	1	1	2	1	2	2	2

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	47	17.783	0.38658	0.61510
2	53	23.387	0.44975	0.66428
Total	100	41.170		

Sq. error divided by (No. patterns − No. of clusters) = 0.42010

Table of distances between cluster centers

CLUS	1	2
1	0.00	0.564
2	0.564	0.00

The clustering statistic: 0.65331

Begin output for clustering number 3

3 Clusters: Cluster Membership Table

3	1	2	2	2	1	3	2	1	3	1	3	2	3	1	2	3	1	1	3
1	2	1	3	1	1	3	2	2	1	3	1	1	2	2	1	1	3	1	1
1	1	3	1	2	2	3	1	2	1	1	2	1	1	3	3	3	2	3	2
3	3	1	2	1	2	3	1	3	1	2	3	1	2	1	3	3	2	2	1
2	1	3	1	3	1	1	3	1	3	3	1	1	1	1	3	1	3	3	3

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	43	15.884	0.37820	0.60779
2	24	8.3599	0.36348	0.59020
3	33	12.054	0.37668	0.60438
Total	100	36.298		

Sq. error divided by (No. patterns − No. of clusters) = 0.37421

Table of distances between cluster centers

CLUS	1	2	3
1	0.00	0.669	0.606
2	0.669	0.00	0.610
3	0.606	0.610	0.00

The clustering statistic: 0.49489

Begin output for clustering number 4

4 Clusters: Cluster Membership Table

4	1	2	2	2	1	3	2	4	3	4	3	2	3	1	2	3	1	1	4
1	2	1	3	1	4	4	2	2	1	3	1	4	3	2	4	4	3	4	1
1	1	4	1	2	4	3	1	2	1	4	2	1	1	3	3	3	2	3	2
4	3	1	2	1	4	3	1	4	4	4	1	4	2	1	3	4	2	2	1
2	1	2	4	3	1	1	3	1	3	3	1	4	1	1	3	4	3	3	3

Figure 3.29 Results of CLUSTER on DATA2.

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	31	10.590	0.35301	0.58449
2	21	6.3966	0.31983	0.55190
3	25	7.9114	0.32964	0.56254
4	23	8.1612	0.37097	0.59568
Total	100	33.060		

Sq. error divided by (No. patterns − No. of clusters) = 0.34437

Table of distances between cluster centers

CLUS	1	2	3	4
1	0.00	0.659	0.659	0.583
2	0.659	0.00	0.654	0.764
3	0.659	0.654	0.00	0.638
4	0.583	0.764	0.638	0.00

The clustering statistic: 0.36890

Begin output for clustering number 5

5 Clusters: Cluster Membership Table

4	5	2	2	2	1	3	2	4	3	4	3	2	4	1	5	3	1	1	4
1	2	5	3	5	4	4	2	2	1	3	1	4	3	2	4	4	3	4	1
1	5	2	1	2	4	3	1	5	1	4	5	1	4	3	3	3	2	3	5
4	3	1	2	4	4	3	1	4	4	4	1	4	2	1	3	4	2	2	1
5	5	2	4	3	1	1	3	1	2	3	5	4	1	1	3	4	2	3	3

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	23	6.6293	0.30133	0.53687
2	19	5.9713	0.33174	0.56061
3	22	6.2765	0.29888	0.53413
4	25	8.5578	0.35658	0.58507
5	11	2.4206	0.24206	0.46910
Total	100	29.856		

Sq. error divided by (No. patterns − No. of clusters) = 0.31427

Table of distances between cluster centers

CLUS	1	2	3	4	5
1	0.00	0.718	0.667	0.567	0.695
2	0.718	0.00	0.681	0.760	0.765
3	0.667	0.681	0.00	0.654	0.796
4	0.567	0.760	0.654	0.00	0.823
5	0.695	0.765	0.796	0.823	0.00

The clustering statistic: 0.38778

Begin output for clustering number 6

6 Clusters: Cluster Membership Table

6	5	2	2	2	3	3	2	1	3	4	3	2	4	6	5	3	1	1	4
1	2	5	3	5	6	4	2	2	1	3	1	4	1	2	4	6	4	4	6
6	5	6	3	2	4	4	1	3	5	6	5	1	1	4	3	3	2	6	5
4	3	5	2	1	4	3	1	4	4	4	6	1	2	1	3	4	2	2	1
5	5	2	4	3	6	6	3	1	2	3	5	4	1	1	4	4	2	3	3

Squared error for cluster number J

J	N(J)	SQUARED ERROR	S. E./(N(J)-1)	CLAVGD(J)
1	18	4.3726	0.25721	0.49287
2	18	5.2696	0.30998	0.54107
3	19	5.1522	0.28263	0.52074
4	21	6.7206	0.33603	0.56571
5	12	2.3807	0.21643	0.44541
6	12	3.0590	0.27809	0.50489
Total	100	26.955		

Sq. error divided by (No. patterns − No. of clusters) = 0.28675

Table of distances between cluster centers

CLUS	1	2	3	4	5	6
1	0.00	0.759	0.742	0.612	0.736	0.717
2	0.759	0.00	0.736	0.780	0.816	0.697
3	0.742	0.736	0.00	0.647	0.748	0.736
4	0.612	0.780	0.647	0.00	0.874	0.676
5	0.736	0.816	0.748	0.874	0.00	0.736
6	0.717	0.697	0.736	0.676	0.736	0.000

The clustering statistic: 0.29051

Figure 3.29 ***(continued)***

Other data sets that cannot be adequately clustered by square-error programs, such as CLUSTER and FORGY, are demonstrated by Zahn (1971) and Hall et al. (1973).

3.3.4 Clustering by Mixture Decomposition

A popular approach to clustering is based on the notion of a *mixture density*. Each pattern is assumed to be drawn from one of K underlying populations, or clusters. The clustering problem is to allocate each pattern to its correct population. Unlike the density estimation or the mode-seeking clustering algorithms discussed in Section 3.3.5, the form and the number of underlying population densities are assumed to be known here. The patterns are not labeled by population. If the parameters of the population densities can be estimated from the patterns, each pattern can be assigned to its appropriate cluster based on estimated probability densities. This model of clustering is identical to the problem of unsupervised learning in statistical pattern recognition (Appendix A) and has been used to estimate crop acreages from remote-sensing data (Odell and Basu, 1976).

The patterns are drawn from a population with a known number of clusters, or classes. The underlying probability density function for class ω_i is denoted $p(\mathbf{x}|\omega_i,\boldsymbol{\theta}_i)$, where $\boldsymbol{\theta}_i$ is a vector of unknown parameters for ω_i. If $P(\omega_i)$ is the a priori probability of class ω_i, or the chance that a pattern comes from ω_i, the mixture density can be written as

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{i=1}^{K} p(\mathbf{x}|\omega_i, \boldsymbol{\theta}_i)P(\omega_i)$$

where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$. The class-conditional densities $p(\mathbf{x}|\omega_j, \boldsymbol{\theta}_j)$ are called the *component densities*, and the a priori probabilities $P(\omega_j)$ are called the *mixing parameters*. Note that

$$\sum_{i=1}^{K} P(\omega_i) = 1$$

We would like to use the patterns to estimate the parameter vector $\boldsymbol{\theta}$ so that the mixture can be decomposed into its component clusters, assuming that the mixture is "identifiable" (Duda and Hart, 1973; Titterington et al., 1985). If one cluster is to be assigned to each category, the parameters of each component density can be estimated separately. This is estimation, not clustering.

The general formulation given above is applicable for arbitrary density functions. In the absence of prior knowledge about the shape and size of the clusters present in the data, it is a common practice to assume that the component densities are multivariate normal with different mean vectors and, perhaps, different covariance matrices (McLachlan, 1982; Symons, 1981; Sclove, 1977; Scott and Symons, 1971; Wolfe, 1970; Day, 1969). This approach to clustering is model based, but the model is Gaussian. The parameter vector $\boldsymbol{\theta}$ is usually estimated by the maximum

likelihood approach, although the Bayesian approach has also been used (Binder, 1978; Symons, 1981). Duda and Hart (1973) lucidly illustrate the practical difficulties associated with obtaining the maximum likelihood estimates of $\boldsymbol{\theta}$. One difficulty is that no explicit solution for the maximum likelihood estimator of $\boldsymbol{\theta}$ exists, so an iterative estimation scheme must be employed. Starting with an initial estimate of $\boldsymbol{\theta}$, a hill-climbing or gradient-descent procedure maximizes the log-likelihood function. Problems such as the rate of convergence, singular solutions, dependence on the starting point, and local versus global maximum are inherent in this procedure. These problems are further compounded as the number of unknown parameters increases. It is commonly assumed that the K covariance matrices are equal to limit the size of the problem.

It turns out that this maximum likelihood approach of mixture decomposition is related to a well-known clustering method (Symons, 1981). If the K covariance matrices are equal, then the maximum likelihood criterion is a simple modification of Friedman and Rubin's invariant criterion of minimizing $|\mathcal{S}_W|$. When the covariance matrices or cluster shapes are different, the maximum likelihood grouping minimizes

$$\prod_{i=1}^{K} |\mathcal{S}_W^{(i)}|^{n_i}$$

Symons (1981) has compared several of these criteria on real as well as synthetic data sets. His empirical results show that the choice of the most appropriate criterion depends on the similarity of the component covariance matrices and the relative sizes of the clusters. A suboptimal but practical approach is to take the clusters generated in a square-error clustering program, such as CLUSTER (Section 3.3.3), and use the cluster centers as estimates of mean vectors and sample covariance matrices from each cluster as estimates of the covariance matrix.

3.3.5 Clustering by Density Estimation and Mode Seeking

Clusters can be viewed as regions of the pattern space in which the patterns are dense, separated by regions of low pattern density. Clusters can be identified by searching for regions of high density, called modes, in the pattern space. Each mode is associated with a cluster center and each pattern is assigned to the cluster with the closest center. The probability density estimate at a point $\mathbf{x}$ is proportional to the number of patterns, k_n, falling in a small region of volume V_n around $\mathbf{x}$ (Duda and Hart, 1973; Silverman, 1986).

$$\hat{p}_n(\mathbf{x}) = \frac{k_n/n}{V_n}$$

where n is the total number of patterns. For a fixed V_n, k_n will be large for points lying in a dense region, resulting in a large estimate $\hat{p}_n(\mathbf{x})$. The choice of V_n is critical when n is small and is governed either by the Parzen window approach or by the nearest-neighbor approach (Duda and Hart, 1973).

The volume V_n in the Parzen window approach is specified as a function of

n. In the nearest-neighbor approach, k_n is specified as a function of n. The region around each pattern is examined to capture its k_n nearest neighbors. In both approaches, convergence arguments and other heuristics suggest that V_n be inversely proportional to $\sqrt{n}$. Once more, k_n is usually taken as proportional to $\sqrt{n}$. The primary difference between these two approaches is that the window around each point in the Parzen-window approach has the same volume, whereas the window size depends on the location of the pattern in the pattern space in the nearest-neighbor approach.

The simplest way to identify modes in the data is to construct a histogram by partitioning the pattern space into a number of nonoverlapping regions or cells. Cells with relatively high frequency counts are the potential modes or cluster centers and the boundaries between clusters fall in the "valleys" of the histogram. This method has the capability of identifying unimodal clusters of any shape. However, the number of patterns must be sufficiently large (compared to the number of features) in order for the histogram to be a good estimate of the density function. Even if the sample-size requirement is met, the success of such an approach depends on two factors. First, cells of small volume will give a very "noisy" estimate of the density, whereas large cells tend to overly smooth the density estimate. Second, the procedure for locating peaks and valleys in the histogram must be performed over a neighborhood whose size is known. These factors are difficult to handle in more than a few dimensions.

The general concept of identifying modes for clustering has been proposed by a number of researchers (Torn, 1977; Wong and Liu, 1977; Kittler, 1976; Koontz et al., 1976; Eigen et al., 1974; Katz and Rholf, 1973; Gitman and Levine, 1970; Mucciardi and Gose, 1972; Sebestyen and Edie, 1966). This approach has been quite popular in the clustering of multispectral data in remote sensing, where the large-sample-size requirement is easily met and the size of the histogram cells is naturally defined because of the gray-level quantization (Narendra and Goldberg, 1977; Goldberg and Shlien, 1978; Wharton, 1983). Wharton (1983) shows that the performance of clustering based on histograms is comparable to that of K-means clustering. The surprising result of this study was that even for moderately sized data sets (100 observations per category), the histograms gave a reasonable estimate of the density function.

The memory and run-time requirements of storing and searching multidimensional histograms can be enormous. Chhikara and Register (1979) get around this problem by constructing one histogram for each feature. Histograms with only one mode are eliminated and the set of patterns is dichotomized on the feature whose histogram has the smallest number of modes. This sequential splitting procedure is repeated by recomputing histograms for individual clusters. A merging procedure is invoked whenever the clusters overlap in the pattern space. The splitting and merging stop when all the clusters have unimodal frequency distributions for every feature. This algorithm requires the user to interactively specify the valleys in the histogram for data splitting. Eigen et al. (1974) also use equal-interval histograms for each dimension and identify modes by recording sign changes in the finite differences of the cell counts.

The mode separation procedure of Kittler (1976) uses a Parzen window estimate of the density function with a hypercubic "kernel function" (Duda and Hart, 1973). Unimodal regions of the pattern space are identified as follows. The starting pattern is chosen randomly and corresponds to the first point in the sequence. The second point in the sequence is that pattern which has a maximum density in a hypercubic window around the first pattern. The pattern with the maximum density in the region which is the union of the windows around the first two patterns is selected for the third point. A one-dimensional sequence of density estimates is thus obtained in which each pattern is represented once and only once. The regions of dense patterns correspond to the peaks in this plot. Shaffer et al. (1979) have analyzed this algorithm and demonstrate that the results of this mode-seeking algorithm are always identical to the results of single-link clustering. Thus, seemingly different clustering algorithms can give the same results.

The underlying density of patterns can also be estimated by the k_n-nearest-neighbor method (Wong and Lane, 1983). Two patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ are said to be neighbors if $\mathbf{x}_i$ is one of the k_n nearest neighbors of $\mathbf{x}_j$ and if $\mathbf{x}_j$ is among the k_n patterns closest to $\mathbf{x}_i$. The dissimilarity between neighboring patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ is given by

$$d(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2\hat{p}_n(\mathbf{x}_i)} + \frac{1}{2\hat{p}_n(\mathbf{x}_j)}$$

Pairs of patterns that are not neighbors are assigned arbitrarily large dissimilarities. A single-link clustering algorithm (Section 3.2.1) is then applied to this dissimilarity matrix to generate hierarchical clusters. The parameter k_n is a function of n and is usually taken to be $\log_2 n$. However, different values of k_n can lead to different clusterings. Wong and Lane (1983) demonstrate that their clustering algorithm is "strongly set consistent" for high-density clusters; that is, the resulting single-linkage clusters are maximally connected sets of the form

$$\{\mathbf{x} \mid \hat{p}_n(\mathbf{x}) \geq p^*\}$$

for some fixed density level p^*. Mode-seeking clustering methods have been used extensively in the engineering literature, particularly in remote sensing applications.

3.3.6 Clustering by Graph Theory

Various kinds of geometric structures or graphs for analyzing multidimensional patterns have led to some useful algorithms which can identify irregularly shaped or nonglobular clusters. Section 3.2 has exhibited a number of algorithms for hierarchical clustering based on graph theory. This section treats objects as points in a pattern space, so distances are available between all pairs of objects. The methods in this section seek single partitions, not hierarchies.

A graph is constructed whose nodes represent the patterns to be clustered and whose edges represent relations between the nodes (see Appendix G). In the simplest case, every node is connected to the remaining $(n - 1)$ nodes, resulting

in the complete graph. The edge weights are distances between pairs of patterns. For the purpose of clustering, it is the relative positions of the points that is important; pairs of patterns in the same cluster should be closer than pairs of patterns belonging to different clusters.

Several graph structures, such as minimum spanning trees, relative neighborhood graphs, and Gabriel graphs, have been imposed on the set of patterns to capture perceptual grouping. These graphs choose a subset of the $n(n-1)/2$ edges in the complete graph to reflect the "structure" or the inherent separation among clusters. The edges in these graphs mostly correspond to small interpoint distances. These graphs depend only on the ordering of the lengths of the edges. Clustering methods decompose the graphs into connected components by identifying and deleting "inconsistent" edges. Each component represents a cluster.

Zahn (1971) demonstrated how the minimum spanning tree (MST) can be used to detect clusters. His choice of MST was influenced by the Gestalt principle, which favors that grouping of patterns which represents smaller interpoint distances. The basic idea of Zahn's clustering algorithm is very simple and consists of the following steps.

ZAHN'S CLUSTERING ALGORITHM

Step 1. Construct the MST for the set of n patterns given.

Step 2. Identify inconsistent edges in the MST.

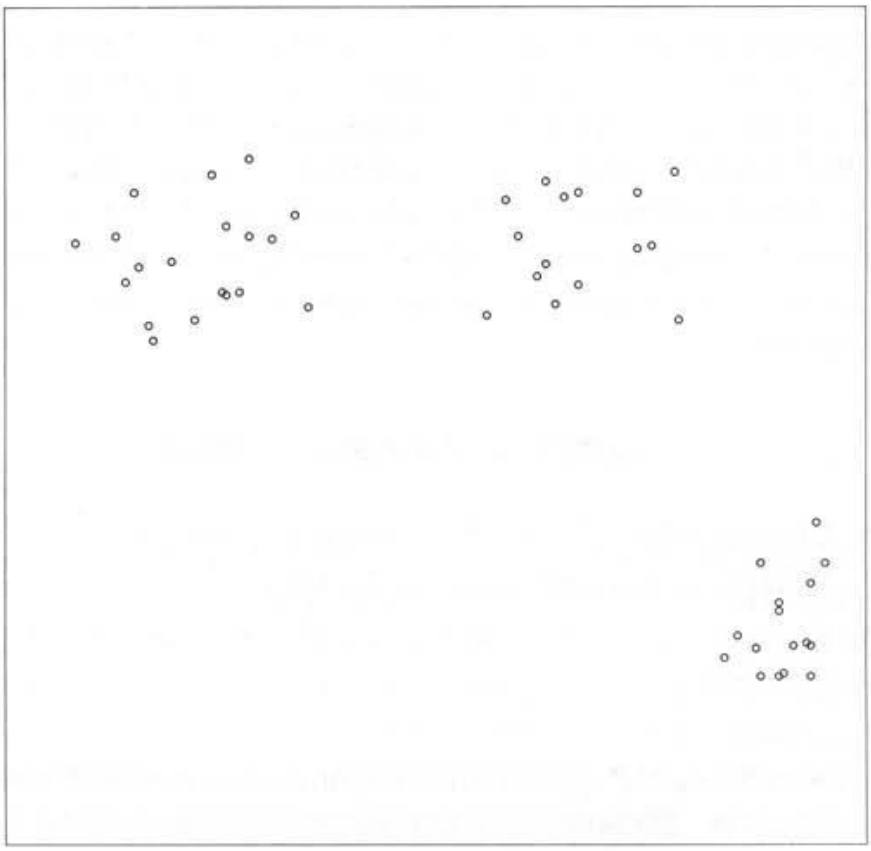
Step 3. Remove the inconsistent edges to form connected components and call them clusters.

Zahn's algorithm can be applied iteratively to each of the resulting components to identify subclusters. Section 3.2.4 explains the relation between single-link clustering and the MST. The crucial step in the algorithm is the definition of inconsistency. Zahn considers several criteria for inconsistency. In one, an edge is inconsistent if its weight (interpoint distance) is significantly larger than the average of nearby edge weights. Thus the inconsistent edges are related to cluster separation. The number of standard deviations by which an edge weight differs from the average of nearby edge weights and the ratio of the edge weight to the average of nearby edge weights are two means for identifying inconsistent edges. An edge with a factor of inconsistency of two usually links two clusters and can be deleted.

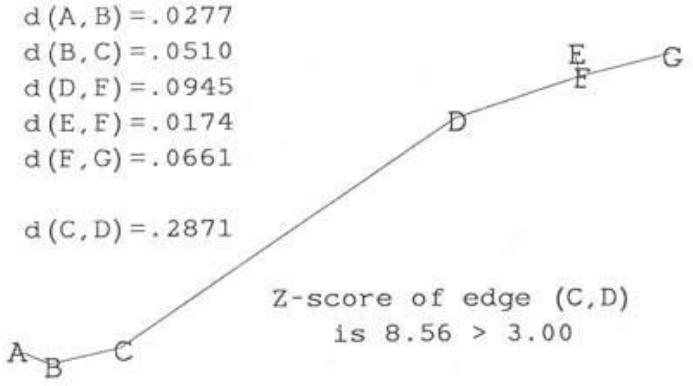
Figure 3.30 illustrates inconsistent edges for a two-dimensional data set. Figure 3.30(a) shows a set of two-dimensional patterns containing three well-separated clusters. Figure 3.30(b) demonstrates an inconsistent edge in the MST for a set of seven patterns. Since the z-score of edge (C, D) is greater than 3, it is unusually long compared to its neighboring edges, and hence is labeled as an inconsistent edge. Figure 3.30(c) identifies inconsistent edges in the MST for the data in Figure 3.30(a). The two intercluster edges have been correctly identified,

but an additional inconsistent edge breaks the cluster in the upper left corner into two components.

Zahn has applied his algorithm to a number of data sets consisting of clusters with different shapes and properties, including touching clusters, clusters with smoothly varying point densities, smoothly varying nonhomogeneous clusters, and line-like clusters. The results show that while the above-mentioned notion of inconsistent edges works well for disjoint clusters, special heuristics are needed for

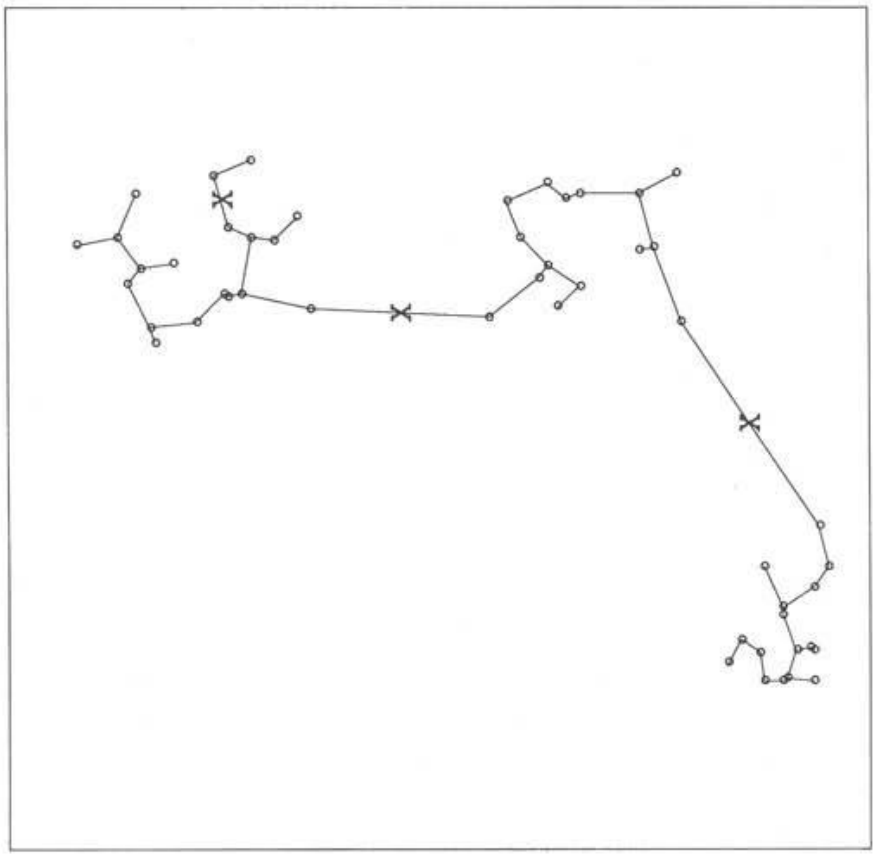


(a)



(b)

Figure 3.30 Inconsistent edges in two-dimensional data: (a) three well-separated clusters; (b) an example of an inconsistent edge; (c) MST of (a) with inconsistent edges marked X.



(c)

Figure 3.30 (*continued*)

more complex situations. For example, in the case of two fairly homogeneous clusters of different point density shown in Figure 3.31, several inconsistent edges will be found in the sparse cluster. Zahn suggests first detecting and deleting the denser cluster and then analyzing the remaining data. The histogram of the edge lengths in the MST helps us in identifying the intercluster edges; intercluster edges occupy the region between the two peaks (corresponding to within-cluster edge lengths) in the histogram of edge lengths.

Prior knowledge of the shapes of the clusters is needed to select the proper heuristic to identify inconsistent edges. This is the greatest deficiency of the MST-based approach in more than two dimensions. The very reason for applying cluster analysis is often to estimate the shapes of clusters and their structure. Nevertheless, MST-based clustering is an important technique which complements the square-error partitional technique. Some other variations of Zahn's idea have also been reported in the literature (Fehlauer and Eisenstein, 1978; Page, 1974; Magnuski, 1975). Koontz et al. (1976) and Mizoguchi and Shimura (1980) base their clustering algorithms on directed trees.

Two other geometric structures, the relative neighborhood graph (RNG) and the Gabriel graph (GG), have also been used in cluster analysis (Urquhart, 1982; Matula and Sokal, 1980). These connected graphs are based on a region of influence

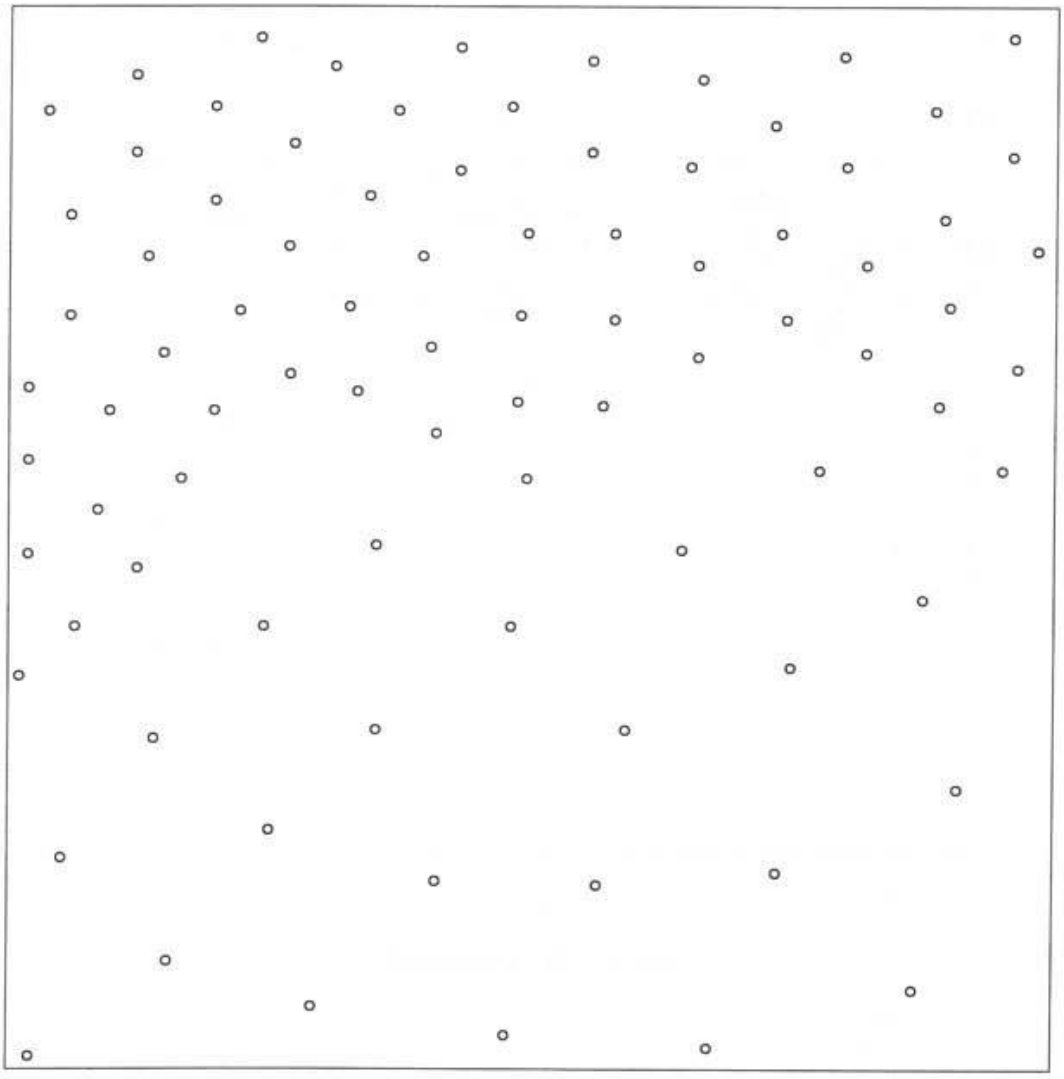


Figure 3.31 Two homogeneous clusters with different density.

(Toussaint, 1980). Patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ are defined to be relative neighbors, and are connected in the RNG, if and only if

$$d(\mathbf{x}_i, \mathbf{x}_j) \leq \max \{d(\mathbf{x}_i, \mathbf{x}_k), d(\mathbf{x}_j, \mathbf{x}_k)\} \qquad \text{for all } k, \quad k \neq i \text{ and } k \neq j$$

where $d(\mathbf{x}_i, \mathbf{x}_j)$ denotes the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. Or, we can say that $\mathbf{x}_i$ and $\mathbf{x}_j$ are connected in RNG if and only if no other point falls in LUNE($\mathbf{x}_i$, $\mathbf{x}_j$), where LUNE($\mathbf{x}_i$, $\mathbf{x}_j$) is the intersection of the two disks of radius $d(\mathbf{x}_i, \mathbf{x}_j)$ centered at $\mathbf{x}_i$ and $\mathbf{x}_j$. LUNE is the region of influence of RNG and is shown in Figure 3.32(a).

The Gabriel graph (GG) is defined as follows. Points $\mathbf{x}_i$ and $\mathbf{x}_j$ are connected in GG if and only if

$$d^2(\mathbf{x}_i, \mathbf{x}_j) < d^2(\mathbf{x}_i, \mathbf{x}_k) + d^2(\mathbf{x}_j, \mathbf{x}_k) \qquad \text{for all } k, \quad k \neq i \text{ and } k \neq j$$

This is equivalent to the condition that two points $\mathbf{x}_i$ and $\mathbf{x}_j$ are connected in GG if and only if no other point lies in DISK($\mathbf{x}_i$, $\mathbf{x}_j$), where DISK($\mathbf{x}_i$, $\mathbf{x}_j$) is the disk with diameter $d(\mathbf{x}_i, \mathbf{x}_j)$ as shown in Figure 3.32(b). We say that DISK is the

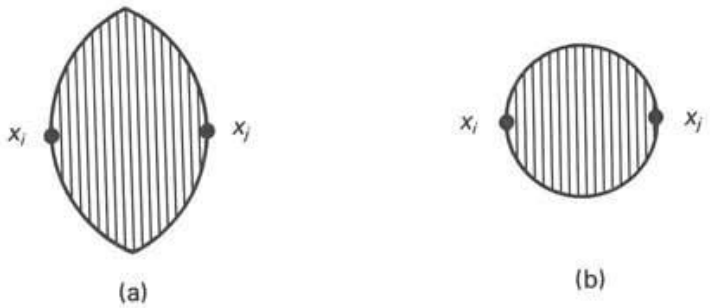


Figure 3.32 Regions of influence for RNG and GG.

region of influence for this graph. The construction of RNG and GG is well understood for two-dimensional data, but the construction of these graphs in high dimensions is a difficult problem.

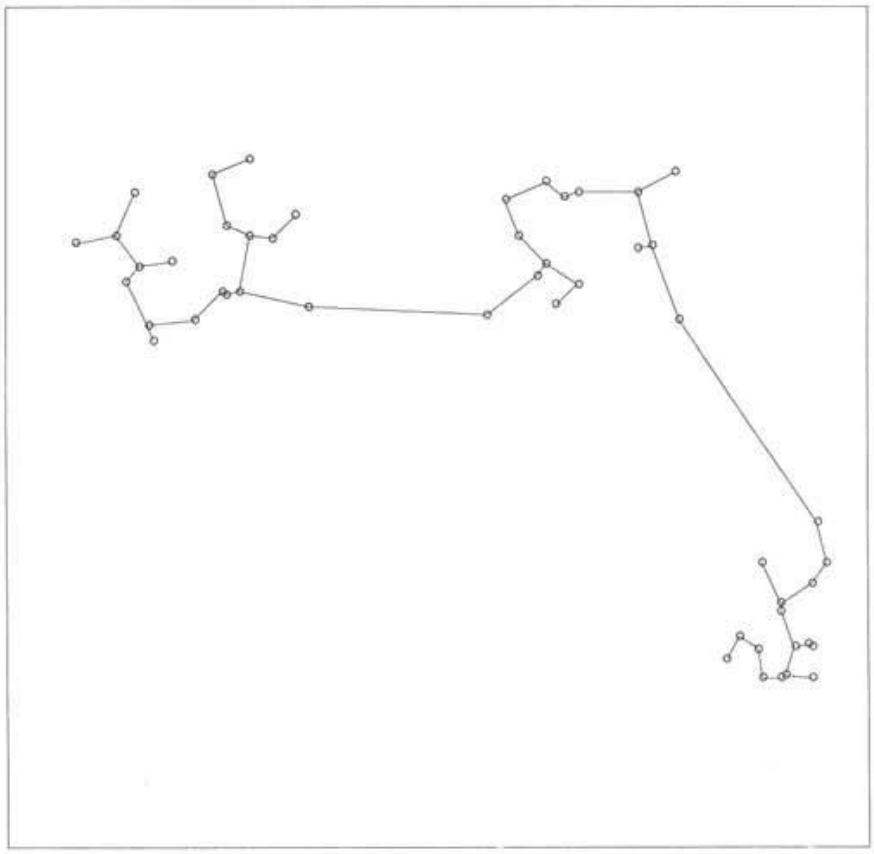
Delaunay triangulation (DT) is another graph structure that is useful in point pattern analysis and plays a prominent role in many algorithms that compute GG and RNG; efficient algorithms for computing DT are available and RNG and GG can easily be obtained from DT by deleting some of the edges in DT. The definition of DT is best made in terms of its dual structure, the Dirichlet tessellation. The Dirichlet tessellation, also well known as the Voronoi diagram, of a set of patterns $\mathcal{X}$ in R^d (d-dimensional Euclidean space) is a partition of R^d into "cells" about each pattern vector $\mathbf{x}_i$ such that each cell consists of those points of R^d lying closer to $\mathbf{x}_i$ than to any other pattern in $\mathcal{X}$. Cell boundaries are intersections of the perpendicular bisectors of the lines connecting $\mathbf{x}_i$ to each of the $(n - 1)$ other patterns in $\mathcal{X}$. Thus each cell is a convex polygon.

The Delaunay triangulation is defined as follows. The edge connecting points $\mathbf{x}_i$ and $\mathbf{x}_j$ is in the DT if and only if the two cells of the Dirichlet tessellation containing $\mathbf{x}_i$ and $\mathbf{x}_j$ share a common boundary. A large body of applications of DT resides in such varied disciplines as biology and geography. For example, DT has been used as a model of territories of breeding bird species (Sibson, 1980). Ahuja (1982) outlines applications of DT to problems in clustering, matching, and segmentation. (See Chapter 5 for segmentation and matching in the image processing context.) An implementation of an agglomerative clustering algorithm based on DT has been made by Howe (1978) and applied to observations of pollen in lake sediments with the goal of partitioning the region with respect to forest type. However, these applications have been developed only for two-dimensional data.

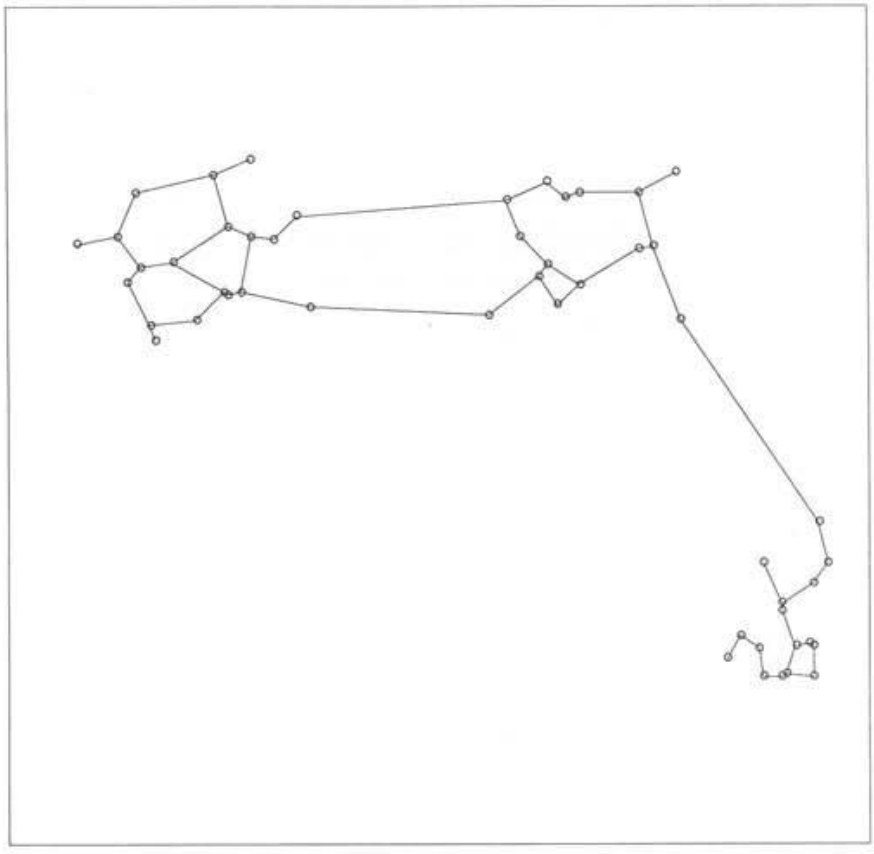
A minimum spanning tree and a Delaunay triangulation (DT) play important roles as "bounds" on RNG and GG. It can be shown that

$$E(\text{MST}) \subseteq E(\text{RNG}) \subseteq E(\text{GG}) \subseteq E(\text{DT})$$

where E denotes the edge set of a graph. The first set inclusion guarantees that the RNG is a supergraph of its MST. Thus every RNG is connected. Lee and Preparata (1984) define a method for efficiently computing the RNG or GG. Figure 3.33 demonstrates the MST, RNG, GG, and Delaunay triangulation for the data in Figure 3.30(a).

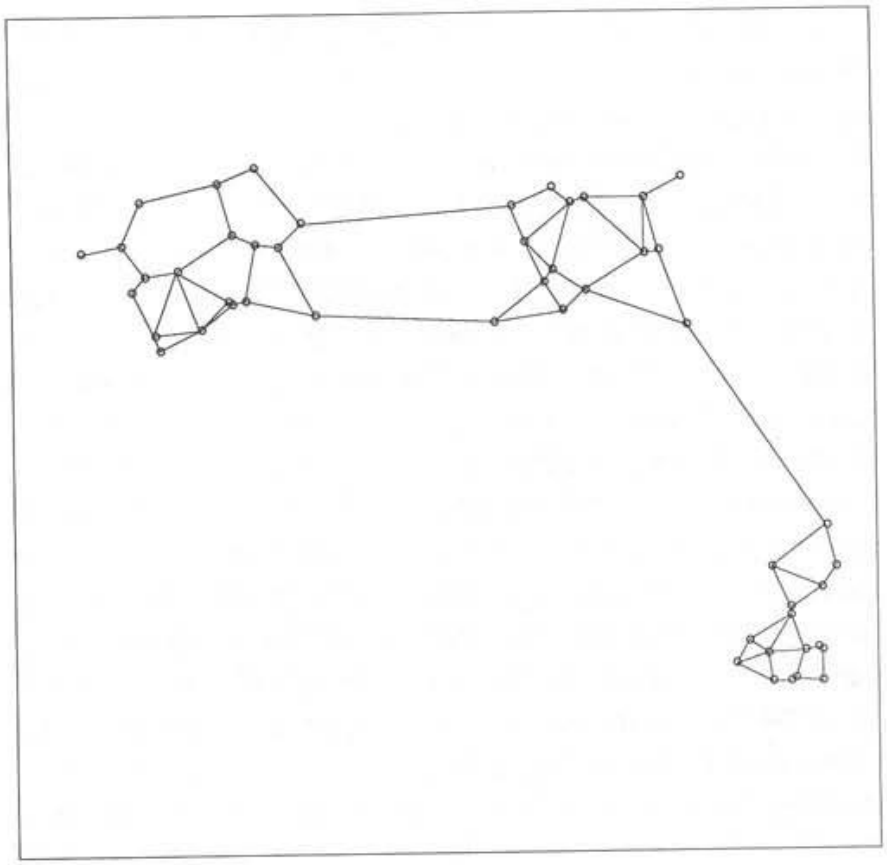


(a)

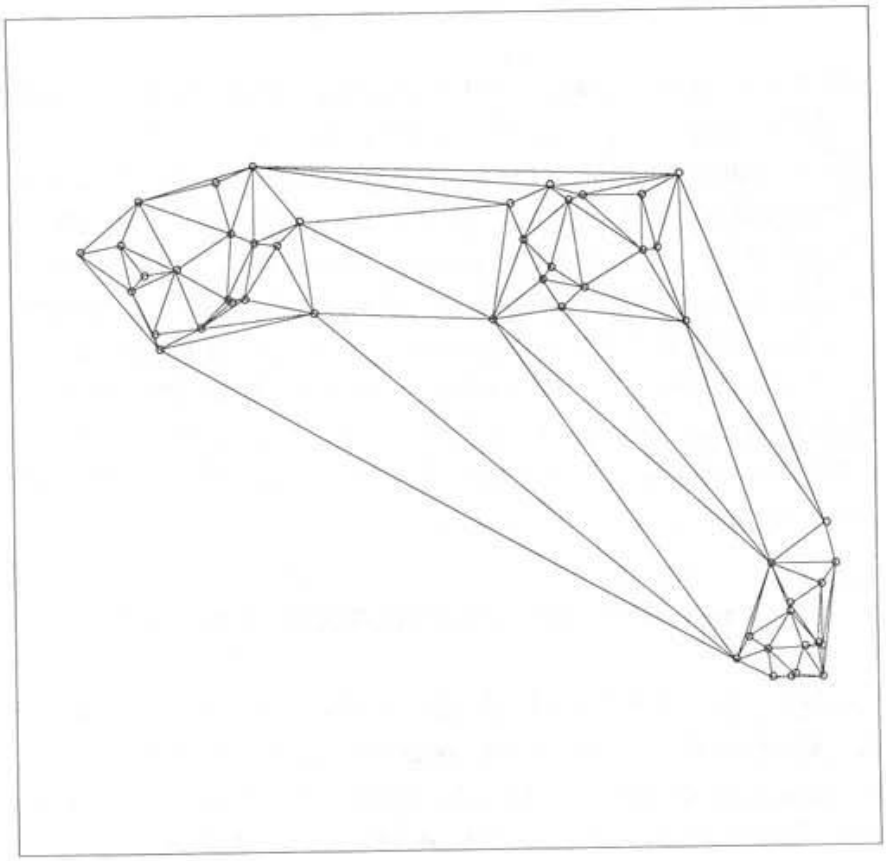


(b)

Figure 3.33 MST, RNG, GG, and Delaunay triangulation for the data in Figure 3.30(a): (a) MST; (b) RNG; (c) GG; (d) Delaunay triangulation.



(c)



(d)

Figure 3.33 (*continued*)

Clustering algorithms based on RNG, GG, and DT are very similar to Zahn's MST-based clustering algorithm. Only the first step in his algorithm needs to be changed: from "Construct the MST" to "Construct the appropriate graph structure." The heuristics needed to define inconsistent edges becomes more complex with larger sets of edges. In the case of the MST, removing a single edge always results in two components. This is not always true with the RNG, GG, and DT because there can be more than one path between two patterns. Applications of the RNG, GG, and DT have been limited so far to two- and three-dimensional patterns primarily due to computational difficulties in higher dimensions.

Urquhart (1982) favors the use of GG and RNG over the MST for clustering problems for the following reasons. First, RNG and GG are less sensitive to changes in the positions of patterns than the MST. Second, since the GG and RNG are supergraphs of the MST, they exhibit a greater degree of interconnectedness of patterns and so may be more appropriate in capturing the cluster structure in the data than the MST. Indeed, the RNG-based clustering algorithms developed by Urquhart avoid some of the problems of Zahn's MST algorithm. Ahuja (1982) and Tuceryan (1986) argue for the intuitively appealing characteristics of DT over other graph structures in processing patterns. They have demonstrated that DT is useful for grouping or clustering two-dimensional patterns. However, we again emphasize that the performance of a clustering algorithm is data dependent.

3.3.7 Nearest-Neighbor Clustering

A natural way to define clusters is by utilizing the property of nearest neighbors; a pattern should usually be put in the same cluster as its nearest neighbor. Two patterns should be considered similar if they share neighbors. The notion of nearest neighbors is inherent in the construction of various graphs, particularly the graphs discussed in Section 3.3.6, so graph-theoretic clustering methods are closely related to nearest-neighbor clustering methods. However, they differ significantly in how the clusters are formed and, most important, in the final partition.

A very simple clustering algorithm which is based on the nearest neighbor rule is given below (Lu and Fu, 1978). A set of patterns $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ is to be partitioned into K clusters. The user specifies a threshold, t, on the nearest-neighbor distance.

NEAREST-NEIGHBOR CLUSTERING ALGORITHM

Step 1. Set $i \leftarrow 1$ and $k \leftarrow 1$. Assign pattern $\mathbf{x}_1$ to cluster C_1.

Step 2. Set $i \leftarrow i + 1$. Find the nearest neighbor of $\mathbf{x}_i$ among the patterns already assigned to clusters. Let d_m denote the distance from $\mathbf{x}_i$ to its nearest neighbor. Suppose that the nearest neighbor is in cluster m.

Step 3. If $d_m \leq t$, then assign $\mathbf{x}_i$ to C_m. Otherwise, set $k \leftarrow k + 1$ and assign $\mathbf{x}_i$ to a new cluster C_k.

Step 4. If every pattern has been assigned to a cluster, stop. Else, go to step 2.

The number of clusters generated, K, is a function of the parameter t. As the value of t increases, fewer clusters are generated. The nearest neighbor distance in step 2 can be replaced by the average distance between $\mathbf{x}_i$ and its p nearest neighbors in the mth cluster. Then the user has to specify another parameter, namely, p. Lu and Fu (1978) have used this clustering algorithm to cluster patterns represented by sentences or strings in an application of syntactic pattern recognition to character recognition.

Jarvis and Patrick (1973) defined a proximity measure as the number of matches in near-neighbor lists for two patterns. Their clustering algorithm can be summarized as follows: Place patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ into the same cluster if $\mathbf{x}_i$ and $\mathbf{x}_j$ share at least k_t near neighbors and $\mathbf{x}_i$ and $\mathbf{x}_j$ are k-near neighbors of each other. This algorithm is noniterative and is computationally attractive since near neighbors can be computed efficiently (Kamgar-Parsi and Kanal, 1985). However, the user has to specify the size of the neighborhood, k, and the similarity threshold, k_t. Jarvis and Patrick (1973) do not provide any guidelines for choosing these parameters but suggest finding the "best" value interactively. Note that large values of k bias the algorithm toward globular structures, whereas small values of k favor chained or elongated structures (Jarvis, 1978). A hierarchy can also be generated by varying the value of k_t. Jarvis and Patrick claim that since Zahn's MST method is based on the first near neighbor, it is a first-order method and seeks linear and elongated clusters at the expense of globular structures.

The notion of proximity based on shared nearest neighbors has been modified by Gowda and Krishna (1978) to measure the "mutual nearness" of two patterns. If $\mathbf{x}_j$ is the pth near neighbor of $\mathbf{x}_i$ and $\mathbf{x}_i$ is the qth near neighbor of $\mathbf{x}_j$, then the *mutual neighborhood value* (MNV) between $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined as $(p + q)$. The smaller MNV, the more similar the patterns. This represents a stronger notion of similarity than the number of shared neighbors of Jarvis and Patrick. Gowda and Krishna's clustering algorithm is described below.

MUTUAL NEIGHBORHOOD CLUSTERING ALGORITHM

Step 1. Determine the k near neighbors of every pattern.

Step 2. Compute the MNV for every pair of patterns. If patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ are not mutual neighbors for a given value of k, set MNV $(\mathbf{x}_j, \mathbf{x}_j)$ to an arbitrarily large number.

Step 3. Identify all the pairs of patterns with MNV of 2. Merge each such pair into a cluster, starting with the pair having the smallest distance.

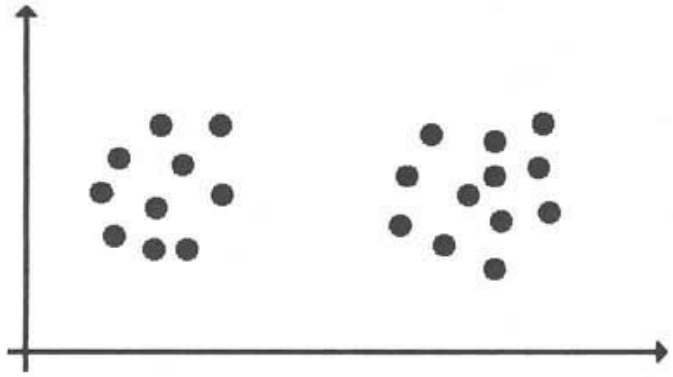
Repeat step 3 for MNV thresholds of 3, 4, . . . , $2k$ to generate a hierarchy.

The parameter k that controls the neighborhood depth is crucial to the performance of the algorithm. Small values of k give several "strong" clusters and

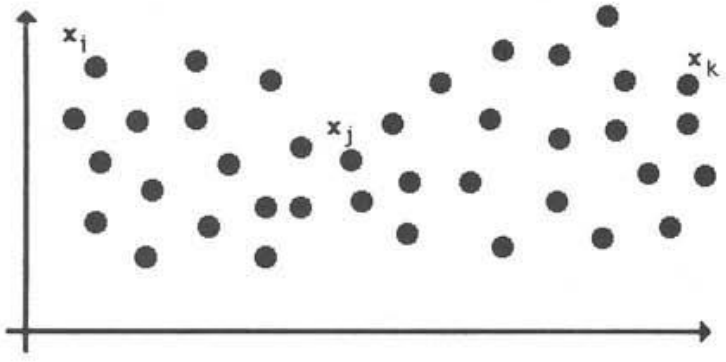
large values of k give fewer "weak" clusters. In fact, k can always be chosen sufficiently large to make the algorithm return a single cluster. Gowda and Krishna (1978) demonstrate that the algorithm is able to identify nonspherical clusters, linearly nonseparable clusters, clusters with unequal populations, and clusters with low-density bridges when k is 5 in two dimensions. However, no heuristic is provided to select an appropriate value of k for arbitrary data sets.

3.3.8 Fuzzy Clustering

The clustering algorithms described so far assign each pattern to one and only one cluster. In other words, the patterns are partitioned into disjoint sets; patterns in one cluster are supposed to be more similar to each other than to patterns in different clusters. If the clusters are compact and well separated, as demonstrated in Figure 3.34(a), there is no ambiguity or uncertainty associated with assigning each pattern to one cluster. We can easily see that there are two clusters in Figure 3.34(a) with well-defined boundaries. But what happens if the



(a)



(b)

Figure 3.34 Examples of cluster structures: (a) well-separated clusters; (b) touching or overlapping clusters.

clusters are touching or overlapping? Figure 3.34(b) illustrates a case in which cluster boundaries are not sharp and the assignment of patterns to clusters is difficult. Although it is clear that patterns $\mathbf{x}_i$ and $\mathbf{x}_k$ should be put in different clusters, pattern $\mathbf{x}_j$ could be put into either the cluster containing $\mathbf{x}_i$ or the cluster containing $\mathbf{x}_k$. Such clusters are said to have "fuzzy" boundaries.

The fuzzy set theory developed by Zadeh (1965) permits an object to belong to a cluster with a grade of membership. The degree of membership takes a value in the interval [0, 1]. For ordinary clusters, called "crisp" clusters, the membership grade for pattern $\mathbf{x}_i$ in a particular cluster is 1 if the pattern belongs to the cluster and 0 if it does not. With fuzzy clusters, pattern $\mathbf{x}_i$ has a grade of membership, $f_q(\mathbf{x}_i) \geq 0$, or degree of belonging to the qth cluster, where $\sum_q f_q(\mathbf{x}_i) = 1$. The larger $f_q(\mathbf{x}_i)$, the more confidence exists that $\mathbf{x}_i$ belongs to cluster q. If $f_j(\mathbf{x}_i)$ is 1, pattern $\mathbf{x}_i$ belongs to cluster j with absolute certainty. The interpretations of values such as 0.3 is less clear. Membership grades are subjective in nature and are based on definitions rather than measurements (Zadeh, 1984). For example, pattern $\mathbf{x}_j$ in Figure 3.34(b) could belong to one cluster with membership value 0.45 and to the other cluster with membership value 0.55.

The grade of membership is not the same as the probability that the pattern belongs to the cluster even though grades of membership and probabilities both take values in the range [0, 1]. Under a probabilistic framework, pattern $\mathbf{x}_j$ belongs to one and only one cluster, depending on the outcome of a random experiment. In fuzzy set theory, pattern $\mathbf{x}_j$ can belong to two clusters simultaneously. The membership grades determine the degree to which two cluster labels are applicable.

Another interpretation of the degree of membership is that it measures the compatibility of a pattern or an object with the description of a fuzzy set. Sometimes this property is useful in interpreting the results of a clustering algorithm. Consider the problem of clustering a variety of computers ranging from microcomputers to mainframes based on such attributes as memory size, CPU speed, and processor type. The objective might be to partition computers into two clusters labeled "personal" and "multiuser system." To which cluster should a computer based on a Motorola 68020 processor be assigned, which is used in a variety of personal computers and workstations?

Proponents argue that fuzzy clustering is more appropriate than ordinary clustering for capturing human concepts such as "small," "big," "high," and "low." Fuzzy sets are likely to find increasing use in applications involving imprecise and incomplete information, commonsense reasoning, and complex concepts (Zadeh, 1984). Skeptics of fuzzy clustering do not doubt its mathematical correctness, but are not convinced that it offers any advantages over the classical and better understood clustering methods. Many more papers have been written on the theoretical foundations of fuzzy sets and fuzzy logic than on its practical applications.

Clustering has always been a popular domain for fuzzy sets. Early work by Bellman et al. (1966), Ruspini (1969), Gitman and Levine (1970), Bezdek (1974), and Dunn (1974) have culminated in two books on fuzzy clustering (Bezdek,

1981; Backer, 1978). Virtually all the clustering algorithms based on fuzzy set theory are partitional in nature, but a few generate hierarchies. Some of these fuzzy algorithms are straightforward modifications of the square-error type of partitional algorithms discussed in Section 3.3.1. Indeed, Bezdek (1976) proposed a fuzzy ISODATA clustering algorithm for which convergence theorems are available.

The crucial step in a fuzzy clustering algorithm is the definition of the membership function. Backer (1978) shows how to construct a membership function based on similarity decomposition. Let the set of patterns $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be initially partitioned into clusters $\{C_1, \ldots, C_K\}$ and let n_i be the number of patterns in C_i. Let $\delta(\mathbf{x}, C_i)$ denote the similarity between pattern $\mathbf{x}$ and cluster C_i. The larger this value, the closer are the pattern and the cluster. The cluster membership function $f_{C_i}(\mathbf{x})$ for pattern $\mathbf{x}$ induced by cluster C_i is given by

$$f_{C_i}(\mathbf{x}) = P_i\delta(\mathbf{x}, C_i) \Big/ \sum_{k=1}^{K} P_k\delta(\mathbf{x}, C_k)$$

where $P_k = n_k/n$ is the relative size of cluster C_k. This membership function is nonnegative and sums to 1 for every pattern.

$$f_{C_k}(\mathbf{x}) \geq 0 \qquad \text{and} \qquad \sum_{k=1}^{K} f_{C_k}(\mathbf{x}) = 1$$

The similarity or affinity function, $\delta(\mathbf{x}, C_k)$, can be based on the distance concept, the neighborhood concept, or the probabilistic concept (Backer, 1978). It measures the relationship between a pattern and a cluster as a whole or between a pattern and one or more representatives of that cluster. The choice of this function depends on the data. Backer and Jain (1981) define an affinity function based on the mean vectors of the clusters. The membership function is

$$f_{C_k}(\mathbf{x}) = \frac{1 - (1/\beta)d(\mathbf{x}, \mathbf{m}^{(k)})}{K - (1/\beta)\sum_j d(\mathbf{x}, \mathbf{m}^{(j)})}$$

where $d(\mathbf{x}, \mathbf{m}^{(k)})$ denotes the Euclidean distance between the pattern vector $\mathbf{x}$ and the centroid $\mathbf{m}^{(k)}$ of cluster C_k. The parameter β controls the neighborhood size and affects the values of cluster belongingness. Only general guidelines are available for choosing β. The performance of a fuzzy clustering algorithm depends critically on the definition of the membership function.

Fuzzy partitional clustering algorithms generate partitions that minimize induced fuzziness following the same steps as square-error clustering algorithms. The induced fuzziness takes its minimum value if we can obtain a partition for which $f_{C_k}(\mathbf{x}) \in \{0, 1\}$ or, equivalently, when the partition is nonfuzzy. Therefore, a criterion function needs to be defined to characterize the induced fuzziness of a partition. Backer (1978) defines a number of fuzzy partitioning criterion functions. For example, a criterion based on the average pairwise fuzzy set separability is given by

$$\Phi_f = 1 - \frac{2}{K-1} \sum_{k=1}^{K-1} \sum_{j=k+1}^{K} I(f_{C_k} \cap f_{C_j})$$

where K is the number of clusters and $I(f_{C_k} \cap f_{C_j})$ is the intersection of two fuzzy sets (clusters) defined as follows:

$$I(f_{C_k} \cap f_{C_j}) = \frac{1}{n} \sum_{\mathbf{x} \in \mathcal{X}} \min [f_{C_k}(\mathbf{x}), f_{C_j}(\mathbf{x})]$$

The minimum value of Φ_f is 0, which represents maximum fuzziness, and the maximum value of Φ_f is 1, which corresponds to a nonfuzzy partition. The problem of fuzzy clustering is to find that partition which maximizes Φ_f. The basic steps in a fuzzy partitional clustering algorithm are given below.

FUZZY PARTITIONAL CLUSTERING ALGORITHM

Step 1. Select an initial partition $\{C_k\}_{k=1}^{K}$.

Repeat steps 2 to 4 until the cluster memberships stabilize.

Step 2. Compute the membership functions $\{f_{C_k}(\mathbf{x}_j)\}$.

Step 3. Compute the criterion function Φ_f.

Step 4. Reclassify patterns to improve Φ_f.

The output of a fuzzy algorithm not only includes a partition but also additional information in the form of membership values. However, the new information provided by the membership values must be interpreted by the data analyst. In summary, fuzzy clustering is an interesting concept that includes most partitional clustering algorithms as special cases. However, its superiority to ordinary clustering has yet to be demonstrated in applications.

3.4 CLUSTERING SOFTWARE

Someone interested in applying clustering techniques has plenty of software to choose from. As we reported earlier, there is no shortage of clustering algorithms, and most of them have been implemented to run on a variety of computers. One of the main reasons for the large number of "different" clustering programs available is that most researchers and research groups put all their trust in the algorithm they have developed even though it is very similar to existing algorithms. In addition, documented and tested clustering packages were not available until recently, so users had to write their own software. The paper by Johnston et al. (1979) illustrates how the "law of serendipity" can lead to a "new" clustering algorithm. During the coding of the mode-seeking algorithm by Koontz et al. (1976), Johnston et al., by mistake, modified one of the expression for the relative density of a pair of patterns. This led to a new clustering algorithm which provided better clustering for "uniform, touching" clusters than the original!