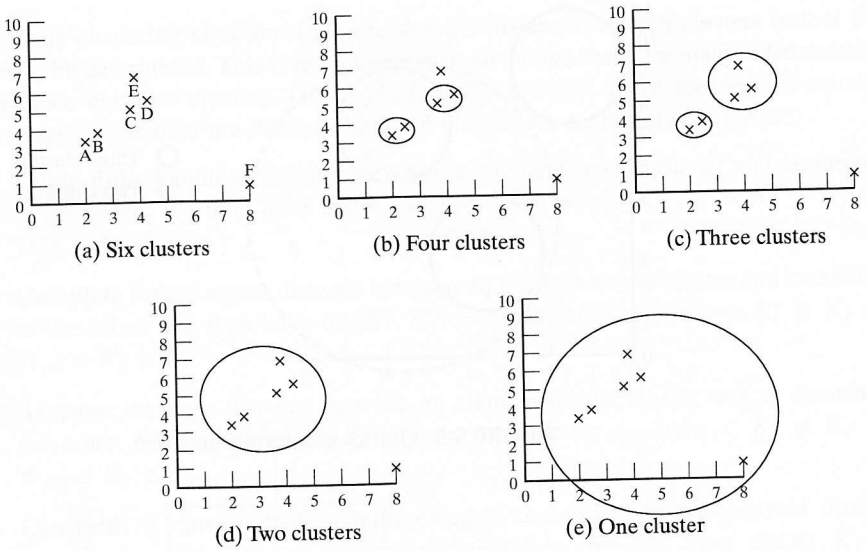


(a) Six clusters (b) Four clusters (c) Three clusters

(d) Two clusters (e) One cluster

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;
      Ad = Vertex adjacency matrix for graph with threshold
           distance of d;
      ⟨k, K⟩ = NewClusters(Ad, 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.

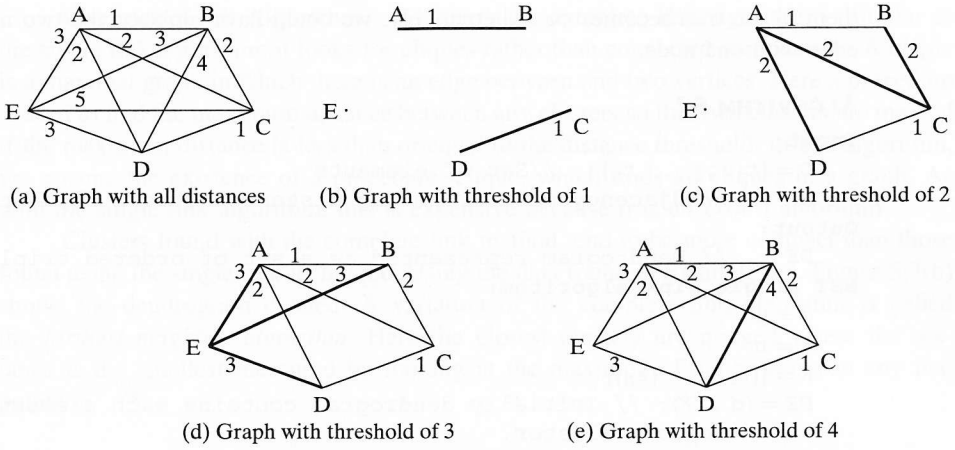
EXAMPLE 5.3

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

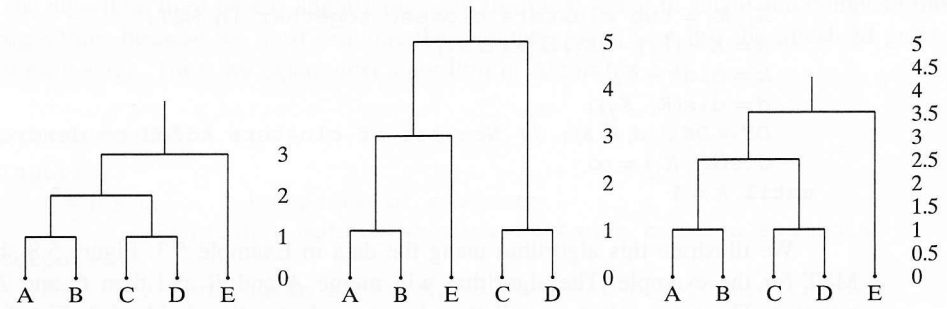
Item	A	B	C	D	E
A	0	1	2	2	3
B	1	0	2	4	3
C	2	2	0	1	5
D	2	4	1	0	3
E	3	3	5	3	0



(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.

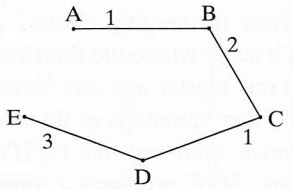


FIGURE 5.8: MST for Example 5.3.

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 = {t1, t2, ..., tn}   //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 = {{t1}, ..., {tn}};
   DE = ⟨d, k, K⟩; // Initially dendrogram contains each element
                   in its own cluster.
   repeat
      oldk = k;
      d = d + 0.5;
      for each pair of Ki, Kj ∈ K do
         ave = average distance between all ti ∈ Ki and tj ∈ Kj;
         if ave ≤ d, then
            K = K − {Ki} − {Kj} ∪ {Ki ∪ Kj};
            k = oldk − 1;
            DE = DE ∪ ⟨d, k, K⟩; // 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 $\text{dis}(A, B) = 1$, 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 $\text{dis}(D, C) = 1 < 2$, 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 $O(n^2)$.

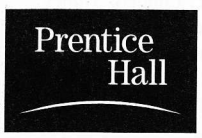
5.5.5 PAM Algorithm

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

Data Mining
Introductory and Advanced Topics

Margaret H. Dunham
Southern Methodist University

An Alan R. Apt Book



PEARSON EDUCATION INC.
Upper Saddle River, New Jersey 07458