CHAPTER 5

Clustering

5.1 INTRODUCTION

Clustering is similar to classification in that data are grouped. However, unlike classification, the groups are not predefined. Instead, the grouping is accomplished by finding similarities between data according to characteristics found in the actual data. The groups are called *clusters*. Some authors view clustering as a special type of classification. In this text, however, we follow a more conventional view in that the two are different. Many definitions for clusters have been proposed:

- Set of like elements. Elements from different clusters are not alike.

- The distance between points in a cluster is less than the distance between a point in the cluster and any point outside it.

A term similar to clustering is *database segmentation*, where like tuples (records) in a database are grouped together. This is done to partition or segment the database into components that then give the user a more general view of the data. In this text, we do not differentiate between segmentation and clustering. A simple example of clustering is found in Example 5.1. This example illustrates the fact that determining how to do the clustering is not straightforward.

EXAMPLE 5.1

An international online catalog company wishes to group its customers based on common features. Company management does not have any predefined labels for these groups. Based on the outcome of the grouping, they will target marketing and advertising campaigns to the different groups. The information they have about the customers includes

TABLE 5.1: Sample Data for Example 5.1

Income	Age	Children	Marital Status	Education
$25,000	35	3	Single	High school
$15,000	25	1	Married	High school
$20,000	40	0	Single	High school
$30,000	20	0	Divorced	High school
$20,000	25	3	Divorced	College
$70,000	60	0	Married	College
$90,000	30	0	Married	Graduate school
$200,000	45	5	Married	Graduate school
$100,000	50	2	Divorced	College

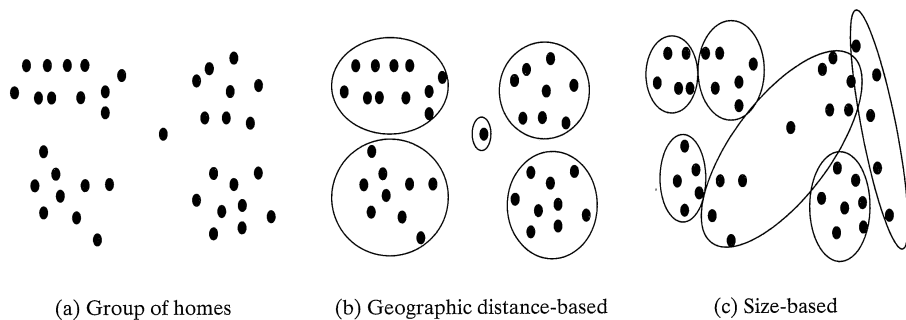(a) Group of homes   (b) Geographic distance-based   (c) Size-based

FIGURE 5.1: Different clustering attributes.

income, age, number of children, marital status, and education. Table 5 shows some tuples from this database for customers in the United States. Depending on the type of advertising, not all attributes are important. For example, suppose the advertising is for a special sale on children's clothes. We could target the advertising only to the persons with children. One possible clustering is that shown by the divisions of the table. The first group of people have young children and a high school degree, while the second group is similar but have no children. The third group has both children and a college degree. The last two groups have higher incomes and at least a college degree. The very last group has children. Different clusterings would have been found by examining age or marital status.

As illustrated in Figure 5.1, a given set of data may be clustered on different attributes. Here a group of homes in a geographic area is shown. The first type of clustering is based on the location of the home. Homes that are geographically close to each other are clustered together. In the second clustering, homes are grouped based on the size of the house.

Clustering has been used in many application domains, including biology, medicine, anthropology, marketing, and economics. Clustering applications include plant and animal

classification, disease classification, image processing, pattern recognition, and document retrieval. One of the first domains in which clustering was used was biological taxonomy. Recent uses include examining Web log data to detect usage patterns.

When clustering is applied to a real-world database, many interesting problems occur:

- Outlier handling is difficult. Here the elements do not naturally fall into any cluster. They can be viewed as solitary clusters. However, if a clustering algorithm attempts to find larger clusters, these outliers will be forced to be placed in some cluster. This process may result in the creation of poor clusters by combining two existing clusters and leaving the outlier in its own cluster.

- Dynamic data in the database implies that cluster membership may change over time.

- Interpreting the semantic meaning of each cluster may be difficult. With classification, the labeling of the classes is known ahead of time. However, with clustering, this may not be the case. Thus, when the clustering process finishes creating a set of clusters, the exact meaning of each cluster may not be obvious. Here is where a domain expert is needed to assign a label or interpretation for each cluster.

- There is no one correct answer to a clustering problem. In fact, many answers may be found. The exact number of clusters required is not easy to determine. Again, a domain expert may be required. For example, suppose we have a set of data about plants that have been collected during a field trip. Without any prior knowledge of plant classification, if we attempt to divide this set of data into similar groupings, it would not be clear how many groups should be created.

- Another related issue is what data should be used for clustering. Unlike learning during a classification process, where there is some a priori knowledge concerning what the attributes of each classification should be, in clustering we have no supervised learning to aid the process. Indeed, clustering can be viewed as similar to unsupervised learning.

We can then summarize some basic features of clustering (as opposed to classification):

- The (best) number of clusters is not known.

- There may not be any a priori knowledge concerning the clusters.

- Cluster results are dynamic.

The clustering problem is stated as shown in Definition 5.1. Here we assume that the number of clusters to be created is an input value, k. The actual content (and interpretation) of each cluster, K_j, $1 \leq j \leq k$, is determined as a result of the function definition. Without loss of generality, we will view that the result of solving a clustering problem is that a set of clusters is created: $K = \{K_1, K_2, \ldots, K_k\}$.

DEFINITION 5.1. Given a database $D = \{t_1, t_2, \ldots, t_n\}$ of tuples and an integer value k, the **clustering problem** is to define a mapping $f : D \rightarrow \{1, \ldots, k\}$ where each t_i is assigned to one cluster K_j, $1 \leq j \leq k$. A **cluster**, K_j, contains precisely those tuples mapped to it; that is, $K_j = \{t_i \mid f(t_i) = K_j, 1 \leq i \leq n, \text{ and } t_i \in D\}$.

A classification of the different types of clustering algorithms is shown in Figure 5.2. Clustering algorithms themselves may be viewed as hierarchical or partitional. With
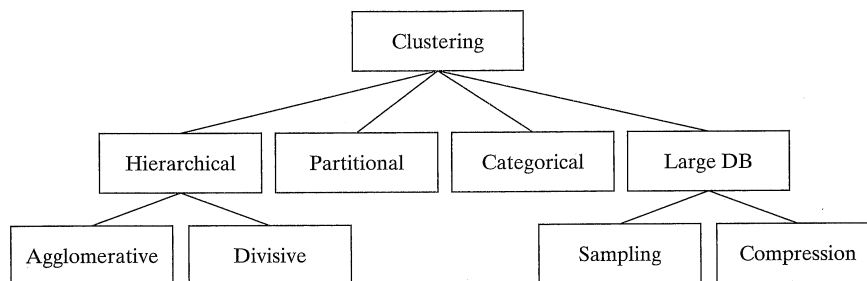
FIGURE 5.2: Classification of clustering algorithms.

hierarchical clustering, a nested set of clusters is created. Each level in the hierarchy has a separate set of clusters. At the lowest level, each item is in its own unique cluster. At the highest level, all items belong to the same cluster. With hierarchical clustering, the desired number of clusters is not input. With *partitional* clustering, the algorithm creates only one set of clusters. These approaches use the desired number of clusters to drive how the final set is created. Traditional clustering algorithms tend to be targeted to small numeric databases that fit into memory. There are, however, more recent clustering algorithms that look at categorical data and are targeted to larger, perhaps dynamic, databases. Algorithms targeted to larger databases may adapt to memory constraints by either sampling the database or using data structures, which can be compressed or pruned to fit into memory regardless of the size of the database. Clustering algorithms may also differ based on whether they produce overlapping or nonoverlapping clusters. Even though we consider only nonoverlapping clusters, it is possible to place an item in multiple clusters. In turn, nonoverlapping clusters can be viewed as extrinsic or intrinsic. *Extrinsic* techniques use labeling of the items to assist in the classification process. These algorithms are the traditional classification supervised learning algorithms in which a special input training set is used. *Intrinsic* algorithms do not use any a priori category labels, but depend only on the adjacency matrix containing the distance between objects. All algorithms we examine in this chapter fall into the intrinsic class.

The types of clustering algorithms can be furthered classified based on the implementation technique used. Hierarchical algorithms can be categorized as agglomerative or divisive. "*Agglomerative*" implies that the clusters are created in a bottom-up fashion, while *divisive* algorithms work in a top-down fashion. Although both hierarchical and partitional algorithms could be described using the agglomerative vs. divisive label, it typically is more associated with hierarchical algorithms. Another descriptive tag indicates whether each individual element is handled one by one, *serial* (sometimes called *incremental*), or whether all items are examined together, *simultaneous*. If a specific tuple is viewed as having attribute values for all attributes in the schema, then clustering algorithms could differ as to how the attribute values are examined. As is usually done with decision tree classification techniques, some algorithms examine attribute values one at a time, *monothetic*. *Polythetic* algorithms consider all attribute values at one time. Finally, clustering algorithms can be labeled based on the mathematical formulation given to the algorithm: graph theoretic or matrix algebra. In this chapter we generally use the graph approach and describe the input to the clustering algorithm as an adjacency matrix labeled with distance measures.

We discuss many clustering algorithms in the following sections. This is only a representative subset of the many algorithms that have been proposed in the literature. Before looking at these algorithms, we first examine possible similarity measures and examine the impact of outliers.

5.2 SIMILARITY AND DISTANCE MEASURES

There are many desirable properties for the clusters created by a solution to a specific clustering problem. The most important one is that a tuple within one cluster is more like tuples within that cluster than it is similar to tuples outside it. As with classification, then, we assume the definition of a similarity measure, $sim(t_i, t_l)$, defined between any two tuples, $t_i, t_l \in D$. This provides a more strict and alternative clustering definition, as found in Definition 5.2. Unless otherwise stated, we use the first definition rather than the second. Keep in mind that the similarity relationship stated within the second definition is a desirable, although not always obtainable, property.

> **DEFINITION 5.2.** Given a database $D = \{t_1, t_2, \ldots, t_n\}$ of tuples, a similarity measure, $sim(t_i, t_l)$, defined between any two tuples, $t_i, t_l \in D$, and an integer value k, the **clustering problem** is to define a mapping $f : D \rightarrow \{1, \ldots, k\}$ where each t_i is assigned to one cluster K_j, $1 \leq j \leq k$. Given a cluster, K_j, $\forall t_{jl}, t_{jm} \in K_j$ and $t_i \notin K_j$, $sim(t_{jl}, t_{jm}) > sim(t_{jl}, t_i)$.

A distance measure, $dis(t_i, t_j)$, as opposed to similarity, is often used in clustering. The clustering problem then has the desirable property that given a cluster, K_j, $\forall t_{jl}$, $t_{jm} \in K_j$ and $t_i \notin K_j$, $dis(t_{jl}, t_{jm}) \leq dis(t_{jl}, t_i)$.

Some clustering algorithms look only at numeric data, usually assuming metric data points. *Metric* attributes satisfy the triangular inequality. The clusters can then be described by using several characteristic values. Given a cluster, K_m of N points $\{t_{m1}, t_{m2}, \ldots, t_{mN}\}$, we make the following definitions [ZRL96]:

$$\text{centroid} \quad = \quad C_m = \frac{\sum\limits_{i=1}^{N}(t_{mi})}{N} \tag{5.1}$$

$$\text{radius} \quad = \quad R_m = \sqrt{\frac{\sum\limits_{i=1}^{N}(t_{mi} - C_m)^2}{N}} \tag{5.2}$$

$$\text{diameter} \quad = \quad D_m = \sqrt{\frac{\sum\limits_{i=1}^{N}\sum\limits_{j=1}^{N}(t_{mi} - t_{mj})^2}{(N)(N-1)}} \tag{5.3}$$

Here the centroid is the "middle" of the cluster; it need not be an actual point in the cluster. Some clustering algorithms alternatively assume that the cluster is represented by one centrally located object in the cluster called a *medoid*. The radius is the square root of the average mean squared distance from any point in the cluster to the centroid, and the diameter is the square root of the average mean squared distance between all pairs of points in the cluster. We use the notation M_m to indicate the medoid for cluster K_m.

Many clustering algorithms require that the distance between clusters (rather than elements) be determined. This is not an easy task given that there are many interpretations for distance between clusters. Given clusters K_i and K_j, there are several standard alternatives to calculate the distance between clusters. A representative list is:

- **Single link:** Smallest distance between an element in one cluster and an element in the other. We thus have $\text{dis}(K_i, K_j) = \min(\text{dis}(t_{il}, t_{jm})) \forall t_{il} \in K_i \notin K_j$ and $\forall t_{jm} \in K_j \notin K_i$.

- **Complete link:** Largest distance between an element in one cluster and an element in the other. We thus have $\text{dis}(K_i, K_j) = \max(\text{dis}(t_{il}, t_{jm})) \forall t_{il} \in K_i \notin K_j$ and $\forall t_{jm} \in K_j \notin K_i$.

- **Average:** Average distance between an element in one cluster and an element in the other. We thus have $\text{dis}(K_i, K_j) = \text{mean}(\text{dis}(t_{il}, t_{jm})) \forall t_{il} \in K_i \notin K_j$ and $\forall t_{jm} \in K_j \notin K_i$.

- **Centroid:** If clusters have a representative centroid, then the centroid distance is defined as the distance between the centroids. We thus have $\text{dis}(K_i, K_j) = \text{dis}(C_i, C_j)$, where C_i is the centroid for K_i and similarly for C_j.

- **Medoid:** Using a medoid to represent each cluster, the distance between the clusters can be defined by the distance between the medoids: $\text{dis}(K_i, K_j) = \text{dis}(M_i, M_j)$.

5.3 OUTLIERS

As mentioned earlier, *outliers* are sample points with values much different from those of the remaining set of data. Outliers may represent errors in the data (perhaps a malfunctioning sensor recorded an incorrect data value) or could be correct data values that are simply much different from the remaining data. A person who is 2.5 meters tall is much taller than most people. In analyzing the height of individuals, this value probably would be viewed as an outlier.

Some clustering techniques do not perform well with the presence of outliers. This problem is illustrated in Figure 5.3. Here if three clusters are found (solid line), the outlier will occur in a cluster by itself. However, if two clusters are found (dashed line), the two (obviously) different sets of data will be placed in one cluster because they are closer together than the outlier. This problem is complicated by the fact that many clustering algorithms actually have as input the number of desired clusters to be found.

Clustering algorithms may actually find and remove outliers to ensure that they perform better. However, care must be taken in actually removing outliers. For example, suppose that the data mining problem is to predict flooding. Extremely high water level values occur very infrequently, and when compared with the normal water level values may seem to be outliers. However, removing these values may not allow the data mining algorithms to work effectively because there would be no data that showed that floods ever actually occurred.

Outlier detection, or *outlier mining*, is the process of identifying outliers in a set of data. Clustering, or other data mining, algorithms may then choose to remove or treat these values differently. Some outlier detection techniques are based on statistical techniques. These usually assume that the set of data follows a known distribution and that outliers can be detected by well-known tests such as *discordancy tests*. However, these
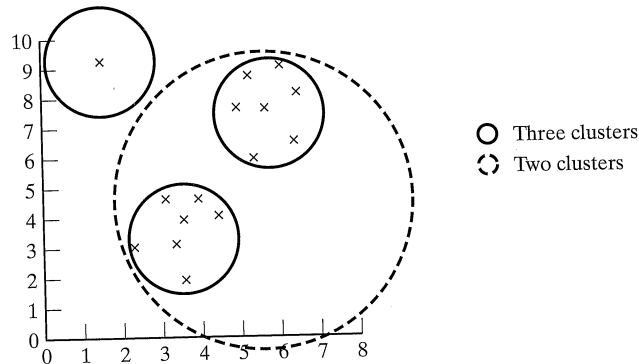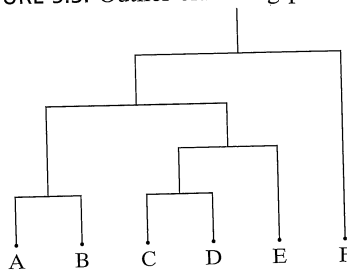
FIGURE 5.3: Outlier clustering problem.

FIGURE 5.4: Dendrogram for Example 5.2.

tests are not very realistic for real-world data because real-world data values may not follow well-defined data distributions. Also, most of these tests assume a single attribute value, and many attributes are involved in real-world datasets. Alternative detection techniques may be based on distance measures.

5.4 HIERARCHICAL ALGORITHMS

As mentioned earlier, hierarchical clustering algorithms actually creates sets of clusters. Example 5.2 illustrates the concept. Hierarchical algorithms differ in how the sets are created. A tree data structure, called a *dendrogram*, can be used to illustrate the hierarchical clustering technique and the sets of different clusters. The root in a dendrogram tree contains one cluster where all elements are together. The leaves in the dendrogram each consist of a single element cluster. Internal nodes in the dendrogram represent new clusters formed by merging the clusters that appear as its children in the tree. Each level in the tree is associated with the distance measure that was used to merge the clusters. All clusters created at a particular level were combined because the children clusters had a distance between them less than the distance value associated with this level in the tree. A dendrogram for Example 5.2 is seen in Figure 5.4.

EXAMPLE 5.2

Figure 5.5 shows six elements, {A, B, C, D, E, F}, to be clustered. Parts (a) to (e) of the figure show five different sets of clusters. In part (a) each cluster is viewed to consist of
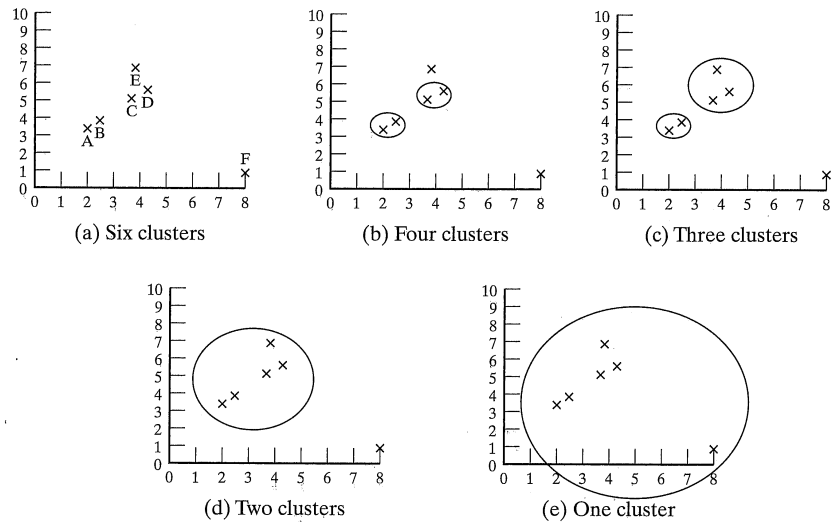
FIGURE 5.5: Five levels of clustering for Example 5.2.

a single element. Part (b) illustrates four clusters. Here there are two sets of two-element clusters. These clusters are formed at this level because these two elements are closer to each other than any of the other elements. Part (c) shows a new cluster formed by adding a close element to one of the two-element clusters. In part (d) the two-element and three-element clusters are merged to give a five-element cluster. This is done because these two clusters are closer to each other than to the remote element cluster, {F}. At the last stage, part (e), all six elements are merged.

The space complexity for hierarchical algorithms is $O(n^2)$ because this is the space required for the adjacency matrix. The space required for the dendrogram is $O(kn)$, which is much less than $O(n^2)$. The time complexity for hierarchical algorithms is $O(kn^2)$ because there is one iteration for each level in the dendrogram. Depending on the specific algorithm, however, this could actually be $O(maxd \ n^2)$ where $maxd$ is the maximum distance between points. Different algorithms may actually merge the closest clusters from the next lowest level or simply create new clusters at each level with progressively larger distances.

Hierarchical techniques are well suited for many clustering applications that naturally exhibit a nesting relationship between clusters. For example, in biology, plant and animal taxonomies could easily be viewed as a hierarchy of clusters.

5.4.1 Agglomerative Algorithms

Agglomerative algorithms start with each individual item in its own cluster and iteratively merge clusters until all items belong in one cluster. Different agglomerative algorithms differ in how the clusters are merged at each level. Algorithm 5.1 illustrates the typical agglomerative clustering algorithm. It assumes that a set of elements and distances between them is given as input. We use an $n \times n$ vertex adjacency matrix, A, as input.

Here the adjacency matrix, A, contains a distance value rather than a simple boolean value: $A[i, j] = \text{dis}(t_i, t_j)$. The output of the algorithm is a dendrogram, DE, which we represent as a set of ordered triples $\langle d, k, K \rangle$ where d is the threshold distance, k is the number of clusters, and K is the set of clusters. The dendrogram in Figure 5.7(a) would be represented by the following:

$$\{\langle 0, 5, \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}\}\rangle, \langle 1, 3, \{\{A, B\}, \{C, D\}, \{E\}\}\rangle$$
$$\langle 2, 2, \{\{A, B, C, D\}, \{E\}\}\rangle, \langle 3, 1, \{\{A, B, C, D, E\}\}\rangle\}$$

Outputting the dendrogram produces a set of clusters rather than just one clustering. The user can determine which of the clusters (based on distance threshold) he or she wishes to use.

ALGORITHM 5.1

```
Input:
    D = {t₁, t₂, ..., tₙ}    //Set of elements
    A        //Adjacency matrix showing distance between elements
Output:
    DE       // Dendrogram represented as a set of ordered triples
Agglomerative algorithm:
    d = 0;
    k = n;
    K = {{t₁}, ..., {tₙ}};
    DE = {⟨d, k, K⟩};  // Initially dendrogram contains each element
                          in its own cluster.
    repeat
        oldk = k;
        d = d + 1;
        A_d = Vertex adjacency matrix for graph with threshold
              distance of d;
        ⟨k, K⟩ = NewClusters(A_d, D);
        if oldk ≠ k then
            DE = DE ∪ ⟨d, k, K⟩;  // New set of clusters added to dendrogram.
        until k = 1
```

This algorithm uses a procedure called *NewClusters* to determine how to create the next level of clusters from the previous level. This is where the different types of agglomerative algorithms differ. It is possible that only two clusters from the prior level are merged or that multiple clusters are merged. Algorithms also differ in terms of which clusters are merged when there are several clusters with identical distances. In addition, the technique used to determine the distance between clusters may vary. *Single link*, *complete link*, and *average link* techniques are perhaps the most well known agglomerative techniques based on well-known graph theory concepts.

All agglomerative approaches experience excessive time and space constraints. The space required for the adjacency matrix is $O(n^2)$ where there are n items to cluster. Because of the iterative nature of the algorithm, the matrix (or a subset of it) must be accessed multiple times. The simplistic algorithm provided in Algorithm 5.1 performs at most *maxd* examinations of this matrix, where *maxd* is the largest distance between any two points. In addition, the complexity of the *NewClusters* procedure could be expensive. This is a potentially severe problem in large databases. Another issue with

the agglomerative approach is that it is not incremental. Thus, when new elements are added or old ones are removed or changed, the entire algorithm must be rerun. More recent incremental variations, as discussed later in this text, address this problem.

Single Link Technique. The single link technique is based on the idea of finding maximal connected components in a graph. A *connected component* is a graph in which there exists a path between any two vertices. With the single link approach, two clusters are merged if there is at least one edge that connects the two clusters; that is, if the minimum distance between any two points is less than or equal to the threshold distance being considered. For this reason, it is often called the *nearest neighbor* clustering technique. Example 5.3 illustrates this process.

EXAMPLE 5.3

Table 5.2 contains five sample data items with the distance between the elements indicated in the table entries. When viewed as a graph problem, Figure 5.6(a) shows the general graph with all edges labeled with the respective distances. To understand the idea behind the hierarchical approach, we show several graph variations in Figures 5.6(b), (c), (d), and (e). Figure 5.6(b) shows only those edges with a distance of 1 or less. There are only two edges. The first level of single link clustering then will combine the connected clusters (single elements from the first phase), giving three clusters: {A,B}, {C,D}, and {E}. During the next level of clustering, we look at edges with a length of 2 or less. The graph representing this threshold distance is shown in Figure 5.6(c). Note that we now have an edge (actually three) between the two clusters {A,B} and {C,D}. Thus, at this level of the single link clustering algorithm, we merge these two clusters to obtain a total of two clusters: {A,B,C,D} and {E}. The graph that is created with a threshold distance of 3 is shown in Figure 5.6(d). Here the graph is connected, so the two clusters from the last level are merged into one large cluster that contains all elements. The dendrogram for this single link example is shown in Figure 5.7(a). The labeling on the right-hand side shows the threshold distance used to merge the clusters at each level.

The single link algorithm is obtained by replacing the *NewClusters* procedure in the agglomerative algorithm with a procedure to find connected components of a graph. We assume that this connected components procedure has as input a graph (actually represented by a vertex adjacency matrix and set of vertices) and as outputs a set of

TABLE 5.2: Sample Data for Example 5.3

Item	A	B	C	D	E
A	0	1	2	2	3
B	1	0	2	4	3
C	2	2	0	1	5
D	2	4	1	0	3
E	3	3	5	3	0

(a) Graph with all distances   (b) Graph with threshold of 1   (c) Graph with threshold of 2

(d) Graph with threshold of 3   (e) Graph with threshold of 4

FIGURE 5.6: Graphs for Example 5.3.

(a) Single link   (b) Complete link   (c) Average link
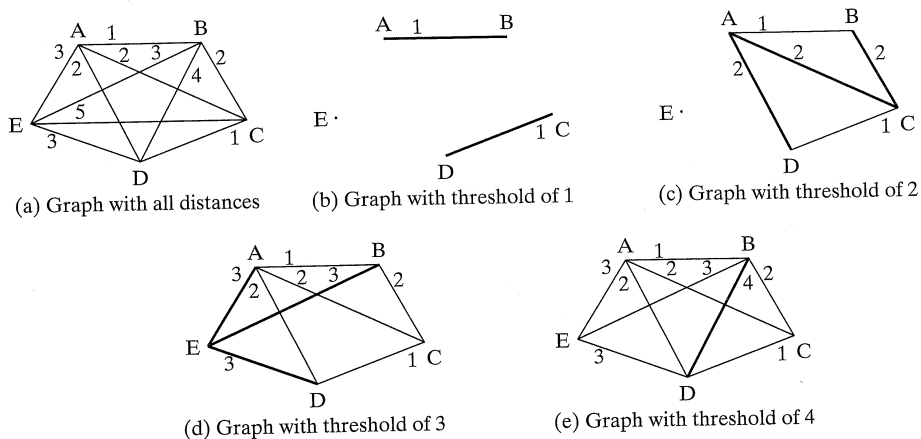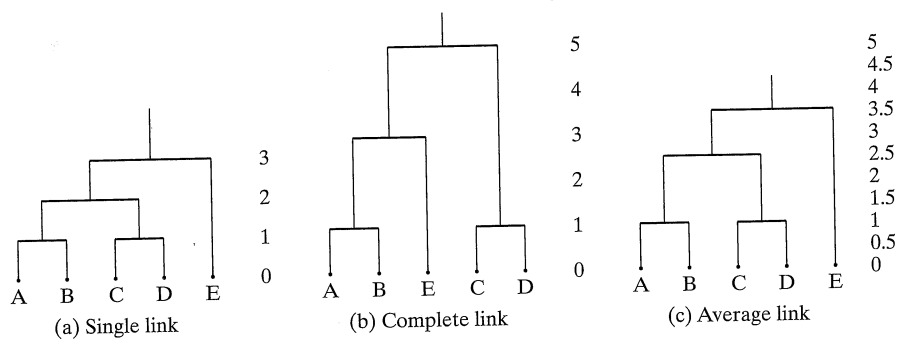
FIGURE 5.7: Dendrograms for Example 5.3.

connected components defined by a number (indicating the number of components) and an array containing the membership of each component. Note that this is exactly what the last two entries in the ordered triple are used for by the dendrogram data structure.

The single link approach is quite simple, but it suffers from several problems. This algorithm is not very efficient because the connected components procedure, which is an $O(n^2)$ space and time algorithm, is called at each iteration. A more efficient algorithm could be developed by looking at which clusters from an earlier level can be merged at each step. Another problem is that the clustering creates clusters with long chains.

An alternative view to merging clusters in the single link approach is that two clusters are merged at a stage where the threshold distance is d if the minimum distance between any vertex in one cluster and any vertex in the other cluster is at most d.

There have been other variations of the single link algorithm. One variation, based on the use of a *minimum spanning tree (MST)*, is shown in Algorithm 5.2. Here we assume that a procedure, *MST*, produces a minimum spanning tree given an adjacency matrix as input. The clusters are merged in increasing order of the distance found in the MST. In the algorithm we show that once two clusters are merged, the distance between

them in the tree becomes ∞. Alternatively, we could have replaced the two nodes and edge with one node.

ALGORITHM 5.2

```
Input:
    D = {t₁, t₂, ..., tₙ}    //Set of elements
    A        //Adjacency matrix showing distance between elements
Output:
    DE      // Dendrogram represented as a set of ordered triples
MST single link algorithm:
    d = 0
    k = n
    K = {{t₁}, ..., {tₙ}}
    DE = ⟨d, k, K⟩;  // Initially dendrogram contains each element in
            its own cluster.
    M = MST(A);
    repeat
        oldk = k;
        Kᵢ, Kⱼ = two clusters closest together in MST;
        K = K - {Kᵢ} - {Kⱼ} ∪ {Kᵢ ∪ Kⱼ};
        k = oldk - 1;
        d = dis(Kᵢ, Kⱼ);
        DE = DE ∪ ⟨d, k, K⟩;  // New set of clusters added to dendrogram.
        dis(Kᵢ, Kⱼ) = ∞
    until k = 1
```

We illustrate this algorithm using the data in Example 5.3. Figure 5.8 shows one MST for the example. The algorithm will merge A and B and then C and D (or the reverse). These two clusters will then be merged at a threshold of 2. Finally, E will be merged at a threshold of 3. Note that we get exactly the same dendrogram as in Figure 5.7(a).

The time complexity of this algorithm is $O(n^2)$ because the procedure to create the minimum spanning tree is $O(n^2)$ and it dominates the time of the algorithm. Once it is created having $n - 1$ edges, the *repeat* loop will be repeated only $n - 1$ times.

The single linkage approach is infamous for its chain effect; that is, two clusters are merged if only two of their points are close to each other. There may be points in the respective clusters to be merged that are far apart, but this has no impact on the algorithm. Thus, resulting clusters may have points that are not related to each other at all, but simply happen to be near (perhaps via a transitive relationship) points that are close to each other.
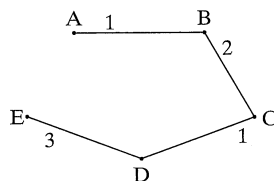

FIGURE 5.8: MST for Example 5.3.

Complete Link Algorithm.   Although the complete link algorithm is similar to the single link algorithm, it looks for cliques rather than connected components. A *clique* is a maximal graph in which there is an edge between any two vertices. Here a procedure is used to find the maximum distance between any clusters so that two clusters are merged if the maximum distance is less than or equal to the distance threshold. In this algorithm, we assume the existence of a procedure, clique, which finds all cliques in a graph. As with the single link algorithm, this is expensive because it is an $O(n^2)$ algorithm.

Clusters found with the complete link method tend to be more compact than those found using the single link technique. Using the data found in Example 5.3, Figure 5.7(b) shows the dendrogram created. A variation of the complete link algorithm is called the *farthest neighbor algorithm.* Here the closest clusters are merged where the distance is the smallest measured by looking at the maximum distance between any two points.

Average Link.   The average link technique merges two clusters if the average distance between any two points in the two target clusters is below the distance threshold. The algorithm used here is slightly different from that found in single and complete link algorithms because we must examine the complete graph (not just the threshold graph) at each stage. Thus, we restate this algorithm in Algorithm 5.3.

ALGORITHM 5.3

Input:
 $D = \{t_1, t_2, \ldots, t_n\}$ //Set of elements
 A //Adjacency matrix showing distance between elements
Output:
 DE // Dendrogram represented as a set of ordered triples
Average link algorithm:
 $d = 0$;
 $k = n$;
 $K = \{\{t_1\}, \ldots, \{t_n\}\}$;
 $DE = \langle d, k, K \rangle$; // Initially dendrogram contains each element
      in its own cluster.
 repeat
  $\text{old}k = k$;
  $d = d + 0.5$;
  for each pair of $K_i, K_j \in K$ **do**
   ave = average distance between all $t_i \in K_i$ and $t_j \in K_j$;
   if $ave \le d$, **then**
    $K = K - \{K_i\} - \{K_j\} \cup \{K_i \cup K_j\}$;
    $k = \text{old}k - 1$;
    $DE = DE \cup \langle d, k, K \rangle$; // New set of clusters added
         to dendrogram.
 until $k = 1$

Note that in this algorithm we increment d by 0.5 rather than by 1. This is a rather arbitrary decision based on understanding of the data. Certainly, we could have used an increment of 1, but we would have had a dendrogram different from that seen in Figure 5.7(c).

5.4.2 Divisive Clustering

With divisive clustering, all items are initially placed in one cluster and clusters are repeatedly split in two until all items are in their own cluster. The idea is to split up clusters where some elements are not sufficiently close to other elements.

One simple example of a divisive algorithm is based on the MST version of the single link algorithm. Here, however, we cut out edges from the MST from the largest to the smallest. Looking at Figure 5.8, we would start with a cluster containing all items: $\{A, B, C, D, E\}$. Looking at the MST, we see that the largest edge is between D and E. Cutting this out of the MST, we then split the one cluster into two: $\{E\}$ and $\{A, B, C, D\}$. Next we remove the edge between B and C. This splits the one large cluster into two: $\{A, B\}$ and $\{C, D\}$. These will then be split at the next step. The order depends on how a specific implementation would treat identical values. Looking at the dendrogram in Figure 5.7(a), we see that we have created the same set of clusters as with the agglomerative approach, but in reverse order.

5.5 PARTITIONAL ALGORITHMS

Nonhierarchical or partitional clustering creates the clusters in one step as opposed to several steps. Only one set of clusters is created, although several different sets of clusters may be created internally within the various algorithms. Since only one set of clusters is output, the user must input the desired number, k, of clusters. In addition, some metric or criterion function is used to determine the goodness of any proposed solution. This measure of quality could be the average distance between clusters or some other metric. The solution with the best value for the criterion function is the clustering solution used. One common measure is a squared error metric, which measures the squared distance from each point to the centroid for the associated cluster:

$$\sum_{m=1}^{k} \sum_{t_{mi} \in K_m} \text{dis}(C_m, t_{mi})^2 \tag{5.4}$$

A problem with partitional algorithms is that they suffer from a combinatorial explosion due to the number of possible solutions. Clearly, searching all possible clustering alternatives usually would not be feasible. For example, given a measurement criteria, a naive approach could look at all possible sets of k clusters. There are $S(n, k)$ possible combinations to examine. Here

$$S(n, k) = \frac{1}{k!} \sum_{i=1}^{k} (-1)^{k-i} \binom{k}{i} (i)^n \tag{5.5}$$

There are 11,259,666,000 different ways to cluster 19 items into 4 clusters. Thus, most algorithms look only at a small subset of all the clusters using some strategy to identify sensible clusters. Because of the plethora of partitional algorithms, we will look at only a representative few. We have chosen some of the most well known algorithms as well as some others that have appeared recently in the literature.

5.5.1 Minimum Spanning Tree

Since we have agglomerative and divisive algorithms based on the use of an MST, we also present a partitional MST algorithm. This is a very simplistic approach, but it

illustrates how partitional algorithms work. The algorithm is shown in Algorithm 5.4. Since the clustering problem is to define a mapping, the output of this algorithm shows the clusters as a set of ordered pairs $\langle t_i, j \rangle$ where $f(t_i) = K_j$.

ALGORITHM 5.4

```
Input:
    D = {t₁, t₂, ..., tₙ}   //Set of elements
    A        //Adjacency matrix showing distance between elements
    k        //Number of desired clusters
Output:
    f        //Mapping represented as a set of ordered pairs
Partitional MST algorithm:
    M = MST(A)
    identify inconsistent edges in M;
    remove k - 1 inconsistent edges;
    create output representation;
```

The problem is how to define "inconsistent." It could be defined as in the earlier division MST algorithm based on distance. This would remove the largest $k - 1$ edges from the starting completely connected graph and yield the same results as this corresponding level in the dendrogram. Zahn proposes more reasonable inconsistent measures based on the weight (distance) of an edge as compared to those close to it. For example, an inconsistent edge would be one whose weight is much larger than the average of the adjacent edges.

The time complexity of this algorithm is again dominated by the *MST* procedure, which is $O(n^2)$. At most, $k - 1$ edges will be removed, so the last three steps of the algorithm, assuming each step takes a constant time, is only $O(k - 1)$. Although determining the inconsistent edges in M may be quite complicated, it will not require a time greater than the number of edges in M. When looking at edges adjacent to one edge, there are at most $k - 2$ of these edges. In this case, then, the last three steps are $O(k^2)$, and the total algorithm is still $O(n^2)$.

5.5.2 Squared Error Clustering Algorithm

The *squared error* clustering algorithm minimizes the squared error. The *squared error for a cluster* is the sum of the squared Euclidean distances between each element in the cluster and the cluster centroid, C_k. Given a cluster K_i, let the set of items mapped to that cluster be $\{t_{i1}, t_{i2}, \ldots, t_{im}\}$. The squared error is defined as

$$se_{K_i} = \sum_{j=1}^{m} \|t_{ij} - C_k\|^2 \tag{5.6}$$

Given a set of clusters $K = \{K_1, K_2, \ldots, K_k\}$, the *squared error* for K is defined as

$$se_K = \sum_{j=1}^{k} se_{K_j} \tag{5.7}$$

In actuality, there are many different examples of squared error clustering algorithms. They all follow the basic algorithm structure shown in Algorithm 5.5.

ALGORITHM 5.5

```
Input:
    D = {t₁, t₂, ..., tₙ}    //Set of elements
    k        //Number of desired clusters
Output:
    K        //Set of clusters
Squared error algorithm:
    assign each item tᵢ to a cluster;
    calculate center for each cluster;
    repeat
        assign each item tᵢ to the cluster which has the closest center;
        calculate new center for each cluster;
        calculate squared error;
    until the difference between successive squared errors
        is below a threshold;
```

For each iteration in the squared error algorithm, each tuple is assigned to the cluster with the closest center. Since there are k clusters and n items, this is an $O(kn)$ operation. Assuming t iterations, this becomes an $O(tkn)$ algorithm. The amount of space may be only $O(n)$ because an adjacency matrix is not needed, as the distance between all items is not used.

5.5.3 K-Means Clustering

K-means is an iterative clustering algorithm in which items are moved among sets of clusters until the desired set is reached. As such, it may be viewed as a type of squared error algorithm, although the convergence criteria need not be defined based on the squared error. A high degree of similarity among elements in clusters is obtained, while a high degree of dissimilarity among elements in different clusters is achieved simultaneously. The *cluster mean* of $K_i = \{t_{i1}, t_{i2}, \ldots, t_{im}\}$ is defined as

$$m_i = \frac{1}{m} \sum_{j=1}^{m} t_{ij} \tag{5.8}$$

This definition assumes that each tuple has only one numeric value as opposed to a tuple with many attribute values. The K-means algorithm requires that some definition of cluster mean exists, but it does not have to be this particular one. Here the mean is defined identically to our earlier definition of centroid. This algorithm assumes that the desired number of clusters, k, is an input parameter. Algorithm 5.6 shows the K-means algorithm. Note that the initial values for the means are arbitrarily assigned. These could be assigned randomly or perhaps could use the values from the first k input items themselves. The convergence criteria could be based on the squared error, but they need not be. For example, the algorithm could stop when no (or a very small) number of tuples are assigned to different clusters. Other termination techniques have simply looked at a fixed number of iterations. A maximum number of iterations may be included to ensure stopping even without convergence.

ALGORITHM 5.6

```
Input:
    D = {t₁, t₂, ..., tₙ}    //Set of elements
```

```
    k       //Number of desired clusters
Output:
    K       //Set of clusters
K-means algorithm:
    assign initial values for means m₁, m₂, ..., mₖ;
    repeat
        assign each item tᵢ to the cluster which has the closest mean;
        calculate new mean for each cluster;
    until convergence criteria is met;
```

The K-means algorithm is illustrated in Example 5.4.

EXAMPLE 5.4

Suppose that we are given the following items to cluster:

$$\{2, 4, 10, 12, 3, 20, 30, 11, 25\} \tag{5.9}$$

and suppose that $k = 2$. We initially assign the means to the first two values: $m_1 = 2$ and $m_2 = 4$. Using Euclidean distance, we find that initially $K_1 = \{2, 3\}$ and $K_2 = \{4, 10, 12, 20, 30, 11, 25\}$. The value 3 is equally close to both means, so we arbitrarily choose K_1. Any desired assignment could be used in the case of ties. We then recalculate the means to get $m_1 = 2.5$ and $m_2 = 16$. We again make assignments to clusters to get $K_1 = \{2, 3, 4\}$ and $K_2 = \{10, 12, 20, 30, 11, 25\}$. Continuing in this fashion, we obtain the following:

m_1	m_2	K_1	K_2
3	18	$\{2, 3, 4, 10\}$	$\{12, 20, 30, 11, 25\}$
4.75	19.6	$\{2, 3, 4, 10, 11, 12\}$	$\{20, 30, 25\}$
7	25	$\{2, 3, 4, 10, 11, 12\}$	$\{20, 30, 25\}$

Note that the clusters in the last two steps are identical. This will yield identical means, and thus the means have converged. Our answer is thus $K_1 = \{2, 3, 4, 10, 11, 12\}$ and $K_2 = \{20, 30, 25\}$.

The time complexity of K-means is $O(tkn)$ where t is the number of iterations. K-means finds a local optimum and may actually miss the global optimum. K-means does not work on categorical data because the mean must be defined on the attribute type. Only convex-shaped clusters are found. It also does not handle outliers well. One variation of K-means, *K-modes*, does handle categorical data. Instead of using means, it uses modes. A typical value for k is 2 to 10.

Although the K-means algorithm often produces good results, it is not time-efficient and does not scale well. By saving distance information from one iteration to the next, the actual number of distance calculations that must be made can be reduced.

Some K-means variations examine ways to improve the chances of finding the global optimum. This often involves careful selection of the initial clusters and means. Another variation is to allow clusters to be split and merged. The variance within a cluster is examined, and if it is too large, a cluster is split. Similarly, if the distance between two cluster centroids is less than a predefined threshold, they will be combined.

5.5.4 Nearest Neighbor Algorithm

An algorithm similar to the single link technique is called the *nearest neighbor algorithm*. With this serial algorithm, items are iteratively merged into the existing clusters that are closest. In this algorithm a threshold, t, is used to determine if items will be added to existing clusters or if a new cluster is created.

ALGORITHM 5.7

```
Input:
    D = {t₁, t₂, ..., tₙ}    //Set of elements
    A        //Adjacency matrix showing distance between elements
Output:
    K        //Set of clusters
Nearest neighbor algorithm:
    K₁ = {t₁};
    K = {K₁};
    k = 1;
    for i = 2 to n do
        find the tₘ in some cluster Kₘ in K such that dis(tᵢ, tₘ) is
            the smallest;
        if dis(tᵢ, tₘ), ≤ t then
            Kₘ = Kₘ ∪ tᵢ
        else
            k = k + 1;
            Kₖ = {tᵢ};
```

Example 5.5 shows the application of the nearest neighbor algorithm to the data shown in Table 5.2 assuming a threshold of 2. Notice that the results are the same as those seen in Figure 5.7(a) at the level of 2.

EXAMPLE 5.5

Initially, A is placed in a cluster by itself, so we have $K_1 = \{A\}$. We then look at B to decide if it should be added to K_1 or be placed in a new cluster. Since $dis(A, B) = 1$, which is less than the threshold of 2, we place B in K_1 to get $K_1 = \{A, B\}$. When looking at C, we see that its distance to both A and B is 2, so we add it to the cluster to get $K_1 = \{A, B, C\}$. The $dis(D, C) = 1 < 2$, so we get $K_1 = \{A, B, C, D\}$. Finally, looking at E, we see that the closest item in K_1 has a distance of 3, which is greater than 2, so we place it in its own cluster: $K_2 = \{E\}$.

The complexity of the nearest neighbor algorithm actually depends on the number of items. For each loop, each item must be compared to each item already in a cluster. Obviously, this is n in the worst case. Thus, the time complexity is $O(n^2)$. Since we do need to examine the distances between items often, we assume that the space requirement is also $O(n^2)$.

5.5.5 PAM Algorithm

The *PAM (partitioning around medoids)* algorithm, also called the *K-medoids* algorithm, represents a cluster by a medoid. Using a medoid is an approach that handles outliers

well. The PAM algorithm is shown in Algorithm 5.8. Initially, a random set of k items is taken to be the set of medoids. Then at each step, all items from the input dataset that are not currently medoids are examined one by one to see if they should be medoids. That is, the algorithm determines whether there is an item that should replace one of the existing medoids. By looking at all pairs of medoid, non-medoid objects, the algorithm chooses the pair that improves the overall quality of the clustering the best and exchanges them. Quality here is measured by the sum of all distances from a non-medoid object to the medoid for the cluster it is in. An item is assigned to the cluster represented by the medoid to which it is closest (minimum distance). We assume that K_i is the cluster represented by medoid t_i. Suppose t_i is a current medoid and we wish to determine whether it should be exchanged with a non-medoid t_h. We wish to do this swap only if the overall impact to the cost (sum of the distances to cluster medoids) represents an improvement.

Following the lead in [NH94], we use C_{jih} to be the cost change for an item t_j associated with swapping medoid t_i with non-medoid t_h. The cost is the change to the sum of all distances from items to their cluster medoids. There are four cases that must be examined when calculating this cost:

1. $t_j \in K_i$, but $\exists$ another medoid t_m where $\mathrm{dis}(t_j, t_m) \leq \mathrm{dis}(t_j, t_h)$;
2. $t_j \in K_i$, but $\mathrm{dis}(t_j, t_h) \leq \mathrm{dis}(t_j, t_m) \forall$ other medoids t_m;
3. $t_j \in K_m, \notin K_i$, and $\mathrm{dis}(t_j, t_m) \leq \mathrm{dis}(t_j, t_h)$; and
4. $t_j \in K_m, \notin K_i$, but $\mathrm{dis}(t_j, t_h) \leq \mathrm{dis}(t_j, t_m)$.

We leave it as an exercise to determine the cost of each of these cases. The total impact to quality by a medoid change TC_{ih} then is given by

$$TC_{ih} = \sum_{j=1}^{n} C_{jih} \tag{5.10}$$

ALGORITHM 5.8

```
Input:
    D = {t₁, t₂, ..., tₙ}    //Set of elements
    A       //Adjacency matrix showing distance between elements
    k       //Number of desired clusters
Output:
    K       //Set of clusters
PAM algorithm:
    arbitrarily select k medoids from D;
    repeat
        for each tₕ not a medoid do
            for each medoid tᵢ do
                calculate TCᵢₕ;
        find i, h where TCᵢₕ is the smallest;
        if TCᵢₕ < 0, then
            replace medoid tᵢ with tₕ;
    until TCᵢₕ ≥ 0;
    for each tᵢ ∈ D do
        assign tᵢ to Kⱼ, where dis(tᵢ, tⱼ) is the smallest over all medoids;
```

Example 5.6 shows the application of the PAM algorithm to the data shown in Table 5.2 assuming a threshold of 2.

EXAMPLE 5.6

Suppose that the two medoids that are initially chosen are A and B. Based on the distances shown in Table 5.2 and randomly placing items when distances are identical to the two medoids, we obtain the clusters $\{A, C, D\}$ and $\{B, E\}$ The three non-medoids, $\{C, D, E\}$, are then examined to see which (if any) should be used to replace A or B. We thus have six costs to determine: TC_{AC}, TC_{AD}, TC_{AE}, TC_{BC}, TC_{BD}, and TC_{BE}. Here we use the name of the item instead of a numeric subscript value. We obtain the following:

$$TC_{AC} = C_{AAC} + C_{BAC} + C_{CAC} + C_{DAC} + C_{EAC} = 1 + 0 - 2 - 1 + 0 = -2 \quad (5.11)$$

Here A is no longer a medoid, and since it is closer to B, it will be placed in the cluster with B as medoid, and thus its cost is $C_{AAC} = 1$. The cost for B is 0 because it stays a cluster medoid. C is now a medoid, so it has a negative cost based on its distance to the old medoid; that is, $C_{CAB} = -2$. D is closer to C than it was to A by a distance of 1, so its cost is $C_{DAC} = -1$. Finally, E stays in the same cluster with the same distance, so its cost change is 0. Thus, we have that the overall cost is a reduction of 2. Figure 5.9 illustrates the calculation of these six costs. Looking at these, we see that the minimum cost is 2 and that there are several ways to reduce this cost. Arbitrarily choosing the first swap, we get C and B as the new medoids with the clusters being $\{C, D\}$ and $\{B, A, E\}$. This concludes the first iteration of PAM. At the next iteration, we examine changing medoids again and pick the choice that best reduces the cost. The iterations stop when no changes will reduce the cost. We leave the rest of this problem to the reader as an exercise.

PAM does not scale well to large datasets because of its computational complexity. For each iteration, we have $k(n-k)$ pairs of objects i, h for which a cost, TC_{ih}, should be determined. Calculating the cost during each iteration requires that the cost be calculated for all other non-medoids t_j. There are $n - k$ of these. Thus, the total complexity per iteration is $k(n - k)^2$. The total number of iterations can be quite large, so PAM is not an alternative for large databases. However, there are some clustering algorithms based on PAM that are targeted to large datasets.

CLARA (Clustering LARge Applications) improves on the time complexity of PAM by using samples of the dataset. The basic idea is that it applies PAM to a sample of the underlying database and then uses the medoids found as the medoids for the complete clustering. Each item from the complete database is then assigned to the cluster with the medoid to which it is closest. To improve the CLARA accuracy, several samples can be drawn with PAM applied to each. The sample chosen as the final clustering is the one that performs the best. Because of the sampling, CLARA is more efficient than PAM for large databases. However, it may not be as effective, depending on the sample size. Five samples of size $40 + 2k$ seem to give good results [KR90].

CLARANS (clustering large applications based upon randomized search) improves on CLARA by using multiple different samples. In addition to the normal input to PAM,
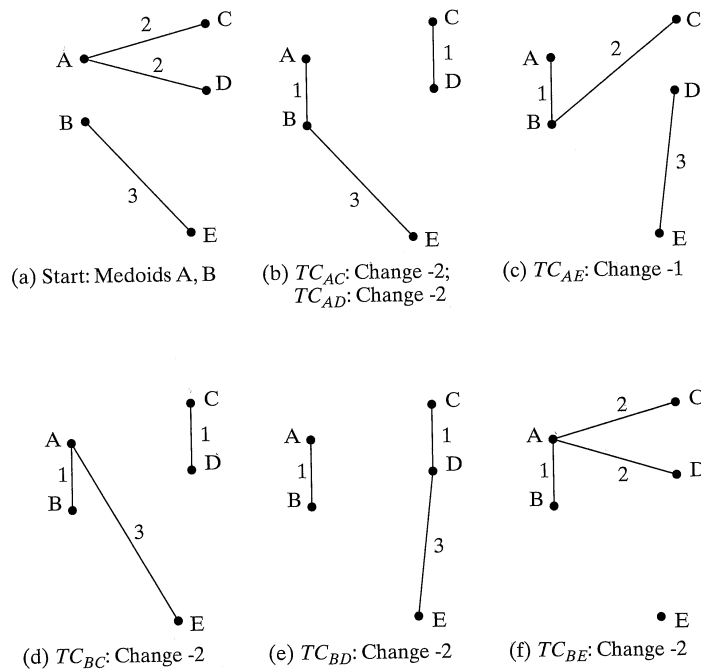
FIGURE 5.9: Cost calculations for Example 5.6.

CLARANS requires two additional parameters: *maxneighbor* and *numlocal*. *Maxneighbor* is the number of neighbors of a node to which any specific node can be compared. As *maxneighbor* increases, CLARANS looks more and more like PAM because all nodes will be examined. *Numlocal* indicates the number of samples to be taken. Since a new clustering is performed on each sample, this also indicates the number of clusterings to be made. Performance studies indicate that *numlocal* = 2 and *maxneighbor* = max$((0.0125 \times k(n - k)), 250)$ are good choices [NH94]. CLARANS is shown to be more efficient than either PAM or CLARA for any size dataset. CLARANS assumes that all data are in main memory. This certainly is not a valid assumption for large databases.

5.5.6 Bond Energy Algorithm

The *bond energy algorithm (BEA)* was developed and has been used in the database design area to determine how to group data and how to physically place data on a disk. It can be used to cluster attributes based on usage and then perform logical or physical design accordingly. With BEA, the *affinity* (bond) between database attributes is based on common usage. This bond is used by the clustering algorithm as a similarity measure. The actual measure counts the number of times the two attributes are used together in a given time. To find this, all common queries must be identified.

The idea is that attributes that are used together form a cluster and should be stored together. In a distributed database, each resulting cluster is called a *vertical fragment* and