UNIT V

Syllabus: Clustering: Clustering Techniques, Cluster analysis, Partitioning Methods, Hierarchical methods, Density Based Methods, Grid Based Methods, Evaluation of clustering, Clustering high dimensional data, Clustering with constraints, Outlier analysis, outlier detection methods.

Cluster is a group of objects that belongs to the same class. In other words, similar objects are grouped in one cluster and dissimilar objects are grouped in another cluster.

What is Clustering?

Clustering is the process of making a group of abstract objects into classes of similar objects.

Points to Remember

- A cluster of data objects can be treated as one group.
- While doing cluster analysis, we first partition the set of data into groups based on data similarity and then assign the labels to the groups.
- The main advantage of clustering over classification is that, it is adaptable to changes and helps single out useful features that distinguish different groups.

Cluster Analysis:

- The process of grouping a set of physical or abstract objects into classes of similar objectsis called clustering.
- A cluster is a collection of data objects that are similar to one another within the samecluster and are dissimilar to the objects in other clusters.
- A cluster of data objects can be treated collectively as one group and so may be consideredas a form of data compression.
- Cluster analysis tools based on k-means, k-medoids, and several methods have also been built into many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS.

Applications:

- Cluster analysis has been widely used in numerous applications, including market research,pattern recognition, data analysis, and image processing.
- In business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns.
- In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations.

- Clustering may also help in the identification of areas of similar land use in an earth observation database and in the identification of groups of houses in a city according to house type, value, and geographic location, as well as the identification of groups of automobile insurance policy holders with a high average claim cost.
- Clustering is also called data segmentation in some applications because clustering partitions large data sets into groups according to their *similarity*.
- Clustering can also be used for outlier detection, Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce.

Typical Requirements Of Clustering In Data Mining:

- ***Scalability:***

  Many clustering algorithms work well on small data sets containing fewer than several hundred data objects; however, a large database may contain millions of objects. Clusteringon a sample of a given large data set may lead to biased results.

  Highly scalable clustering algorithms are needed.

- ***Ability to deal with different types of attributes:***

  Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical (nominal), and ordinal data, or mixtures of these data types.

- ***Discovery of clusters with arbitrary shape:***

  Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density.

  However, a cluster could be of any shape. It is important to develop algorithms that can detect clusters of arbitrary shape.

- ***Minimal requirements for domain knowledge to determine input parameters:***

  Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results can be quite sensitive toinput parameters. Parameters are often difficult to determine, especially for data setscontaining high-dimensional objects. This not only burdens users, but it also makes thequality of clustering difficult to control.

- ***Ability to deal with noisy data:***

  Most real-world databases contain outliers or missing, unknown, or erroneous data.

  Some clustering algorithms are sensitive to such data and may lead to clusters of poorquality.

- ***Incremental clustering and insensitivity to the order of input records:***

  Some clustering algorithms cannot incorporate newly inserted data (i.e., database updates) into existing clustering structures and, instead, must determine a new clustering fromscratch. Some clustering algorithms are sensitive to the order of input data.

  That is, given a set of data objects, such an algorithm may return dramatically different clusterings depending on the order of presentation of the input objects.

  It is important to develop incremental clustering algorithms and algorithms that areinsensitive to the order of input.

- ***High dimensionality:***

  A database or a data warehouse can contain several dimensions or attributes. Many clustering algorithms are good at handling low-dimensional data, involving only two to three dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. Finding clusters of data objects in high dimensional space is challenging, especially considering that such data can be sparse and highly skewed.

- ***Constraint-based clustering:***

  Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your job is to choose the locations for a given number of new automatic banking machines (ATMs) in a city. To decide upon this, you may cluster householdswhile considering constraints such as the city's rivers and highway networks, and the type and number of customers per cluster. A challenging task is to find groups of data with good clustering behavior that satisfy specified constraints.

- ***Interpretability and usability:***

  Users expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied to specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering features and methods.

Major Clustering Methods:

- Partitioning Methods

- Hierarchical Methods
- Density-Based Methods
- Grid-Based Methods
- Model-Based Methods

Partitioning Methods:

A partitioning method constructs *k* partitions of the data, where each partition represents a cluster and *k* <= *n*. That is, it classifies the data into *k* groups, which together satisfy the following requirements:

- Each group must contain at least one object, and
- Each object must belong to exactly one group.

A partitioning method creates an initial partitioning. It then uses an iterative relocation technique that attempts to improve the partitioning by moving objects from one group to another.

The general criterion of a good partitioning is that objects in the same cluster are close or related to each other, whereas objects of different clusters are far apart or very different.

Hierarchical Methods:

A hierarchical method creates a hierarchical decomposition of the given set of data objects. A hierarchical method can be classified as being either agglomerative or divisive, based on how the hierarchical decomposition is formed.

- The agglomerative approach, also called the bottom-up approach, starts with each object forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one or until a termination condition holds.
- The divisive approach, also called the top-down approach, starts with all of the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a

termination condition holds.

Hierarchical methods suffer from the fact that once a step (merge or split) is done, it can never be undone. This rigidity is useful in that it leads to smaller computation costs by not having to worry about a combinatorial number of different choices.

There are two approaches to improving the quality of hierarchical clustering:

- Perform careful analysis of object –linkages|| at each hierarchical partitioning, such as in Chameleon, or
- Integrate hierarchical agglomeration and other approaches by first using a hierarchical agglomerative algorithm to group objects into micro clusters, and then performing macro clustering on the micro clusters using another clustering method such as iterative relocation.

Density-based methods:

- Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty at discovering clusters of arbitrary shapes.
- Other clustering methods have been developed based on the notion of density. Their general idea is to continue growing the given cluster as long as the density in the neighborhood exceeds some threshold; that is, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to filter out noise (outliers) and discover clusters of arbitrary shape.
- DBSCAN and its extension, OPTICS, are typical density-based methods that grow clusters according to a density-based connectivity analysis. DENCLUE is a method that clusters objects based on the analysis of the value distributions of density functions.

Grid-Based Methods:

- Grid-based methods quantize the object space into a finite number of cells that form a grid structure.
- All of the clustering operations are performed on the grid structure i.e., on the quantized space. The main advantage of this approach is its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space.
- STING is a typical example of a grid-based method. Wave Cluster applies wavelet transformation for clustering analysis and is both grid-based and density-based.

Model-Based Methods:

- Model-based methods hypothesize a model for each of the clusters and find the best fit of the data to the given model.
- A model-based algorithm may locate clusters by constructing a density function that reflects the spatial distribution of the data points.
- It also leads to a way of automatically determining the number of clusters based on standard statistics, taking –noise|| or outliers into account and thus yielding robust clustering methods.

Tasks in Data Mining:

- Clustering High-Dimensional Data
- Constraint-Based Clustering

Clustering High-Dimensional Data:

- It is a particularly important task in cluster analysis because many applications require the analysis of objects containing a large number of features or dimensions.
- For example, text documents may contain thousands of terms or keywords asfeatures, and DNA micro array data may provide information on the expression levels of thousands of genes under hundreds of conditions.
- Clustering high-dimensional data is challenging due to the curse of dimensionality. • Many dimensions may not be relevant. As the number of dimensions increases, The data become increasingly sparse so that the distance measurement between pairs of points become meaningless and the average density of points anywhere in the data is likely to be low. Therefore, a different clustering methodology needs to be developed for high-dimensional data.
- CLIQUE and PROCLUS are two influential subspace clustering methods, which search for clusters in subspaces of the data, rather than over the entire data space.
- Frequent pattern–based clustering, another clustering methodology, extracts distinct frequent patterns among subsets of dimensions that occur frequently. It uses such patterns to group objects and generate meaningful clusters.

Constraint-Based Clustering:

- It is a clustering approach that performs clustering by incorporation of user-specified or application-oriented constraints.
- A constraint expresses a user's expectation or describes properties of the desired clustering results, and provides an effective means for communicating with the clustering process.
- Various kinds of constraints can be specified, either by a user or as per application requirements.
- Spatial clustering employs with the existence of obstacles and clustering under user- specified

constraints. In addition, semi-supervised clustering employs for pair wise constraints in order to improve the quality of the resulting clustering.

Classical Partitioning Methods:

The most well-known and commonly used partitioning methods are

- The *k*-Means Method
- k-Medoids Method

Centroid-Based Technique: The *K*-Means Method:

The k-means algorithm takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intracluster similarity is high but the intercluster similarity is low. Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster's centroid or center of gravity.

The *k*-means algorithm proceeds as follows.

- First, it randomly selects *k* of the objects, each of which initially represents a clustermean or center.
- For each of the remaining objects, an object is assigned to the cluster to which it is themost similar, based on the distance between the object and the cluster mean.
- It then computes the new mean for each cluster.
- This process iterates until the criterion function converges.

Typically, the square-error criterion is used, defined as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2,$$

Where E is the sum of the square error for all objects in the data setp

is the point in space representing a given object

m_i is the mean of cluster C_i

The k-means partitioning algorithm:

The *k*-means algorithm for partitioning, where each cluster's center is represented by the meanvalue of the objects in the cluster.

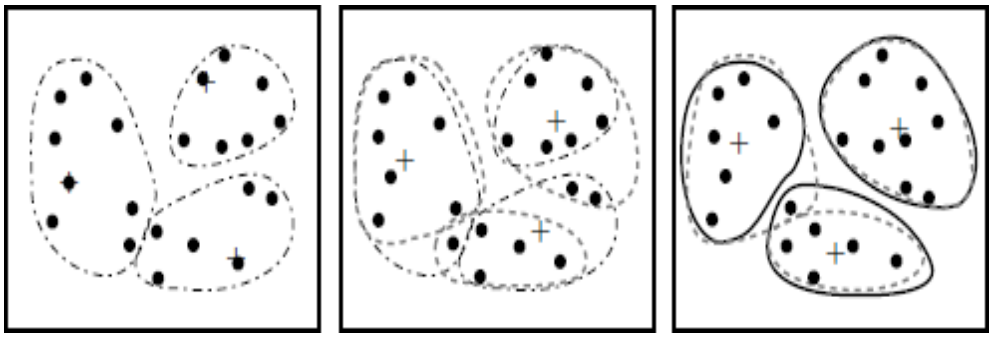
Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects from *D* as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, i.e., calculate the mean value of the objects for each cluster;
(5) **until** no change;



Clustering of a set of objects based on the *k*-means method

Advantages Of K-Means

Relatively efficient: O(tkn), where n is # objects, k is # clusters, and t is # iterations. Normally, k, t << n.

Each object is distributed to a cluster based on the cluster center to which it is the nearest.
Disadvantages Of K-Means

Applicable only when mean is defined, then what about categorical data?

Need to specify k, the number of clusters, in advance.

Unable to handle noisy data and outliers.

Not suitable to discover clusters with non-convex shapes.

The k-Medoids Method:

- The k-means algorithm is sensitive to outliers because an object with an extremely large value may substantially distort the distribution of data. This effect is particularly exacerbated due to the use of the square-error function.
- Instead of taking the mean value of the objects in a cluster as a reference point, we can pick actual objects to represent the clusters, using one representative object per cluster. Each remaining object is clustered with the representative object to which it is the most similar.
- The partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object and its corresponding reference point. That is, an absolute-error criterion is used, defined as

$$E = \sum_{j=1}^{k} \sum_{p \in C_j} |p - o_j|,$$

Where E is the sum of the absolute error for all objects in the data set

$\boldsymbol{p}$ is the point in space representing a given object in cluster C_j

$\mathbf{o}_j$ is the representative object of C_j

- The initial representative objects are chosen arbitrarily. The iterative process of replacing representative objects by non representative objects continues as long as the quality of the resulting clustering is improved.
- This quality is estimated using a cost function that measures the average dissimilarity between an object and the representative object of its cluster.
- To determine whether a non representative object, oj random, is a good replacement for a current representative object, oj, the following four cases are examined for each of the nonrepresentative objects.

Case 1:

p currently belongs to representative object, o_j. If $o_j$is replaced by o_{random} as a representative objectand p is closest to one of the other representative objects, o_i, $i \neq j$, then p is reassigned to o_i.

Case 2:

p currently belongs to representative object, oj. If oj is replaced by o_{random} as a representative objectand p is closest to o_{random}, then p is reassigned to o_{random}.

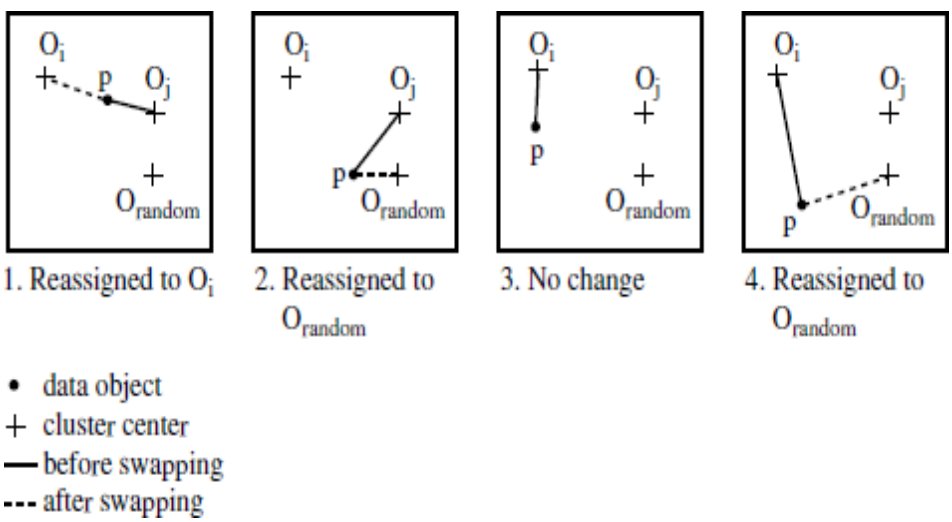
Case 3:

p currently belongs to representative object, o_i, $i \neq j$. If oj is replaced by o_{random} as a representativeobject and p is still closest to o_i, then the assignment does not change.

Case 4:

p currently belongs to representative object, oi, $i \neq j$. If o_j is replaced by o_{random} as a representativeobject and p is closest to o_{random}, then p is reassigned

too$_{random}$.



Four cases of the cost function for *k*-medoids clustering

The *k*-Medoids Algorithm:

The k-medoids algorithm for partitioning based on medoid or central objects.

Input:

- k: the number of clusters,
- D: a data set containing n objects.

Output: A set of k clusters.

Method:

(1) arbitrarily choose k objects in D as the initial representative objects or seeds;
(2) **repeat**
(3) assign each remaining object to the cluster with the nearest representative object;
(4) randomly select a nonrepresentative object, o_{random};
(5) compute the total cost, S, of swapping representative object, o_j, with o_{random};
(6) **if** $S < 0$ **then** swap o_j with o_{random} to form the new set of k representative objects;
(7) **until** no change;

The *k*-medoids method is more robust than *k*-means in the presence of noise and outliers, because a medoid is less influenced by outliers or other extreme values than a mean. However, its processing is more costly than the *k*-means method.

Which Is More Robust -- K-Means or K-Medoids

The k-medoids method is more robust than k-means in the presence of noise and outliers because a medoid is less influenced by outliers or other extreme values than a mean.

However, its processing is more costly than the k-means method. Both methods require the user to specify k, the number of clusters.

Aside from using the mean or the medoid as a measure of cluster center, other alternative measures are also commonly used in partitioning clustering methods.

The median can be used, resulting in the k-median method, where the median or “middle value” is taken for each ordered attribute. Alternatively, in the k-modes method, the most frequent value for each attribute is used.

Hierarchical Methods

While partitioning methods meet the basic clustering requirement of organizing a set of objects into a number of exclusive groups, in some situations we may want to partition our data into groups at

different levels such as in a hierarchy. A **hierarchical clustering method** works by grouping data objects into a hierarchy or "tree" of clusters.

Representing data objects in the form of a hierarchy is useful for data summarization and visualization. For example, as the manager of human resources at *AllElectronics*,

you may organize your employees into major groups such as executives, managers, andstaff. You can further partition these groups into smaller subgroups. For instance, thegeneral group of staff can be further divided into subgroups of senior officers, officers, and trainees. All these groups form a hierarchy. We can easily summarize or characterize the data that are organized into a hierarchy, which can be used to find, say, the average salary of managers and of officers.

Consider handwritten character recognition as another example. A set of handwrit- ing samples may be first partitioned into general groups where each group corresponds to a unique character. Some groups can be further partitioned into subgroups since a character may be written in multiple substantially different ways. If necessary, the hierarchical partitioning can be continued recursively until a desired granularity is reached.

In the previous examples, although we partitioned the data hierarchically, we did not assume that the data have a hierarchical structure (e.g., managers are at the same level in our *AllElectronics* hierarchy as staff). Our use of a hierarchy here is just to summarizeand represent the underlying data in a compressed way. Such a hierarchy is particularly useful for data visualization.

Alternatively, in some applications we may believe that the data bear an underly- ing hierarchical structure that we want to discover. For example, hierarchical clustering may uncover a hierarchy for *AllElectronics* employees structured on, say, salary. In the study of evolution, hierarchical clustering may group animals according to their bio-logical features to uncover evolutionary paths, which are a hierarchy of species. Asanother example, grouping configurations of a strategic game (e.g., chess or checkers) in a hierarchical way may help to develop game strategies that can be used to train players. In this section, you will study hierarchical clustering methods. Section 10.3.1 begins with a discussion of agglomerative versus divisive hierarchical clustering, which organize objects into a hierarchy using a bottom-up or top-down strategy, respectively. Agglo-merative methods start with individual objects as clusters, which are iteratively mergedto form larger clusters. Conversely, divisive methods initially let all the given objects

form one cluster, which they iteratively split into smaller clusters.

Hierarchical clustering methods can encounter difficulties regarding the selection of merge or split points. Such a decision is critical, because once a group of objects is merged or split, the process at the next step will operate on the newly generated clusters. It will neither undo what was done previously, nor perform object swapping betweenclusters. Thus, merge or split decisions, if not well chosen, may lead to low-quality clusters. Moreover, the methods do not scale well because each decision of merge orsplit needs to examine and evaluate many objects or clusters.

A promising direction for improving the clustering quality of hierarchical meth- ods is to integrate hierarchical clustering with other clustering techniques, resulting in **multiple-phase** (or **multiphase**) **clustering**. We introduce two such methods, namely BIRCH and Chameleon. BIRCH (Section 10.3.3) begins by partitioning objects hierar- chically using tree structures, where the leaf or low-level nonleaf nodes can beviewed as "microclusters" depending on the resolution scale. It then applies other

clustering algorithms to perform macroclustering on the microclusters. Chameleon (Section 10.3.4) explores dynamic modeling in hierarchical clustering.

There are several orthogonal ways to categorize hierarchical clustering methods. For instance, they may be categorized into *algorithmic* methods, *probabilistic* methods, and *Bayesian* methods. Agglomerative, divisive, and multiphase methods are *algorithmic*, meaning they consider data objects as deterministic and compute clusters according to the deterministic distances between objects.

Probabilistic methods use probabilistic models to capture clusters and measure the quality of clusters by the fitness of mod- els. We discuss probabilistic hierarchical clustering in Section 10.3.5. *Bayesian methods* compute a distribution of possible clusterings. That is, instead of outputting a single deterministic clustering over a data set, they return a group of clustering structures and their probabilities, conditional on the given data. Bayesian methods are considered an advanced topic and are not discussed in this book.

Agglomerative versus Divisive Hierarchical Clustering

A hierarchical clustering method can be either *agglomerative* or *divisive*, depending on whether the hierarchical decomposition is formed in a bottom-up (merging) or top- down (splitting) fashion. Let's have a closer look at these strategies.

An **agglomerative hierarchical clustering method** uses a bottom-up strategy. It typ- ically starts by letting each object form its own cluster and iteratively merges clusters into larger and larger clusters, until all the objects are in a single cluster or certain termi- nation conditions are satisfied. The single cluster becomes the hierarchy's root. For the merging step, it finds the two clusters that are closest to each other (according to some similarity measure), and combines the two to form one cluster. Because two clusters are merged per iteration, where each cluster contains at least one object, an agglomerative method requires at most *n* iterations.

A **divisive hierarchical clustering method** employs a top-down strategy. It starts by placing all objects in one cluster, which is the hierarchy's root. It then divides the root cluster into several smaller subclusters, and recursively partitions those clusters into smaller ones. The partitioning process continues until each cluster at the lowest level is coherent enough—either containing only one object, or the objects within a cluster are sufficiently similar to each other.

In either agglomerative or divisive hierarchical clustering, a user can specify the desired number of clusters as a termination condition.

Example 10.3 Agglomerative versus divisive hierarchical clustering. Figure 10.6 shows the application of **AGNES** (AGglomerative NESting), an agglomerative hierarchical clustering method, and **DIANA** (DIvisive ANAlysis), a divisive hierarchical clustering method, on a data set of five objects, $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}$. Initially, AGNES, the agglomerative method, places each object into a cluster of its own. The clusters are then merged step-by-step according to some criterion. For example, clusters C_1 and C_2 may be merged if an object in C_1 and an object in C_2 form the minimum Euclidean distance between any two objects from

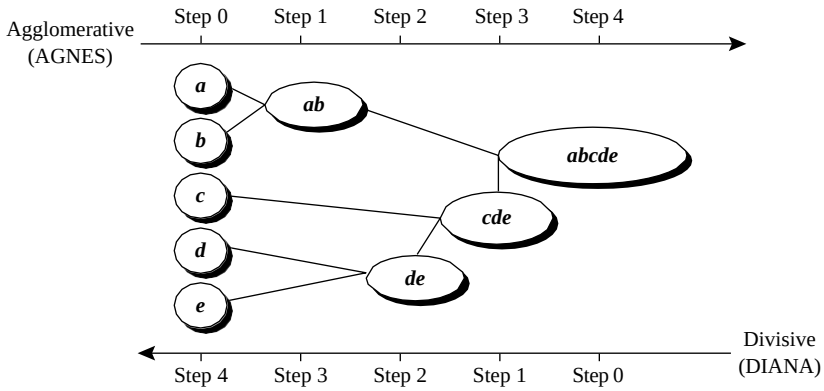


Figure 10.6 Agglomerative and divisive hierarchical clustering on data objects $\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}\}$.

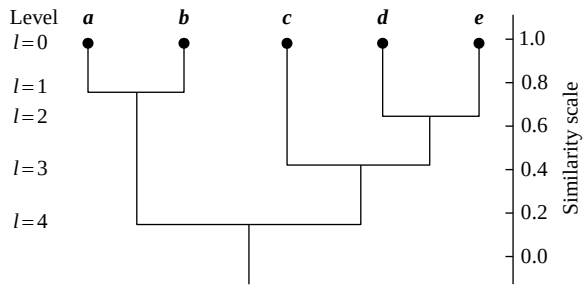


Figure 10.7 Dendrogram representation for hierarchical clustering of data objects $\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}\}$.

different clusters. This is a **single-linkage** approach in that each cluster is represented by all the objects in the cluster, and the similarity between two clusters is measured by the similarity of the *closest* pair of data points belonging to different clusters. The cluster-merging process repeats until all the objects are eventually merged to form onecluster.

DIANA, the divisive method, proceeds in the contrasting way. All the objects are used to form one initial cluster. The cluster is split according to some principle such as the maximum Euclidean distance between the closest neighboring objects in the cluster. The cluster-splitting process repeats until, eventually, each new cluster contains only a single object.

■

A tree structure called a **dendrogram** is commonly used to represent the process of hierarchical clustering. It shows how objects are grouped together (in an agglomerative method) or partitioned (in a divisive method) step-by-step. Figure 10.7 shows a den- drogram for the five objects presented in Figure 10.6, where $l = 0$ shows the five objects as singleton clusters at level 0. At $l = 1$, objects ***a*** and ***b*** are grouped together to form the

first cluster, and they stay together at all subsequent levels. We can also use a vertical axis to show the similarity scale between clusters. For example, when the similarity of two groups of objects, ***a***, ***b*** and ***c***, ***d***, ***e*** , is roughly 0.16, they are merged together to form a single cluster.

A challenge with divisive methods is how to partition a large cluster into several smaller ones. For example, there are 2^{n-1} 1 possible ways to partition a set of *n*-objects into two exclusive subsets, where *n* is the number of objects. When *n* is large, it is computationally prohibitive to examine all possibilities. Consequently, a divisive method typically uses heuristics in partitioning, which can lead to inaccurate results. For the sake of efficiency, divisive methods typically do not backtrack on partitioning decisions that have been made. Once a cluster is partitioned, any alternative partitioning of this cluster will not be considered again. Due to the challenges in divisive methods, there are many more agglomerative methods than divisive methods.

Distance Measures in Algorithmic Methods

Whether using an agglomerative method or a divisive method, a core need is to measure the distance between two clusters, where each cluster is generally a set of objects.

Four widely used measures for distance between clusters are as follows, where $|\boldsymbol{p} - \boldsymbol{p}'|$ is the distance between two objects or points, $\boldsymbol{p}$ and $\boldsymbol{p}'$; $\boldsymbol{m_i}$ is the mean for cluster, C_i; and n_i is the number of objects in C_i. They are also known as *linkage measures*.

Minimum distance: $$dist_{min}(C_i, C_j) = \min_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} \{|\boldsymbol{p} - \boldsymbol{p}'|\} \quad (10.3)$$

Maximum distance: $$dist_{max}(C_i, C_j) = \max_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} \{|\boldsymbol{p} - \boldsymbol{p}'|\} \quad (10.4)$$

Mean distance: $$dist_{mean}(C_i, C_j) = |\boldsymbol{m_i} - \boldsymbol{m_j}| \quad (10.5)$$

Average distance: $$dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} |\boldsymbol{p} - \boldsymbol{p}'| \quad (10.6)$$

When an algorithm uses the *minimum distance*, $d_{min}(C_i, C_j)$, to measure the distance between clusters, it is sometimes called a **nearest-neighbor clustering algorithm**. Moreover, if the clustering process is terminated when the distance between nearest clusters exceeds a user-defined threshold, it is called a **single-linkage algorithm**. If we view the data points as nodes of a graph, with edges forming a path between the nodes in a cluster, then the merging of two clusters, C_i and C_j, corresponds to adding an edge between the nearest pair of nodes in C_i and C_j. Because edges linking clusters always go between distinct clusters, the resulting graph will generate a tree. Thus, an agglomerative hierarchical clustering algorithm that uses the minimum distance measure is also called a **minimal spanning tree algorithm**, where a spanning tree of a graph is a tree that connects all vertices, and a minimal spanning tree is the one with the least sum of edgeweights.

When an algorithm uses the *maximum distance*, $d_{max}(C_i, C_j)$, to measure the distance between clusters, it is sometimes called a **farthest-neighbor clustering algorithm**. If the clustering process is terminated when the

maximum distance between nearest clusters exceeds a user-defined threshold, it is called a **complete-linkage algorithm**. By viewing data points as nodes of a graph, with edges linking nodes, we can think of each cluster as a *complete* subgraph, that is, with edges connecting all the nodes in the clusters. The distance between two clusters is determined by the most distant nodes in the two clusters. Farthest-neighbor algorithms tend to minimize the increase in diameter of the clusters at each iteration. If the true clusters are rather compact and approximately equal size, the method will produce high-quality clusters. Otherwise, the clusters produced can be meaningless.

The previous minimum and maximum measures represent two extremes in measuring the distance between clusters. They tend to be overly sensitive to outliers or noisy data. The use of *mean* or *average distance* is a compromise between the mini- mum and maximum distances and overcomes the outlier sensitivity problem. Whereas the *mean distance* is the simplest to compute, the *average distance* is advantageous in that it can handle categoric as well as numeric data. The computation of the mean vector for categoric data can be difficult or impossible to define.

Example 10.4 Single versus complete linkages. Let us apply hierarchical clustering to the data set of Figure 10.8(a). Figure 10.8(b) shows the dendrogram using single linkage. Figure 10.8(c) shows the case using complete linkage, where the edges between clusters *A*, *B*, *J* , *H* and *C*, *D*, *G*, *F*, *E* are omitted for ease of presentation. This example shows that by using single linkages we can find hierarchical clusters defined by local proximity, whereas complete linkage tends to find clusters opting for global closeness.

■

There are variations of the four essential linkage measures just discussed. For example, we can measure the distance between two clusters by the distance between the centroids (i.e., the central objects) of the clusters.

BIRCH: Multiphase Hierarchical Clustering Using Clustering Feature Trees

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is designed for clustering a large amount of numeric data by integrating hierarchical clustering (at the initial *microclustering* stage) and other clustering methods such as iterative partitioning (at the later *macroclustering* stage). It overcomes the two difficulties in agglomerative clustering methods: (1) scalability and (2) the inability to undo what was done in the previous step.

BIRCH uses the notions of *clustering feature* to summarize a cluster, and *clus- tering feature tree* (*CF-tree*) to represent a cluster hierarchy. These structures help

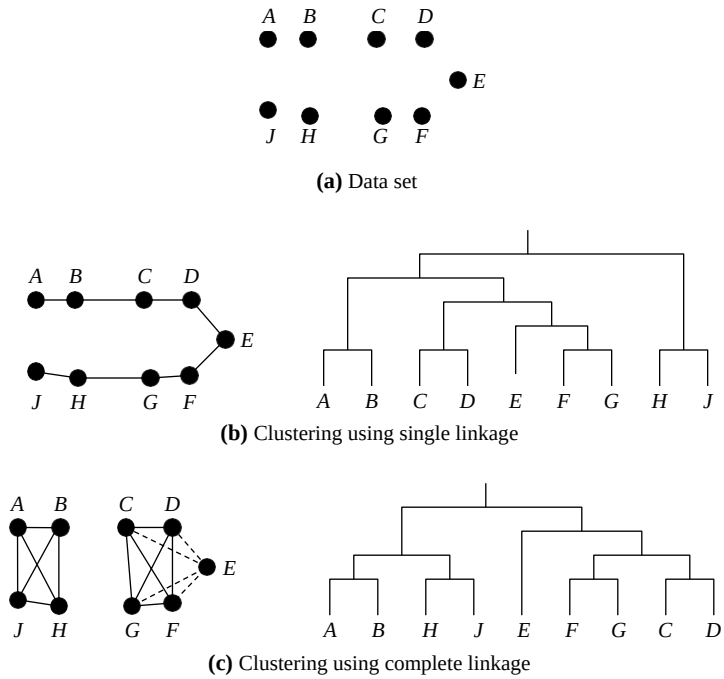


Figure 10.8 Hierarchical clustering using single and complete linkages.

the clustering method achieve good speed and scalability in large or even streaming databases, and also make it effective for incremental and dynamic clustering of incoming objects.

Consider a cluster of n d-dimensional data objects or points. The **clustering feature** (**CF**) of the cluster is a 3-D vector summarizing information about clusters of objects. It is defined as

the clustering method achieve good speed and scalability in large or even streaming databases, and also make it effective for incremental and dynamic clustering of incoming objects.

Consider a cluster of n d-dimensional data objects or points. The **clustering feature** (**CF**) of the cluster is a 3-D vector summarizing information about clusters of objects. It is defined as

$$CF = \langle n, LS, SS \rangle, \tag{10.7}$$

where LS is the linear sum of the n points (i.e., $\sum_{i=1}^{n} \boldsymbol{x_i}$), and SS is the square sum of the data points (i.e., $\sum_{i=1}^{n} \boldsymbol{x_i}^2$).

A clustering feature is essentially a summary of the statistics for the given cluster. Using a clustering feature, we can easily derive many useful statistics of a cluster. For example, the cluster's centroid, $\boldsymbol{x_0}$, radius, R, and diameter, D, are

$$\boldsymbol{x_0} = \frac{\sum_{i=1}^{n} \boldsymbol{x_i}}{n} = \frac{LS}{n}, \tag{10.8}$$

$$R = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x_0)^2}{n}} = \sqrt{\frac{nSS - 2LS^2 + nLS}{n^2}}, \qquad (10.9)$$

$$D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2LS^2}{n(n-1)}}. \qquad (10.10)$$

Here, R is the average distance from member objects to the centroid, and D is the aver-age pairwise distance within a cluster. Both R and D reflect the tightness of the clusteraround the centroid.
Summarizing a cluster using the clustering feature can avoid storing the detailed information about individual objects or points. Instead, we only need a constant size of space to store the clustering feature. This is the key to BIRCH efficiency in space. Moreover, clustering features are *additive*. That is, for two disjoint clusters, C_1 and C_2,with the clustering features $CF_1 = (n_1, LS_1, SS_1)$ and $CF_2 = (n_2, LS_2, SS_2)$, respectively,the clustering feature for the cluster that formed by merging C_1 and C_2 is simply

$$CF_1 + CF_2 = (n_1 + n_2, LS_1 + LS_2, SS_1 + SS_2). \qquad (10.11)$$

Example 10.5 **Clustering feature.** Suppose there are three points, $(2,5), (3,2)$, and $(4,3)$, in a cluster, C_1. The clustering feature of C_1 is

$$CF_1 = (3, (2+3+4, 5+2+3), (2^2+3^2+4^2, 5^2+2^2+3^2)) = (3, (9,10), (29,38)).$$

Suppose that C_1 is disjoint to a second cluster, C_2, where $CF_2 = (3, (35, 36), (417, 440))$. The clustering feature of a new cluster, C_3, that is formed by merging C_1 and C_2, is derived by adding CF_1 and CF_2. That is,

$$CF_3 = (3+3, (9+35, 10+36), (29+417, 38+440)) = (6, (44,46), (446,478)). \quad \blacksquare$$

A **CF-tree** is a height-balanced tree that stores the clustering features for a hierar-chical clustering. An example is shown in Figure 10.9. By definition, a nonleaf node in a tree has descendants or "children." The nonleaf nodes store sums of the CFs of their children, and thus summarize clustering information about their children. A CF-tree has two parameters: *branching factor*, B, and *threshold*, T. The branching factor specifies the maximum number of children per nonleaf node. The threshold parameter specifiesthe maximum diameter of subclusters stored at the leaf nodes of the tree. These twoparameters implicitly control the resulting tree's size.
Given a limited amount of main memory, an important consideration in BIRCHis to minimize the time required for input/output (I/O). BIRCH applies a *multiphase* clustering technique: A single scan of the data set yields a basic, good clustering, and

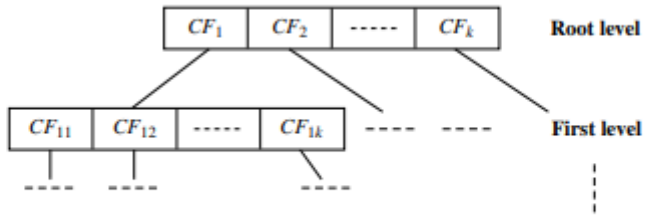


Figure 10.9 CF-tree structure.

one or more additional scans can optionally be used to further improve the quality. Theprimary phases are

Phase 1: BIRCH scans the database to build an initial in-memory CF-tree, which can be viewed as a multilevel compression of the data that tries to preserve the data'sinherent clustering structure.

Phase 2: BIRCH applies a (selected) clustering algorithm to cluster the leaf nodes of the CF-tree, which removes sparse clusters as outliers and groups dense clusters into larger ones.

For Phase 1, the CF-tree is built dynamically as objects are inserted. Thus, the method is incremental. An object is inserted into the closest leaf entry (subcluster). If the dia-meter of the subcluster stored in the leaf node after insertion is larger than the threshold value, then the leaf node and possibly other nodes are split. After the insertion of thenew object, information about the object is passed toward the root of the tree. The size of the CF-tree can be changed by modifying the threshold. If the size of the memory that is needed for storing the CF-tree is larger than the size of the main memory, then alarger threshold value can be specified and the CF-tree is rebuilt.
The rebuild process is performed by building a new tree from the leaf nodes of the old tree. Thus, the process of rebuilding the tree is done without the necessity of rereadingall the objects or points. This is similar to the insertion and node split in the construc-tion of B -trees. Therefore, for building the tree, data has to be read just once. Some heuristics and methods have been introduced to deal with outliers and improve the qual- ity of CF-trees by additional scans of the data. Once the CF-tree is built, any clustering algorithm, such as a typical partitioning algorithm, can be used with the CF-tree in Phase 2.
"How effective is BIRCH?" The time complexity of the algorithm is $O(n)$, where n is the number of objects to be clustered. Experiments have shown the linear scalabilityof the algorithm with respect to the number of objects, and good quality of clusteringof the data. However, since each node in a CF-tree can hold only a limited number of entries due to its size, a CF-tree node does not always correspond to what a user mayconsider a natural cluster. Moreover, if the clusters are not spherical in shape, BIRCH does not perform well because it uses the notion of radius or diameter to control the boundary of a cluster.

The ideas of clustering features and CF-trees have been applied beyond BIRCH. Theideas have been borrowed by many others to tackle problems of clustering streamingand dynamic data.

Chameleon: Multiphase Hierarchical Clustering Using Dynamic Modeling

Chameleon is a hierarchical clustering algorithm that uses dynamic modeling to deter-mine the similarity between pairs of clusters. In Chameleon, cluster similarity is assessed based on (1) how well connected objects are within a cluster and (2) the proximity ofclusters. That is, two clusters are merged if their

interconnectivity is high and they are*close together*. Thus, Chameleon does not depend on a static, user-supplied model andcan automatically adapt to the internal characteristics of the clusters being merged. The merge process facilitates the discovery of natural and homogeneous clusters and applies to all data types as long as a similarity function can be specified.

Figure 10.10 illustrates how Chameleon works. Chameleon uses a *k*-nearest-neighbor graph approach to construct a sparse graph, where each vertex of the graph represents a data object, and there exists an edge between two vertices (objects) if one object isamong the *k*-most similar objects to the other. The edges are weighted to reflect the similarity between objects. Chameleon uses a graph partitioning algorithm to partitionthe *k*-nearest-neighbor graph into a large number of relatively small subclusters such that it minimizes the **edge cut**. That is, a cluster C is partitioned into subclusters C_i and C_j so as to minimize the *weight of the edges* that would be cut should C be bisected into C_i and C_j. It assesses the *absolute* interconnectivity between clusters C_i and C_j.

Chameleon then uses an agglomerative hierarchical clustering algorithm that itera- tively merges subclusters based on their similarity. To determine the pairs of most similarsubclusters, it takes into account both the interconnectivity and the closeness of the clus- ters. Specifically, Chameleon determines the similarity between each pair of clusters $C_i$and C_j according to their *relative interconnectivity*, $RI(C_i, C_j)$, and their *relative closeness*, $RC(C_i, C_j)$.

The **relative interconnectivity**, $RI(C_i, C_j)$, between two clusters, C_i and C_j, is definedas the absolute interconnectivity between C_i and C_j, normalized with respect to the

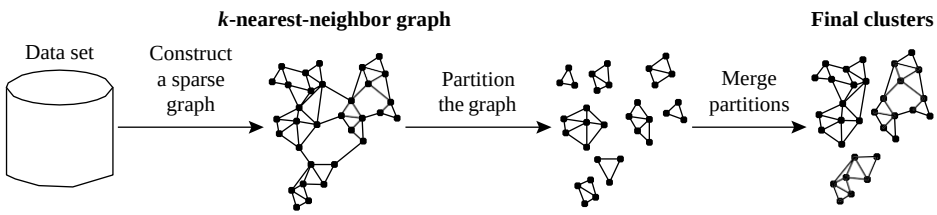


Figure 10.10 Chameleon: hierarchical clustering based on *k*-nearest neighbors and dynamic modeling. *Source:* Based on Karypis, Han, and Kumar [KHK99].

internal interconnectivity of the two clusters, C_i and C_j. That is,

$$RI(C_i, C_j) = \frac{|EC_{\{C_i,C_j\}}|}{\frac{1}{2}(|EC_{C_i}| + |EC_{C_j}|)}, \tag{10.12}$$

where $EC_{\{C_i,C_j\}}$ is the edge cut as previously defined for a cluster containing both C_i and C_j. Similarly, EC_{C_i} (or EC_{C_j}) is the minimum sum of the cut edges that partition C_i (or C_j) into two roughly equal parts.

- The **relative closeness**, $RC(C_i, C_j)$, between a pair of clusters, C_i and C_j, is the absolute closeness between C_i and C_j, normalized with respect to the internal closeness of the two clusters, C_i and C_j. It is defined as

$$RC(C_i, C_j) = \frac{\overline{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC_{C_j}}}, \tag{10.13}$$

where $\overline{S}_{EC_{\{C_i,C_j\}}}$ is the average weight of the edges that connect vertices in C_i to vertices in C_j, and $\overline{S}_{EC_{C_i}}$ (or $\overline{S}_{EC_{C_j}}$) is the average weight of the edges that belong to the min-cut bisector of cluster C_i (or C_j).

Chameleon has been shown to have greater power at discovering arbitrarily shaped clusters of high quality than several well-known algorithms such as BIRCH and density-based DBSCAN (Section 10.4.1). However, the processing cost for high-dimensional data may require $O(n^2)$ time for n objects in the worst case.

Probabilistic Hierarchical Clustering

Algorithmic hierarchical clustering methods using linkage measures tend to be easy to understand and are often efficient in clustering. They are commonly used in many clus-tering analysis applications. However, algorithmic hierarchical clustering methods can suffer from several drawbacks. First, choosing a good distance measure for hierarchical clustering is often far from trivial. Second, to apply an algorithmic method, the data objects cannot have any missing attribute values. In the case of data that are partially observed (i.e., some attribute values of some objects are missing), it is not easy to apply an algorithmic hierarchical clustering method because the distance computation cannot be conducted. Third, most of the algorithmic hierarchical clustering methods are heuris- tic, and at each step locally search for a good merging/splitting decision. Consequently, the optimization goal of the resulting cluster hierarchy can be unclear.

Probabilistic hierarchical clustering aims to overcome some of these disadvantages by using probabilistic models to measure distances between clusters.

One way to look at the clustering problem is to regard the set of data objects to beclustered as a sample of the underlying data generation mechanism to be analyzed or,formally, the *generative model*. For example, when we conduct clustering analysis ona set of marketing surveys, we assume that the surveys collected are a sample of the opinions of all possible customers. Here, the data generation mechanism is a probability

distribution of opinions with respect to different customers, which cannot be obtained directly and completely. The task of clustering is to estimate the generative model as accurately as possible using the observed data objects to be clustered.

In practice, we can assume that the data generative models adopt common distri- bution functions, such as Gaussian distribution or Bernoulli distribution, which are governed by parameters. The task of learning a generative model is then reduced to finding the parameter values for which the model best fits the observed data set.

Example 10.6 **Generative model.** Suppose we are given a set of 1-D points $X = \{x_1, \ldots, x_n\}$ for clustering analysis. Let us assume that the data points are generated by a Gaussian distribution,

$$\mathcal{N}(\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{10.14}$$

where the parameters are μ (the mean) and σ^2 (the variance).

The probability that a point $x_i \in X$ is then generated by the model is

$$P(x_i|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \tag{10.15}$$

Consequently, the likelihood that X is generated by the model is

$$L(\mathcal{N}(\mu,\sigma^2) : X) = P(X|\mu,\sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \tag{10.16}$$

The task of learning the generative model is to find the parameters μ and σ^2 such that the likelihood $L(\mathcal{N}(\mu,\sigma^2) : X)$ is maximized, that is, finding

$$\mathcal{N}(\mu_0,\sigma_0^2) = \arg\max\{L(\mathcal{N}(\mu,\sigma^2) : X)\}, \tag{10.17}$$

where $\max\{L(\mathcal{N}(\mu,\sigma^2):X)\}$ is called the *maximum likelihood.* ■

Given a set of objects, the quality of a cluster formed by all the objects can be measured by the maximum likelihood. For a set of objects partitioned into m clusters $C_1,\ldots,C_m$, the quality can be measured by

$$Q(\{C_1,\ldots,C_m\}) = \prod_{i=1}^{m} P(C_i), \tag{10.18}$$

where $P()$ is the maximum likelihood. If we merge two clusters, C_{j_1} and C_{j_2}, into a cluster, $C_{j_1} \cup C_{j_2}$, then, the change in quality of the overall clustering is

$$\begin{aligned} &Q((\{C_1,\ldots,C_m\} - \{C_{j_1},C_{j_2}\}) \cup \{C_{j_1} \cup C_{j_2}\}) - Q(\{C_1,\ldots,C_m\}) \\ &= \frac{\prod_{i=1}^{m} P(C_i) \cdot P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})} - \prod_{i=1}^{m} P(C_i) \\ &= \prod_{i=1}^{m} P(C_i) \left(\frac{P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})} - 1 \right). \end{aligned} \tag{10.19}$$

When choosing to merge two clusters in hierarchical clustering, $\prod_{i=1}^{m} P(C_i)$ is constant for any pair of clusters. Therefore, given clusters C_1 and C_2, the distance between them can be measured by

$$dist(C_i,C_j) = -\log \frac{P(C_1 \cup C_2)}{P(C_1)P(C_2)}. \tag{10.20}$$

A probabilistic hierarchical clustering method can adopt the agglomerative clustering framework, but use probabilistic models (Eq. 10.20) to measure the distance between clusters.

Upon close observation of Eq. (10.19), we see that merging two clusters may not always lead to an improvement in clustering quality, that is, $\frac{P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})}$ may be less than 1. For example, assume that Gaussian distribution functions are used in the model of Figure 10.11. Although merging clusters C_1 and C_2 results in a cluster that better fits a Gaussian distribution, merging clusters C_3 and C_4 lowers the clustering quality because

no Gaussian functions can fit the merged cluster well.

Based on this observation, a probabilistic hierarchical clustering scheme can start with one cluster per object, and merge two clusters, C_i and C_j, if the distance between them is negative. In each iteration, we try to find C_i and C_j so as to maximize $\log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)}$. The iteration continues as long as $\log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)} > 0$, that is, as long as there is an improvement in clustering quality. The pseudocode is given in Figure 10.12.

Probabilistic hierarchical clustering methods are easy to understand, and generallyhave the same efficiency as algorithmic agglomerative hierarchical clustering methods;in fact, they share the same framework. Probabilistic models are more interpretable, but sometimes less flexible than distance metrics. Probabilistic models can handle partiallyobserved data. For example, given a multidimensional data set where some objects have missing values on some dimensions, we can learn a Gaussian model on each dimen- sion independently using the observed values on the dimension. The resulting clusterhierarchy accomplishes the optimization goal of fitting data to the selected probabilistic models.

A drawback of using probabilistic hierarchical clustering is that it outputs only one hierarchy with respect to a chosen probabilistic model. It cannot handle the uncer- tainty of cluster hierarchies. Given a data set, there may exist multiple hierarchies that

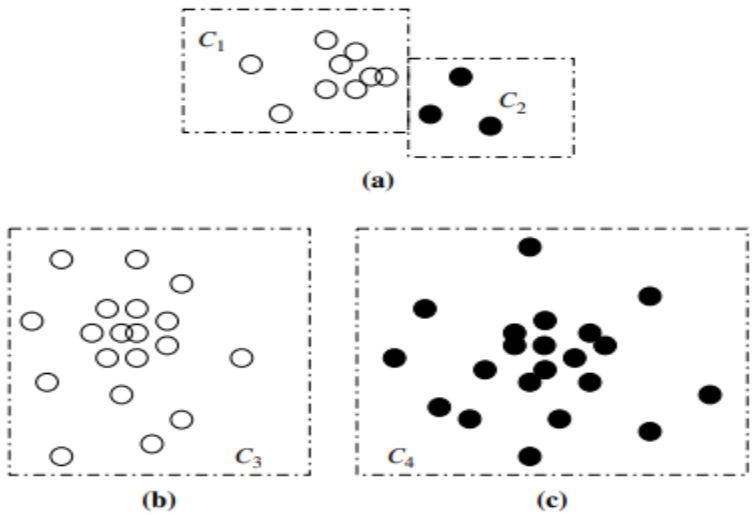


Figure 10.11 Merging clusters in probabilistic hierarchical clustering: (a) Merging clusters C_1 and C_2 leads to an increase in overall cluster quality, but merging clusters (b) C_3 and (c) C_4 does not.

Algorithm: A probabilistic hierarchical clustering algorithm.

Input:

- $D = \{\boldsymbol{o_1}, \ldots, \boldsymbol{o_n}\}$: a data set containing n objects;

Output: A hierarchy of clusters.

Method:

(1) **create** a cluster for each object $C_i = \{\boldsymbol{o_i}\}$, $1 \leq i \leq n$;
(2) **for** $i = 1$ to n
(3) **find** pair of clusters C_i and C_j such that $C_i, C_j = \arg\max_{i \neq j} \log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)}$;
(4) **if** $\log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)} > 0$ then merge C_i and C_j;
(5) **else** stop;

Figure 10.12 A probabilistic hierarchical clustering algorithm.

fit the observed data. Neither algorithmic approaches nor probabilistic approaches can find the distribution of such hierarchies. Recently, Bayesian tree-structured models have been developed to handle such problems. Bayesian and other sophisticated probabilistic clustering methods are considered advanced topics and are not covered in this book.

Density-Based Methods

Partitioning and hierarchical methods are designed to find spherical-shaped clusters. They have difficulty finding clusters of arbitrary shape such as the "S" shape and oval clusters in Figure 10.13. Given such data, they would likely inaccurately identify convexregions, where noise or outliers are included in the clusters.

To find clusters of arbitrary shape, alternatively, we can model clusters as dense regions in the data space, separated by sparse regions. This is the main strategy behind *density-based clustering methods*, which can discover clusters of nonspherical shape. In this section, you will learn the basic techniques of density-based clustering by studying three representative methods, namely, DBSCAN (Section 10.4.1), OPTICS (Section 10.4.2), and DENCLUE (Section 10.4.3).

DBSCAN: Density-Based Clustering Based on Connected Regions with High Density

"How can we find dense regions in density-based clustering?" The *density* of an object ***o*** can be measured by the number of objects close to ***o***. **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) finds *core objects*, that is, objects that have dense neighborhoods. It connects core objects and their neighborhoods to form dense regions as clusters.

"How does **DBSCAN** *quantify the neighborhood of an object?"* A user-specified para- meter < > 0 is used to specify the radius of a neighborhood we consider for every object. The <-**neighborhood** of an object ***o*** is the space within a radius < centered at ***o***.

Due to the fixed neighborhood size parameterized by <, the **density of a neighbor- hood** can be measured simply by the number of objects in the neighborhood. To deter- mine whether a neighborhood is dense or not, DBSCAN uses another user-specified

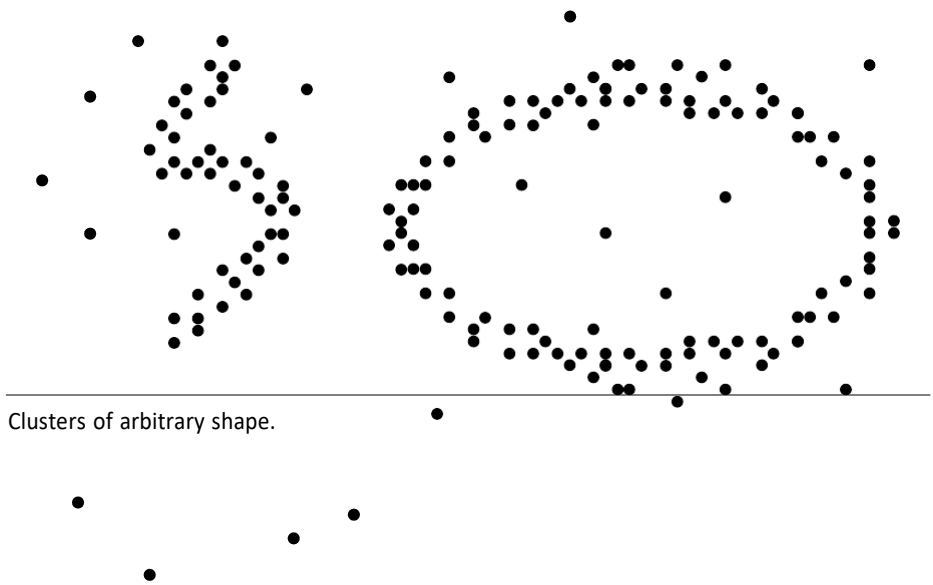


Figure 10.13 Clusters of arbitrary shape.

parameter, *MinPts*, which specifies the density threshold of dense regions. An object is a **core object** if the ε-neighborhood of the object contains at least *MinPts* objects. Core objects are the pillars of dense regions.

Given a set, *D*, of objects, we can identify all core objects with respect to the given parameters, ε and *MinPts*. The clustering task is therein reduced to using core objects and their neighborhoods to form dense regions, where the dense regions are clusters. For a core object ***q*** and an object ***p***, we say that ***p*** is **directly density-reachable** from ***q*** (with respect to ε and *MinPts*) if ***p*** is within the ε-neighborhood of ***q***. Clearly, an object ***p*** is directly density-reachable from another object ***q*** if and only if ***q*** is a core object and ***p*** is in the ε-neighborhood of ***q***. Using the directly density-reachable relation, a core object can "bring" all objects from its ε-neighborhood into a dense region.

"How can we assemble a large dense region using small dense regions centered by core objects?" In DBSCAN, ***p*** is **density-reachable** from ***q*** (with respect to ε and *MinPts* in *D*) if there is a chain of objects $p_1, \ldots, p_n$, such that $p_1 = q$, $p_n = p$, and p_{i+1} is directly density-reachable from p_i with respect to ε and *MinPts*, for $1 \le i \le n$, $p_i \in D$. Note that density-reachability is not an equivalence relation because it is not symmetric. If both o_1 and o_2 are core objects and o_1 is density-reachable from o_2, then o_2 is density-reachable from o_1. However, if o_2 is a core object but o_1 is not, then o_1 may be density-reachable from o_2, but not vice versa.

To connect core objects as well as their neighbors in a dense region, **DBSCAN** uses the notion of density-connectedness. Two objects $p_1, p_2 \in D$ are **density-connected** with respect to ε and *MinPts* if there is an object ***q*** *D* such that both p_1 and p_2 are density- reachable from ***q*** with respect to ε and *MinPts*. Unlike density-reachability, density- connectedness is an equivalence relation. It is easy to show that, for objects o_1, o_2, and o_3, if o_1 and o_2 are density-connected, and o_2 and o_3 are density-connected, then so are o_1 and o_3.

Example 10.7 Density-reachability and density-connectivity. Consider Figure 10.14 for a given ε represented by the radius of the circles, and, say, let *MinPts* = 3.

Of the labeled points, ***m***, ***p***, ***o***, ***r*** are core objects because each is in an ε-neighborhood containing at least three points. Object ***q*** is directly density-reachable from ***m***. Object ***m*** is directly density-reachable from ***p*** and vice versa.

Object ***q*** is (indirectly) density-reachable from ***p*** because ***q*** is directly density- reachable from ***m*** and ***m*** is directly density-reachable from ***p***. However, ***p*** is not density- reachable from ***q*** because ***q*** is not a core object. Similarly, ***r*** and ***s*** are density-reachable from ***o*** and ***o*** is density-reachable from ***r***. Thus, ***o***, ***r***, and ***s*** are all density-connected. ■

We can use the closure of density-connectedness to find connected dense regions as clusters. Each closed set is a **density-based cluster**. A subset $C \subseteq D$ is a cluster if (1) for any two objects $o_1, o_2 \in C$, o_1 and o_2 are density-connected; and (2) there does not exist an object $o \in C$ and another object $o' \in (D - C)$ such that ***o*** and o' are density- connected.

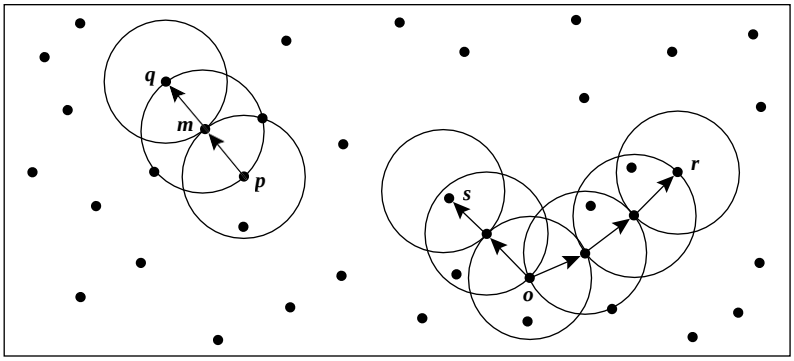


Figure 10.14 Density-reachability and density-connectivity in density-based clustering. *Source:* Based on Ester, Kriegel, Sander, and Xu [EKSX96].

"How does DBSCAN find clusters?" Initially, all objects in a given data set *D* are marked as "unvisited." DBSCAN randomly selects an unvisited object ***p***, marks ***p*** as "visited," and checks whether the <-neighborhood of ***p*** contains at least *MinPts* objects.If not, ***p*** is marked as a noise point. Otherwise, a new cluster *C* is created for ***p***, and allthe objects in the <-neighborhood of ***p*** are added to a candidate set, *N* . DBSCAN iter- atively adds to *C* those objects in *N* that do not belong to any cluster. In this process, for an object p^r in *N* that carries the label "unvisited," DBSCAN marks it as "visited" and
checks its <-neighborhood. If the <-neighborhood of p^r has at least *MinPts* objects, those
objects in the <-neighborhood of p^r are added to *N* . DBSCAN continues adding objectsto *C* until *C* can no longer be expanded, that is, *N* is empty. At this time, cluster *C* is
completed, and thus is output.

To find the next cluster, DBSCAN randomly selects an unvisited object from the remaining ones. The clustering process continues until all objects are visited. The pseudocode of the DBSCAN algorithm is given in Figure 10.15.

If a spatial index is used, the computational complexity of DBSCAN is $O(n \log n)$, where *n* is the number of database objects. Otherwise, the complexity is $O(n^2)$. With appropriate settings of the user-defined parameters, < and *MinPts*, the algorithm is effective in finding arbitrary-shaped clusters.

OPTICS: Ordering Points to Identifythe Clustering Structure

Although DBSCAN can cluster objects given input parameters such as < (the maxi- mum radius of a neighborhood) and *MinPts* (the minimum number of points requiredin the neighborhood of a core object), it encumbers users with the responsibility of selecting parameter values that will lead to the discovery of acceptable clusters. This is a problem associated with many other clustering algorithms. Such parameter settings

Algorithm: DBSCAN: a density-based clustering algorithm.

Input:

- *D*: a data set containing *n* objects,
- <: the radius parameter, and
- *MinPts*: the neighborhood density threshold.

Output: A set of density-based clusters.

Method:

```
(1)  mark all objects as unvisited;
(2)  do
(3)      randomly select an unvisited object p;
(4)      mark p as visited;
(5)      if the <-neighborhood of p has at least MinPts objects
(6)          create a new cluster C, and add p to C;
(7)          let N be the set of objects in the <-neighborhood of p;
(8)          for each point p^r in N
(9)              if p^r is unvisited
(10)                 mark p^r as visited;
(11)                 if the <-neighborhood of p^r has at least MinPts points,
                     add those points to N;
(12)             if p^r is not yet a member of any cluster, add p^r to C;
(13)         end for
(14)         output C;
(15)     else mark p as noise;
(16) until no object is unvisited;
```

Figure 10.15 DBSCAN algorithm.

are usually empirically set and difficult to determine, especially for real-world, high- dimensional data sets. Most algorithms are sensitive to these parameter values: Slightly different settings may lead to very different clusterings of the data. Moreover, real-world, high-dimensional data sets often have very skewed distributions such that their intrin-sic clustering structure may not be well characterized by a single set of *global* densityparameters.

Note that density-based clusters are monotonic with respect to the neighborhoodthreshold. That is, in DBSCAN, for a fixed *MinPts* value and two neighborhood thresh- olds, $<_1 < <_2$, a cluster *C* with respect to $<_1$ and *MinPts* must be a subset of a clusterC^r with respect to $<_2$ and *MinPts*. This means that if two objects are in a density-based cluster, they must also be in a cluster with a lower density requirement.

To overcome the difficulty in using one set of global parameters in clustering analy- sis, a cluster analysis method called **OPTICS** was proposed. OPTICS does not explicitlyproduce a data set clustering. Instead, it outputs a **cluster ordering**. This is a linear list

of all objects under analysis and represents the *density-based clustering structure* of the data. Objects in a denser cluster are listed closer to each other in the cluster ordering. This ordering is equivalent to density-based clustering obtained from a wide range of parameter settings. Thus, OPTICS does not require the user to provide a specific density threshold. The cluster ordering can be used to extract basic clustering information (e.g., cluster centers, or arbitrary-shaped clusters), derive the intrinsic clustering structure, as well as provide a visualization of the clustering.

To construct the different clusterings simultaneously, the objects are processed in a specific order. This order selects an object that is density-reachable with respect to the lowest ε value so that clusters with higher density (lower ε) will be finished first. Based on this idea, OPTICS needs two important pieces of information per object:

The **core-distance** of an object ***p*** is the smallest value ε' such that the ε'-neighborhood of ***p*** has at least *MinPts* objects. That is, ε' is the minimum distance threshold that makes ***p*** a core object. If ***p*** is not a core object with respect to ε and *MinPts*, the core-distance of ***p*** is undefined.

The **reachability-distance** to object ***p*** from ***q*** is the minimum radius value that makes ***p*** density-reachable from ***q***. According to the definition of density-reachability, ***q*** has to be a core object and ***p*** must be in the neighborhood of ***q***. Therefore, the reachability-distance from ***q*** to ***p*** is $\max$ *core-distance*(***q***), *dist*(***p***, ***q***) . If ***q*** is not a core object with respect to ε and *MinPts*, the reachability-distance to ***p*** from ***q*** is undefined.

An object ***p*** may be directly reachable from multiple core objects. Therefore, ***p*** may have multiple reachability-distances with respect to different core objects. The smallest reachability-distance of ***p*** is of particular interest because it gives the shortest path for which ***p*** is connected to a dense cluster.

Example 10.8 Core-distance and reachability-distance. Figure 10.16 illustrates the concepts of core-distance and reachability-distance. Suppose that ε 6 mm and *MinPts* 5. The core-distance of ***p*** is the distance, ε', between ***p*** and the fourth closest data object from ***p***. The reachability-distance of q_1 from ***p*** is the core-distance of ***p*** (i.e., ε' = 3 mm) because this is greater than the Euclidean distance from ***p*** to q_1. The reachability-distance of q_2 with respect to ***p*** is the Euclidean distance from ***p*** to q_2 because this is greater than the core-distance of ***p***.

■

OPTICS computes an ordering of all objects in a given database and, for each object in the database, stores the core-distance and a suitable reachability-distance. OPTICS maintains a list called OrderSeeds to generate the output ordering. Objects in Order-Seeds are sorted by the reachability-distance from their respective closest core objects, that is, by the smallest reachability-distance of each object.

OPTICS begins with an arbitrary object from the input database as the current object, ***p***. It retrieves the ε-neighborhood of ***p***, determines the core-distance, and sets the reachability-distance to *undefined*. The current object, ***p***, is then written to output.

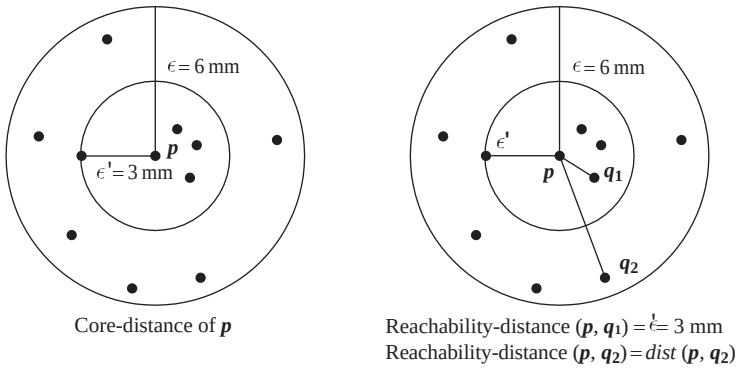


Figure 10.16 OPTICS terminology. *Source:* Based on Ankerst, Breunig, Kriegel, and Sander [ABKS99].

If ***p*** is not a core object, OPTICS simply moves on to the next object in the OrderSeedslist (or the input database if OrderSeeds is empty). If ***p*** is a core object, then for each object, ***q***, in the <-neighborhood of ***p***, OPTICS updates its reachability-distance from ***p*** and inserts ***q*** into OrderSeeds if ***q*** has not yet been processed. The iteration continuesuntil the input is fully consumed and OrderSeeds is empty.

A data set's cluster ordering can be represented graphically, which helps to visual-ize and understand the clustering structure in a data set. For example, Figure 10.17 isthe reachability plot for a simple 2-D data set, which presents a general overview of how the data are structured and clustered. The data objects are plotted in the cluster-ing order (horizontal axis) together with their respective reachability-distances (vertical axis). The three Gaussian "bumps" in the plot reflect three clusters in the data set. Meth- ods have also been developed for viewing clustering structures of high-dimensional data at various levels of detail.

The structure of the OPTICS algorithm is very similar to that of DBSCAN. Conse- quently, the two algorithms have the same time complexity. The complexity is $O(n \log n)$ if a spatial index is used, and $O(n^2)$ otherwise, where n is the number of objects.

DENCLUE: Clustering Based on Density Distribution Functions

Density estimation is a core issue in density-based clustering methods. **DENCLUE** (DENsity-based CLUstEring) is a clustering method based on a set of density distribu- tion functions. We first give some background on density estimation, and then describe the DENCLUE algorithm.

In probability and statistics, **density estimation** is the estimation of an unobservable underlying probability density function based on a set of observed data. In the contextof density-based clustering, the unobservable underlying probability density functionis the true distribution of the population of all possible objects to be analyzed. The observed data set is regarded as a random sample from that population.

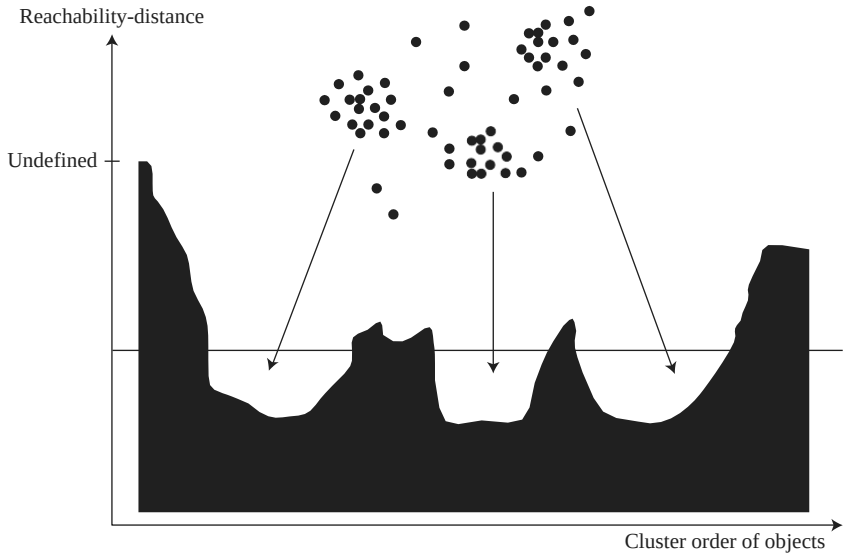


Figure 10.17 Cluster ordering in OPTICS. *Source:* Adapted from Ankerst, Breunig, Kriegel, and Sander [ABKS99].

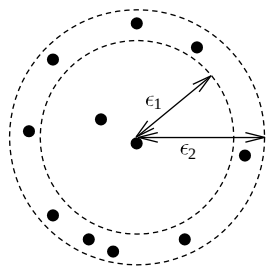


Figure 10.18 The subtlety in density estimation in DBSCAN and OPTICS: Increasing the neighborhood radius slightly from $<_1$ to $<_2$ results in a much higher density.

In DBSCAN and OPTICS, density is calculated by counting the number of objects in a neighborhood defined by a radius parameter, <. Such density estimates can be highly sensitive to the radius value used. For example, in Figure 10.18, the density changes significantly as the radius increases by a small amount.

To overcome this problem, **kernel density estimation** can be used, which is a nonparametric density estimation approach from statistics. The general idea behind kernel density estimation is simple. We treat an observed object as an indicator of

high-probability density in the surrounding region. The probability density at a point depends on the distances from this point to the observed objects.

Formally, let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ be an independent and identically distributed sample of a random variable f. The *kernel density approximation of the probability density function* is

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right), \tag{10.21}$$

where $K()$ is a kernel and h is the bandwidth serving as a smoothing parameter. A **kernel** can be regarded as a function modeling the influence of a sample point within its neighborhood. Technically, a kernel $K()$ is a non-negative real-valued integrable function that should satisfy two requirements: $\int_{-\infty}^{+\infty} K(u)du = 1$ and $K(-u) = K(u)$ for all values of u. A frequently used kernel is a standard Gaussian function with a mean of 0 and a variance of 1:

$$K\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right) = \frac{1}{\sqrt{2\Pi}} e^{-\frac{(\boldsymbol{x} - \boldsymbol{x}_i)^2}{2h^2}}. \tag{10.22}$$

DENCLUE uses a Gaussian kernel to estimate density based on the given set of objects to be clustered. A point $\boldsymbol{x}^*$ is called a **density attractor** if it is a local maximum of the estimated density function. To avoid trivial local maximum points, DENCLUE uses a noise threshold, ξ, and only considers those density attractors $\boldsymbol{x}^*$ such that $\hat{f}(\boldsymbol{x}^*) \geq \xi$. These nontrivial density attractors are the centers of clusters.

Objects under analysis are assigned to clusters through density attractors using a stepwise hill-climbing procedure. For an object, $\boldsymbol{x}$, the hill-climbing procedure starts from $\boldsymbol{x}$ and is guided by the gradient of the estimated density function. That is, the density attractor for $\boldsymbol{x}$ is computed as

$$\boldsymbol{x}^0 = \boldsymbol{x}$$
$$\boldsymbol{x}^{j+1} = \boldsymbol{x}^j + \delta \frac{\nabla \hat{f}(\boldsymbol{x}^j)}{|\nabla \hat{f}(\boldsymbol{x}^j)|}, \tag{10.23}$$

where δ is a parameter to control the speed of convergence, and

$$\nabla \hat{f}(\boldsymbol{x}) = \frac{1}{h^{d+2} n \sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right)(\boldsymbol{x}_i - \boldsymbol{x})}. \tag{10.24}$$

The hill-climbing procedure stops at step $k > 0$ if $\hat{f}(\boldsymbol{x}^{k+1}) < \hat{f}(\boldsymbol{x}^k)$, and assigns $\boldsymbol{x}$ to the density attractor $\boldsymbol{x}^* = \boldsymbol{x}^k$. An object $\boldsymbol{x}$ is an outlier or noise if it converges in the hill-climbing procedure to a local maximum $\boldsymbol{x}^*$ with $\hat{f}(\boldsymbol{x}^*) < \xi$.

A cluster in DENCLUE is a set of density attractors X and a set of input objects C such that each object in C is assigned to a density attractor in X, and there exists a path between every pair of density attractors where the density is above ξ. By using multiple density attractors connected by paths, DENCLUE can find clusters of arbitrary shape.

DENCLUE has several advantages. It can be regarded as a generalization of several well-known clustering methods such as single-linkage approaches and DBSCAN. Moreover, DENCLUE is invariant against noise. The kernel density estimation can effectively reduce the influence of noise by uniformly distributing noise into the input data.

Grid-Based Methods

The clustering methods discussed so far are data-driven—they partition the set of objects and adapt to the distribution of the objects in the embedding space. Alternatively, a **grid-based clustering** method takes a space-driven approach by partitioning the embedding space into *cells* independent of the distribution of the input objects.

The *grid-based clustering* approach uses a multiresolution grid data structure. It quantizes the object space into a finite number of cells that form a grid structure on which all of the operations for clustering are performed. The main advantage of the approach is its fast processing time, which is typically independent of the number of data objects, yet dependent on only the number of cells in each dimension in the quantized space.

In this section, we illustrate grid-based clustering using two typical examples. STING (Section 10.5.1) explores statistical information stored in the grid cells. CLIQUE (Section 10.5.2) represents a grid- and density-based approach for subspace clustering in a high-dimensional data space.

STING: STatistical INformation Grid

STING is a grid-based multiresolution clustering technique in which the embedding spatial area of the input objects is divided into rectangular cells. The space can be divided in a hierarchical and recursive way. Several levels of such rectangular cells correspond to different levels of resolution and form a hierarchical structure: Each cell at a high level is partitioned to form a number of cells at the next lower level. Statistical information regarding the attributes in each grid cell, such as the mean, maximum, and minimum values, is precomputed and stored as *statistical parameters*. These statistical parameters are useful for query processing and for other data analysis tasks.

Figure 10.19 shows a hierarchical structure for STING clustering. The statistical parameters of higher-level cells can easily be computed from the parameters of the lower-level cells. These parameters include the following: the attribute-independent parameter, *count*; and the attribute-dependent parameters, *mean*, *stdev* (standard deviation), *min* (minimum), *max* (maximum), and the type of *distribution* that the attribute value in the cell follows such as *normal, uniform*, *exponential*, or *none* (if the distribution is unknown). Here, the attribute is a selected measure for analysis such as *price* for house objects. When the data are loaded into the database, the parameters *count*, *mean*, *stdev*, *min*, and *max* of the bottom-level cells are calculated directly from the data. The value of *distribution* may either be assigned by the user if the distribution type is known

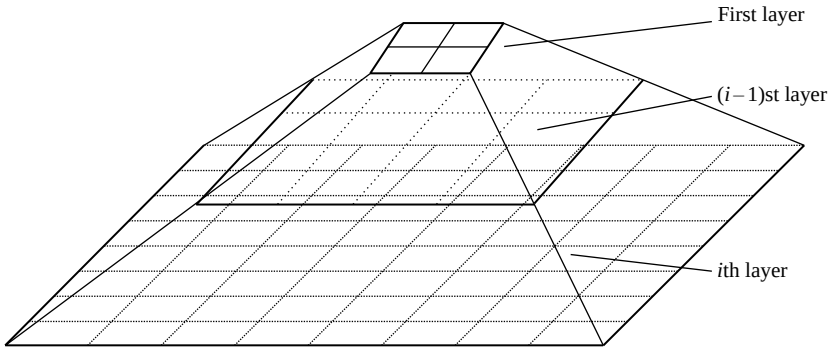


Figure 10.19 Hierarchical structure for STING clustering.

beforehand or obtained by hypothesis tests such as the χ^2 test. The type of distribution of a higher-level cell can be computed based on the majority of distribution types of its corresponding lower-level cells in conjunction with a threshold filtering process. If the distributions of the lower-level cells disagree with each other and fail the threshold test, the distribution type of the high-level cell is set to *none*.

"How is this statistical information useful for query answering?" The statistical parameters can be used in a top-down, grid-based manner as follows. First, a layer within the hierarchical structure is determined from which the query-answering process is to start. This layer typically contains a small number of cells. For each cell in the current layer, we compute the confidence interval (or estimated probability range) reflecting the cell's relevancy to the given query. The irrelevant cells are removed from further consideration. Processing of the next lower level examines only the remaining relevant cells. This process is repeated until the bottom layer is reached. At this time, if the query specification is met, the regions of relevant cells that satisfy the query are returned. Otherwise, the data that fall into the relevant cells are retrieved and further processed until they meet the query's requirements.

An interesting property of STING is that it approaches the clustering result of DBSCAN if the granularity approaches 0 (i.e., toward very low-level data). In other words, using the count and cell size information, dense clusters can be identified approximately using STING. Therefore, STING can also be regarded as a density-based clustering method.

"What advantages does STING offer over other clustering methods?" STING offers several advantages: (1) the grid-based computation is *query-independent* because the statistical information stored in each cell represents the summary information of the data in the grid cell, independent of the query; (2) the grid structure facilitates parallel processing and incremental updating; and (3) the method's efficiency is a major advantage: STING goes through the database once to compute the statistical parameters of the cells, and hence the time complexity of generating clusters is $O(n)$, where n is the total number of objects. After generating the hierarchical structure, the query processing time

is $O(g)$, where g is the total number of grid cells at the lowest level, which is usually much smaller than n.

Because STING uses a multiresolution approach to cluster analysis, the quality of STING clustering depends on the granularity of the lowest level of the grid structure. If the granularity is very fine, the cost of processing will increase substantially; however, if the bottom level of the grid structure is too coarse, it may reduce the quality of cluster analysis. Moreover, STING does not consider the spatial relationship between the children and their neighboring cells for construction of a parent cell. As a result, the shapes of the resulting clusters are isothetic, that is, all the cluster boundaries are either horizontal or vertical, and no diagonal boundary is detected. This may lower the quality and accuracy of the clusters despite the fast processing time of the technique.

CLIQUE: An Apriori-like Subspace Clustering Method

A data object often has tens of attributes, many of which may be irrelevant. The values of attributes may vary considerably. These factors can make it difficult to locate clusters that span the entire data space. It may be more meaningful to instead search for clusters within different *subspaces* of the data. For example, consider a health-informatics application where patient records contain extensive attributes describing personal information, numerous symptoms, conditions, and family history.

Finding a nontrivial group of patients for which all or even most of the attributes strongly agree is unlikely. In bird flu patients, for instance, the *age*, *gender*, and *job* attributes may vary dramatically within a wide range of values. Thus, it can be difficult to find such a cluster within the entire data space. Instead, by searching in subspaces, we may find a cluster of similar patients in a lower-dimensional space (e.g., patients who are similar to one other with respect to symptoms like high fever, cough but no runny nose, and aged between 3 and 16).

CLIQUE (CLustering In QUEst) is a simple grid-based method for finding density-based clusters in subspaces. CLIQUE partitions each dimension into nonoverlapping intervals, thereby partitioning the entire embedding space of the data objects into cells. It uses a density threshold to identify *dense* cells and *sparse* ones. A cell is dense if the number of objects mapped to it exceeds the density threshold.

The main strategy behind CLIQUE for identifying a candidate search space uses the monotonicity of dense cells with respect to dimensionality. This is based on the *Apriori property* used in frequent pattern and association rule mining (Chapter 6). In the context of clusters in subspaces, the monotonicity says the following. A k-dimensional cell c $(k > 1)$ can have at least l points only if every $(k-1)$-dimensional projection of c, which is a cell in a $(k-1)$-dimensional subspace, has at least l points. Consider Figure 10.20, where the embedding data space contains three dimensions: *age, salary*, and *vacation*. A 2-D cell, say in the subspace formed by *age* and *salary*, contains l points only if the projection of this cell in every dimension, that is, *age* and *salary*, respectively, contains at least l points.

CLIQUE performs clustering in two steps. In the first step, CLIQUE partitions the d-dimensional data space into nonoverlapping rectangular units, identifying the dense units among these. CLIQUE finds dense cells in all of the subspaces. To do so,

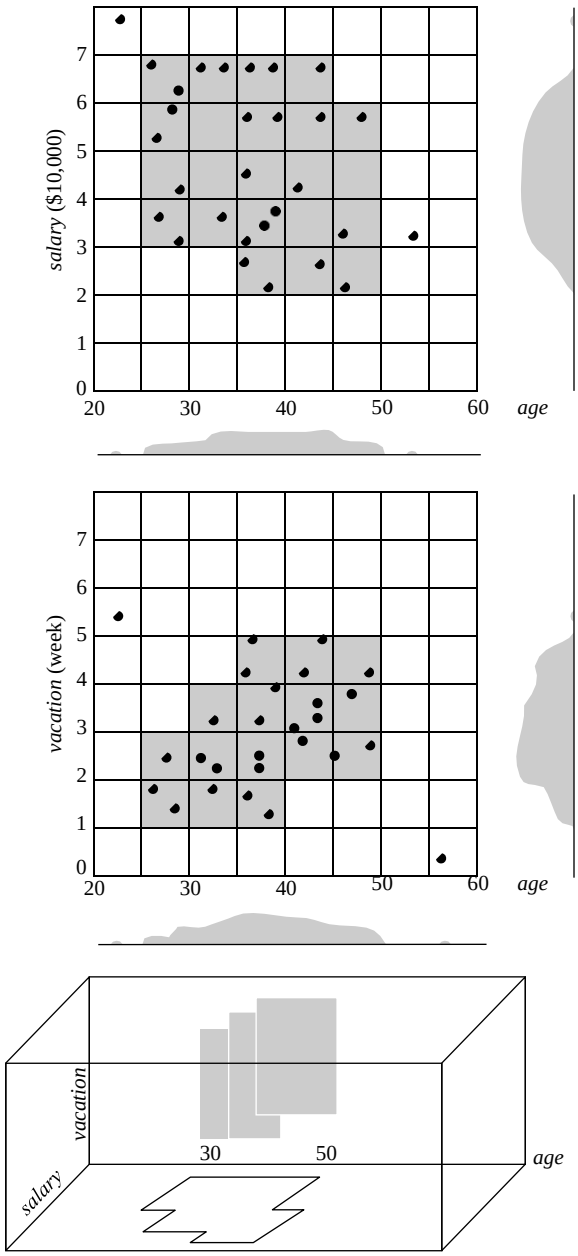


Figure 10.20 Dense units found with respect to *age* for the dimensions *salary* and *vacation* are intersected to provide a candidate search space for dense units of higher dimensionality.

CLIQUE partitions every dimension into intervals, and identifies intervals containing at least l points, where l is the density threshold. CLIQUE then iteratively joins two k-dimensional dense cells, c_1 and c_2, in subspaces $(D_{i_1}, \ldots, D_{i_k})$ and $(D_{j_1}, \ldots, D_{j_k})$, respectively, if $D_{i_1} = D_{j_1}, \ldots, D_{i_{k-1}} = D_{j_{k-1}}$, and c_1 and c_2 share the same intervals in those dimensions. The join operation generates a new $(k+1)$-dimensional candidate cell c in space $(D_{i_1}, \ldots, D_{i_{k-1}}, D_{i_k}, D_{j_k})$. CLIQUE checks whether the number of points in c passes the density threshold. The iteration terminates when no candidates can be generated or no candidate cells are dense.

In the second step, CLIQUE uses the dense cells in each subspace to assemble clusters, which can be of arbitrary shape. The idea is to apply the Minimum Description Length (MDL) principle (Chapter 8) to use the *maximal regions* to cover connected dense cells, where a maximal region is a hyperrectangle where every cell falling into this region is dense, and the region cannot be extended further in any dimension in the subspace. Finding the best description of a cluster in general is NP-Hard. Thus, CLIQUE adopts a simple greedy approach. It starts with an arbitrary dense cell, finds a maximal region covering the cell, and then works on the remaining dense cells that have not yet been covered. The greedy method terminates when all dense cells are covered.

"How effective is CLIQUE?" CLIQUE automatically finds subspaces of the highest dimensionality such that high-density clusters exist in those subspaces. It is insensitive to the order of input objects and does not presume any canonical data distribution. It scales linearly with the size of the input and has good scalability as the number of dimensions in the data is increased. However, obtaining a meaningful clustering is dependent on proper tuning of the grid size (which is a stable structure here) and the density threshold. This can be difficult in practice because the grid size and density threshold are used across all combinations of dimensions in the data set. Thus, the accuracy of the clustering results may be degraded at the expense of the method's simplicity. Moreover, for a given dense region, all projections of the region onto lower-dimensionality subspaces will also be dense. This can result in a large overlap among the reported dense regions. Furthermore, it is difficult to find clusters of rather different densities within different dimensional subspaces.

Several extensions to this approach follow a similar philosophy. For example, we can think of a grid as a set of fixed bins. Instead of using fixed bins for each of the dimensions, we can use an adaptive, data-driven strategy to dynamically determine the bins for each dimension based on data distribution statistics. Alternatively, instead of using a density threshold, we may use entropy (Chapter 8) as a measure of the quality of subspace clusters.

Evaluation of Clustering

By now you have learned what clustering is and know several popular clustering methods. You may ask, "*When I try out a clustering method on a data set, how can I evaluate whether the clustering results are good?*" In general, *cluster evaluation* assesses

the feasibility of clustering analysis on a data set and the quality of the results generated by a clustering method. The major tasks of clustering evaluation include the following:

- *Assessing clustering tendency*. In this task, for a given data set, we assess whether a nonrandom structure exists in the data. Blindly applying a clustering method on a data set will return clusters; however, the clusters mined may be misleading. Clustering analysis on a data set is meaningful only when there is a nonrandom structure in the data.
- *Determining the number of clusters in a data set*. A few algorithms, such as *k*-means, require the number of clusters in a data set as the parameter. Moreover, the number of clusters can be regarded as an interesting and important summary statistic of a data set. Therefore, it is desirable to estimate this number even before a clustering algorithm is used to derive detailed clusters.
- *Measuring clustering quality*. After applying a clustering method on a data set, we want to assess how good the resulting clusters are. A number of measures can be used. Some methods measure how well the clusters fit the data set, while others measure how well the clusters match the ground truth, if such truth is available. There are also measures that score clusterings and thus can compare two sets of clustering results on the same data set.

In the rest of this section, we discuss each of these three topics.

Assessing Clustering Tendency

Clustering tendency assessment determines whether a given data set has a non-random structure, which may lead to meaningful clusters. Consider a data set that does not have any non-random structure, such as a set of uniformly distributed points in a data space. Even though a clustering algorithm may return clusters for the data, those clusters are random and are not meaningful.

Example 10.9 **Clustering requires nonuniform distribution of data.** Figure 10.21 shows a data set that is uniformly distributed in 2-D data space. Although a clustering algorithm may still artificially partition the points into groups, the groups will unlikely mean anything significant to the application due to the uniform distribution of the data. ■

"How can we assess the clustering tendency of a data set?" Intuitively, we can try to measure the probability that the data set is generated by a uniform data distribution. This can be achieved using statistical tests for spatial randomness. To illustrate this idea, let's look at a simple yet effective statistic called the Hopkins Statistic.

The **Hopkins Statistic** is a spatial statistic that tests the spatial randomness of a variable as distributed in a space. Given a data set, *D*, which is regarded as a sample of

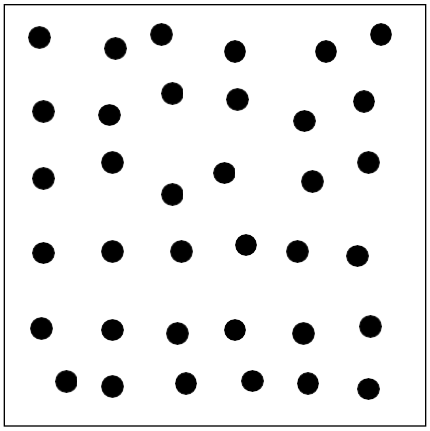


Figure 10.21 A data set that is uniformly distributed in the data space.

a random variable, o, we want to determine how far away o is from being uniformly distributed in the data space. We calculate the Hopkins Statistic as follows:

1. Sample n points, $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$, uniformly from D. That is, each point in D has the same probability of being included in this sample. For each point, $\boldsymbol{p}_i$, we find the nearest neighbor of $\boldsymbol{p}_i$ $(1 \leq i \leq n)$ in D, and let x_i be the distance between $\boldsymbol{p}_i$ and its nearest neighbor in D. That is,

$$x_i = \min_{\boldsymbol{v} \in D} \{dist(\boldsymbol{p}_i, \boldsymbol{v})\}. \tag{10.25}$$

2. Sample n points, $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$, uniformly from D. For each $\boldsymbol{q}_i$ $(1 \leq i \leq n)$, we find the nearest neighbor of $\boldsymbol{q}_i$ in $D - \{\boldsymbol{q}_i\}$, and let y_i be the distance between $\boldsymbol{q}_i$ and its nearest neighbor in $D - \{\boldsymbol{q}_i\}$. That is,

$$y_i = \min_{\boldsymbol{v} \in D, \boldsymbol{v} \neq \boldsymbol{q}_i} \{dist(\boldsymbol{q}_i, \boldsymbol{v})\}. \tag{10.26}$$

3. Calculate the Hopkins Statistic, H, as

$$H = \frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i}. \tag{10.27}$$

"What does the Hopkins Statistic tell us about how likely data set D *follows a uniform distribution in the data space?"* If D were uniformly distributed, then $\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n}$

highly skewed, then $\sum_{i=1}^{n} y_i$ would be substantially smaller than $\sum_{i=1}^{n} x_i$ in expectation,

$\sum_{i=1}^{n} x_i$ would be close to each other, and thus H would be about 0.5. However, if D were and thus H would be close to 0.

Our null hypothesis is the *homogeneous hypothesis*—that D is uniformly distributed and thus contains no meaningful clusters. The *nonhomogeneous hypothesis* (i.e., that D is not uniformly distributed and thus contains clusters) is the alternative hypothesis. We can conduct the Hopkins Statistic test iteratively, using 0.5 as the threshold to reject the alternative hypothesis. That is, if $H > 0.5$, then it is unlikely that D has statistically significant clusters.

Determining the Number of Clusters

Determining the "right" number of clusters in a data set is important, not only because some clustering algorithms like *k*-means require such a parameter, but also because the appropriate number of clusters controls the proper granularity of cluster analysis. It can be regarded as finding a good balance between *compressibility* and *accuracy* in cluster analysis. Consider two extreme cases. What if you were to treat the entire data set as a cluster? This would maximize the compression of the data, but such a cluster analysis has no value. On the other hand, treating each object in a data set as a cluster gives the finest clustering resolution (i.e., most accurate due to the zero distance between an object and the corresponding cluster center). In some methods like *k*-means, this even achieves the best cost. However, having one object per cluster does not enable any data summarization.

Determining the number of clusters is far from easy, often because the "right" number is ambiguous. Figuring out what the right number of clusters should be often depends on the distribution's shape and scale in the data set, as well as the clustering resolution required by the user. There are many possible ways to estimate the number of clusters. Here, we briefly introduce a few simple yet popular and effective methods.

A simple method is to set the number of clusters to about $\sqrt{\frac{n}{2}}$ for a data set of n points. In expectation, each cluster has $\sqrt{2n}$ points.

The **elbow method** is based on the observation that increasing the number of clusters can help to reduce the sum of within-cluster variance of each cluster. This is because having more clusters allows one to capture finer groups of data objects that are more similar to each other. However, the marginal effect of reducing the sum of within-cluster variances may drop if too many clusters are formed, because splitting a cohesive cluster into two gives only a small reduction. Consequently, a heuristic for selecting the right number of clusters is to use the turning point in the curve of the sum of within-cluster variances with respect to the number of clusters.

Technically, given a number, $k > 0$, we can form k clusters on the data set in question using a clustering algorithm like *k*-means, and calculate the sum of within-cluster variances, $var(k)$. We can then plot the curve of *var* with respect to k. The first (or most significant) turning point of the curve suggests the "right" number.

More advanced methods can determine the number of clusters using information criteria or information theoretic approaches. Please refer to the bibliographic notes for further information (Section 10.9).

The "right" number of clusters in a data set can also be determined by **cross-validation**, a technique often used in classification (Chapter 8). First, divide the given data set, D, into m parts. Next, use $m - 1$ parts to build a clustering model, and use the remaining part to test the quality of the clustering. For example, for each point in the test set, we can find the closest centroid. Consequently, we can use the sum of the squared distances between all points in the test set and the closest centroids to measure how well the clustering model fits the test set. For any integer $k > 0$, we repeat this process m times to derive clusterings of k clusters by using each part in turn as the test set. The average of the quality measure is taken as the overall quality measure. We can then compare the overall quality measure with respect to different values of k, and find the number of clusters that best fits the data.

Measuring Clustering Quality

Suppose you have assessed the clustering tendency of a given data set. You may have also tried to predetermine the number of clusters in the set. You can now apply one or multiple clustering methods to obtain clusterings of the data set. "*How good is the clustering generated by a method, and how can we compare the clusterings generated by different methods?*"

We have a few methods to choose from for measuring the quality of a clustering. In general, these methods can be categorized into two groups according to whether ground truth is available. Here, *ground truth* is the ideal clustering that is often built using human experts.

If ground truth is available, it can be used by **extrinsic methods**, which compare the clustering against the group truth and measure. If the ground truth is unavailable, we can use **intrinsic methods**, which evaluate the goodness of a clustering by considering how well the clusters are separated. Ground truth can be considered as supervision in the form of "cluster labels." Hence, extrinsic methods are also known as *supervised methods*, while intrinsic methods are *unsupervised methods*.

Let's have a look at simple methods from each category.

Extrinsic Methods

When the ground truth is available, we can compare it with a clustering to assess the clustering. Thus, the core task in extrinsic methods is to assign a score, $Q(C, C_g)$, to a clustering, C, given the ground truth, C_g. Whether an extrinsic method is effective largely depends on the measure, Q, it uses.

In general, a measure Q on clustering quality is effective if it satisfies the following four essential criteria:

- **Cluster homogeneity.** This requires that the more pure the clusters in a clustering are, the better the clustering. Suppose that ground truth says that the objects in a data set, D, can belong to categories $L_1, \ldots, L_n$. Consider clustering, C_1, wherein a cluster $C \in C_1$ contains objects from two categories L_i, L_j $(1 \leq i < j \leq n)$. Also

consider clustering C_2, which is identical to C_1 except that C_2 is split into two clusters containing the objects in L_i and L_j, respectively. A clustering quality measure, Q, respecting cluster homogeneity should give a higher score to C_2 than C_1, that is, $Q(C_2, C_g) > Q(C_1, C_g)$.

- **Cluster completeness.** This is the counterpart of cluster homogeneity. Cluster completeness requires that for a clustering, if any two objects belong to the same category according to ground truth, then they should be assigned to the same cluster. Cluster completeness requires that a clustering should assign objects belonging to the same category (according to ground truth) to the same cluster. Consider clustering C_1, which contains clusters C_1 and C_2, of which the members belong to the same category according to ground truth. Let clustering C_2 be identical to C_1 except that C_1 and C_2 are merged into one cluster in C_2. Then, a clustering quality measure, Q, respecting cluster completeness should give a higher score to C_2, that is, $Q(C_2, C_g) > Q(C_1, C_g)$.
- **Rag bag.** In many practical scenarios, there is often a "rag bag" category containing objects that cannot be merged with other objects. Such a category is often called "miscellaneous," "other," and so on. The rag bag criterion states that putting a heterogeneous object into a pure cluster should be penalized more than putting it into a rag bag. Consider a clustering C_1 and a cluster $C \in C_1$ such that all objects in C except for one, denoted by $\boldsymbol{o}$, belong to the same category according to ground truth. Consider a clustering C_2 identical to C_1 except that $\boldsymbol{o}$ is assigned to a cluster $C' \neq C$ in C_2 such that C' contains objects from various categories according to ground truth, and thus is noisy. In other words, C' in C_2 is a rag bag. Then, a clustering quality measure Q respecting the rag bag criterion should give a higher score to C_2, that is, $Q(C_2, C_g) > Q(C_1, C_g)$.
- **Small cluster preservation.** If a small category is split into small pieces in a clustering, those small pieces may likely become noise and thus the small category cannot be discovered from the clustering. The small cluster preservation criterion states that splitting a small category into pieces is more harmful than splitting a large category into pieces. Consider an extreme case. Let D be a data set of $n + 2$ objects such that, according to ground truth, n objects, denoted by $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n$, belong to one category and the other two objects, denoted by $\boldsymbol{o}_{n+1}, \boldsymbol{o}_{n+2}$, belong to another category. Suppose clustering C_1 has three clusters, $C_1 = \{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n\}$, $C_2 = \{\boldsymbol{o}_{n+1}\}$, and $C_3 = \{\boldsymbol{o}_{n+2}\}$. Let clustering C_2 have three clusters, too, namely $C_1 = \{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_{n-1}\}$, $C_2 = \{\boldsymbol{o}_n\}$, and $C_3 = \{\boldsymbol{o}_{n+1}, \boldsymbol{o}_{n+2}\}$. In other words, C_1 splits the small category and C_2 splits the big category. A clustering quality measure Q preserving small clusters should give a higher score to C_2, that is, $Q(C_2, C_g) > Q(C_1, C_g)$.

Many clustering quality measures satisfy some of these four criteria. Here, we introduce the *BCubed precision* and *recall* metrics, which satisfy all four criteria.

BCubed evaluates the precision and recall for every object in a clustering on a given data set according to ground truth. The precision of an object indicates how many other objects in the same cluster belong to the same category as the object. The recall

of an object reflects how many objects of the same category are assigned to the same cluster.

Formally, let $D = \{o_1, \ldots, o_n\}$ be a set of objects, and C be a clustering on D. Let $L(o_i)$ $(1 \le i \le n)$ be the category of o_i given by ground truth, and $C(o_i)$ be the *cluster ID* of o_i in C. Then, for two objects, o_i and o_j, $(1 \le i, j, \le n, i \neq j)$, the *correctness* of the relation between o_i and o_j in clustering C is given by

$$\text{Correctness}(o_i, o_j) \begin{cases} 1 & \text{if } L(o_i) = L(o_j) \Leftrightarrow C(o_i) = C(o_j) \\ 0 & \text{otherwise.} \end{cases} \tag{10.28}$$

BCubed precision is defined as

$$\text{Precision BCubed} = \frac{\sum_{i=1}^{n} \dfrac{\sum_{o_j : i \neq j, C(o_i) = C(o_j)} \text{Correctness}(o_i, o_j)}{\|\{o_j \mid i \neq j, C(o_i) = C(o_j)\}\|}}{n}. \tag{10.29}$$

BCubed recall is defined as

$$\text{Recall BCubed} = \frac{\sum_{i=1}^{n} \dfrac{\sum_{o_j : i \neq j, L(o_i) = L(o_j)} \text{Correctness}(o_i, o_j)}{\|\{o_j \mid i \neq j, L(o_i) = L(o_j)\}\|}}{n}. \tag{10.30}$$

Intrinsic Methods

When the ground truth of a data set is not available, we have to use an intrinsic method to assess the clustering quality. In general, intrinsic methods evaluate a clustering by examining how well the clusters are separated and how compact the clusters are. Many intrinsic methods have the advantage of a similarity metric between objects in the data set.

The **silhouette coefficient** is such a measure. For a data set, D, of n objects, suppose D is partitioned into k clusters, $C_1, \ldots, C_k$. For each object $o \in D$, we calculate $a(o)$ as the average distance between o and all other objects in the cluster to which o belongs. Similarly, $b(o)$ is the minimum average distance from o to all clusters to which o does not belong. Formally, suppose $o \in C_i$ $(1 \le i \le k)$; then

$$a(o) = \frac{\sum_{o' \in C_i, o \neq o'} dist(o, o')}{|C_i| - 1} \tag{10.31}$$

and

$$b(\boldsymbol{o}) = \min_{C_j: 1 \le j \le k, j \ne i} \frac{\left(\sum_{\boldsymbol{o}' \in C_j} dist(\boldsymbol{o}, \boldsymbol{o}')\right)}{|C_j|}. \tag{10.32}$$

The **silhouette coefficient** of $\boldsymbol{o}$ is then defined as

$$s(\boldsymbol{o}) = \frac{b(\boldsymbol{o}) - a(\boldsymbol{o})}{\max\{a(\boldsymbol{o}), b(\boldsymbol{o})\}}. \tag{10.33}$$

The value of the silhouette coefficient is between 1 and 1. The value of $a(\boldsymbol{o})$ reflects the compactness of the cluster to which $\boldsymbol{o}$ belongs. The smaller the value, the more compact the cluster. The value of $b(\boldsymbol{o})$ captures the degree to which $\boldsymbol{o}$ is separated from other clusters. The larger $b(\boldsymbol{o})$ is, the more separated $\boldsymbol{o}$ is from other clusters. Therefore, when the silhouette coefficient value of $\boldsymbol{o}$ approaches 1, the cluster containing $\boldsymbol{o}$ is compact and $\boldsymbol{o}$ is far away from other clusters, which is the preferable case. However, when the silhouette coefficient value is negative (i.e., $b(\boldsymbol{o}) < a(\boldsymbol{o})$), this means that, in expectation, $\boldsymbol{o}$ is closer to the objects in another cluster than to the objects in the same cluster as $\boldsymbol{o}$. In many cases, this is a bad situation and should be avoided.

To measure a cluster's fitness within a clustering, we can compute the average silhouette coefficient value of all objects in the cluster. To measure the quality of a clustering, we can use the average silhouette coefficient value of all objects in the data set. The silhouette coefficient and other intrinsic measures can also be used in the elbow method to heuristically derive the number of clusters in a data set by replacing the sum of within-cluster variances.

Clustering High-Dimensional Data

The clustering methods we have studied so far work well when the dimensionality is not high, that is, having less than 10 attributes. There are, however, important applications of high dimensionality. *"How can we conduct cluster analysis on high-dimensional data?"*

In this section, we study approaches to clustering high-dimensional data. Section 11.2.1 starts with an overview of the major challenges and the approaches used. Methods for high-dimensional data clustering can be divided into two categories: subspace clustering methods (Section 11.2.2) and dimensionality reduction methods (Section 11.2.3).

Clustering High-Dimensional Data: Problems, Challenges, and Major Methodologies

Before we present any specific methods for clustering high-dimensional data, let's first demonstrate the needs of cluster analysis on high-dimensional data using examples. We examine the challenges that call for new methods. We then categorize the major methods according to whether they search for clusters in subspaces of the original space, or whether they create a new lower-dimensionality space and search for clusters there.

In some applications, a data object may be described by 10 or more attributes. Such objects are referred to as a high-dimensional data space.

Example 11.9 **High-dimensional data and clustering.** *AllElectronics* keeps track of the products purchased by every customer. As a customer-relationship manager, you want to cluster customers into groups according to what they purchased from *AllElectronics*.

Table 11.4 Customer Purchase Data

Customer	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
Ada	1	0	0	0	0	0	0	0	0	0
Bob	0	0	0	0	0	0	0	0	0	1
Cathy	1	0	0	0	1	0	0	0	0	1

The customer purchase data are of very high dimensionality. *AllElectronics* carries tens of thousands of products. Therefore, a customer's purchase profile, which is a vector of the products carried by the company, has tens of thousands of dimensions.

"Are the traditional distance measures, which are frequently used in low-dimensional cluster analysis, also effective on high-dimensional data?" Consider the customers in Table 11.4, where 10 products, $P_1, \ldots, P_{10}$, are used in demonstration. If a customer purchases a product, a 1 is set at the corresponding bit; otherwise, a 0 appears. Let's calculate the Euclidean distances (Eq. 2.16) among Ada, Bob, and Cathy. It is easy to see that

$$dist(\text{Ada}, \text{Bob}) = dist(\text{Bob}, \text{Cathy}) = dist(\text{Ada}, \text{Cathy}) = \sqrt{2}.$$

According to Euclidean distance, the three customers are equivalently similar (or dissimilar) to each other. However, a close look tells us that Ada should be more similar to Cathy than to Bob because Ada and Cathy share one common purchased item, P_1. ■

As shown in Example 11.9, the traditional distance measures can be ineffective on high-dimensional data. Such distance measures may be dominated by the noise in many dimensions. Therefore, clusters in the full, high-dimensional space can be unreliable, and finding such clusters may not be meaningful.

"Then what kinds of clusters are meaningful on high-dimensional data?" For cluster analysis of high-dimensional data, we still want to group similar objects together. However, the data space is often too big and too messy. An additional challenge is that we need to find not only clusters, but, for each cluster, a set of attributes that manifest the cluster. In other words, a cluster on high-dimensional data often is defined using a small set of attributes instead of the full data space. Essentially, clustering high-dimensional data should return groups of objects as clusters (as conventional cluster analysis does), *in addition to*, for each cluster, the set of attributes that characterize the cluster. For example, in Table 11.4, to characterize the similarity between Ada and Cathy, P_1 may be returned as the attribute because Ada and Cathy both purchased P_1.

Clustering high-dimensional data is the search for clusters and the space in which they exist. Thus, there are two major kinds of methods:

- *Subspace clustering approaches* search for clusters existing in subspaces of the given high-dimensional data space, where a subspace is defined using a subset of attributes in the full space. Subspace clustering approaches are discussed in Section 11.2.2.

- *Dimensionality reduction approaches* try to construct a much lower-dimensional space and search for clusters in such a space. Often, a method may construct new dimensions by combining some dimensions from the original data. Dimensionality reduction methods are the topic of Section 11.2.4.

In general, clustering high-dimensional data raises several new challenges in addition to those of conventional clustering:

- A major issue is how to create appropriate models for clusters in high-dimensional data. Unlike conventional clusters in low-dimensional spaces, clusters hidden in high-dimensional data are often significantly smaller. For example, when clustering customer-purchase data, we would not expect many users to have similar purchase patterns. Searching for such small but meaningful clusters is like finding needles in a haystack. As shown before, the conventional distance measures can be ineffective. Instead, we often have to consider various more sophisticated techniques that can model correlations and consistency among objects in subspaces.
- There are typically an exponential number of possible subspaces or dimensionality reduction options, and thus the optimal solutions are often computationally prohibitive. For example, if the original data space has 1000 dimensions, and we want to find clusters of dimensionality 10, then there are $\binom{1000}{10} = 2.63 \times 10^{23}$ possible subspaces.

Subspace Clustering Methods

"How can we find subspace clusters from high-dimensional data?" Many methods have been proposed. They generally can be categorized into three major groups: *subspace search methods*, *correlation-based clustering methods*, and *biclustering methods*.

Subspace Search Methods

A subspace search method searches various subspaces for clusters. Here, a cluster is a subset of objects that are similar to each other in a subspace. The similarity is often captured by conventional measures such as distance or density. For example, the CLIQUE algorithm introduced in Section 10.5.2 is a subspace clustering method. It enumerates subspaces and the clusters in those subspaces in a dimensionality-increasing order, and applies antimonotonicity to prune subspaces in which no cluster may exist.

A major challenge that subspace search methods face is how to search a series of subspaces effectively and efficiently. Generally there are two kinds of strategies:

- *Bottom-up approaches* start from low-dimensional subspaces and search higher-dimensional subspaces only when there may be clusters in those higher-dimensional

subspaces. Various pruning techniques are explored to reduce the number of higher-dimensional subspaces that need to be searched. CLIQUE is an example of a bottom-up approach.

- *Top-down approaches* start from the full space and search smaller and smaller subspaces recursively. Top-down approaches are effective only if the *locality assumption* holds, which require that the subspace of a cluster can be determined by the local neighborhood.

Example 11.10 **PROCLUS, a top-down subspace approach.** PROCLUS is a *k*-medoid-like method that first generates *k* potential cluster centers for a high-dimensional data set using a sample of the data set. It then refines the subspace clusters iteratively. In each iteration, for each of the current *k*-medoids, PROCLUS considers the local neighborhood of the medoid in the whole data set, and identifies a subspace for the cluster by minimizing the standard deviation of the distances of the points in the neighborhood to the medoid on each dimension. Once all the subspaces for the medoids are determined, each point in the data set is assigned to the closest medoid according to the corresponding subspace. Clusters and possible outliers are identified. In the next iteration, new medoids replace existing ones if doing so improves the clustering quality. ■

Correlation-Based Clustering Methods

While subspace search methods search for clusters with a similarity that is measured using conventional metrics like distance or density, *correlation-based approaches* can further discover clusters that are defined by advanced correlation models.

Example 11.11 **A correlation-based approach using PCA.** As an example, a *PCA-based approach* first applies PCA (Principal Components Analysis; see Chapter 3) to derive a set of new, uncorrelated dimensions, and then mine clusters in the new space or its subspaces. In addition to PCA, other space transformations may be used, such as the Hough transform or fractal dimensions. ■

For additional details on subspace search methods and correlation-based clustering methods, please refer to the bibliographic notes (Section 11.7).

Biclustering Methods

In some applications, we want to cluster both objects and attributes simultaneously. The resulting clusters are known as *biclusters* and meet four requirements: (1) only a small set of objects participate in a cluster; (2) a cluster only involves a small number of attributes; (3) an object may participate in multiple clusters, or does not participate in any cluster; and (4) an attribute may be involved in multiple clusters, or is not involved in any cluster. Section 11.2.3 discusses biclustering in detail.

Biclustering

In the cluster analysis discussed so far, we cluster objects according to their attribute values. Objects and attributes are not treated in the same way. However, in some applications, objects and attributes are defined in a symmetric way, where data analysis involves searching data matrices for submatrices that show unique patterns as clusters. This kind of clustering technique belongs to the category of biclustering.

This section first introduces two motivating application examples of biclustering—gene expression and recommender systems. You will then learn about the different types of biclusters. Last, we present biclustering methods.

Application Examples

Biclustering techniques were first proposed to address the needs for analyzing gene expression data. A *gene* is a unit of the passing-on of traits from a living organism to its offspring. Typically, a gene resides on a segment of DNA. Genes are critical for all living things because they specify all proteins and functional RNA chains. They hold the information to build and maintain a living organism's cells and pass genetic traits to offspring. Synthesis of a functional gene product, either RNA or protein, relies on the process of gene expression. A *genotype* is the genetic makeup of a cell, an organism, or an individual. *Phenotypes* are observable characteristics of an organism. *Gene expression* is the most fundamental level in genetics in that genotypes cause phenotypes.

Using *DNA chips* (also known as *DNA microarrays*) and other biological engineering techniques, we can measure the expression level of a large number (possibly all) of an organism's genes, in a number of different experimental conditions. Such conditions may correspond to different time points in an experiment or samples from different organs. Roughly speaking, the *gene expression data* or *DNA microarray data* are conceptually a gene-sample/condition matrix, where each row corresponds to one gene, and each column corresponds to one sample or condition. Each element in the matrix is a real number and records the expression level of a gene under a specific condition. Figure 11.3 shows an illustration.

From the clustering viewpoint, an interesting issue is that a gene expression data matrix can be analyzed in two dimensions—the gene dimension and the sample/condition dimension.

- When analyzing in the *gene dimension*, we treat each gene as an object and treat the samples/conditions as attributes. By mining in the gene dimension, we may find patterns shared by multiple genes, or cluster genes into groups. For example, we may find a group of genes that express themselves similarly, which is highly interesting in bioinformatics, such as in finding pathways.
- When analyzing in the *sample/condition dimension*, we treat each sample/condition as an object and treat the genes as attributes. In this way, we may find patterns of samples/conditions, or cluster samples/conditions into groups. For example, we may find the differences in gene expression by comparing a group of tumor samples and nontumor samples.

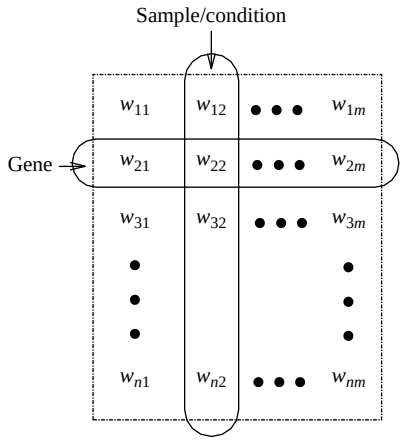


Figure 11.3 Microarrary data matrix.

Example 11.12 **Gene expression.** Gene expression matrices are popular in bioinformatics research and development. For example, an important task is to classify a new gene using the expression data of the gene and that of other genes in known classes. Symmetrically, we may classify a new sample (e.g., a new patient) using the expression data of the sample and that of samples in known classes (e.g., tumor and nontumor). Such tasks are invaluable in understanding the mechanisms of diseases and in clinical treatment. ■

As can be seen, many gene expression data mining problems are highly related to cluster analysis. However, a challenge here is that, instead of clustering in one dimension (e.g., gene or sample/condition), in many cases we need to cluster in two dimensions simultaneously (e.g., both gene and sample/condition). Moreover, unlike the clustering models we have discussed so far, a cluster in a gene expression data matrix is a *submatrix* and usually has the following characteristics:

- Only a small set of genes participate in the cluster.
- The cluster involves only a small subset of samples/conditions.
- A gene may participate in multiple clusters, or may not participate in any cluster.
- A sample/condition may be involved in multiple clusters, or may not be involved in any cluster.

To find clusters in gene-sample/condition matrices, we need new clustering techniques that meet the following requirements for *biclustering*:

- A cluster of genes is defined using only a subset of samples/conditions.
- A cluster of samples/conditions is defined using only a subset of genes.

- The clusters are neither *exclusive* (e.g., where one gene can participate in multiple clusters) nor *exhaustive* (e.g., where a gene may not participate in any cluster).

Biclustering is useful not only in bioinformatics, but also in other applications as well. Consider recommender systems as an example.

Example 11.13 **Using biclustering for a recommender system.** *AllElectronics* collects data from customers' evaluations of products and uses the data to recommend products to customers. The data can be modeled as a customer-product matrix, where each row represents a customer, and each column represents a product. Each element in the matrix represents a customer's evaluation of a product, which may be a score (e.g., like, like somewhat, not like) or purchase behavior (e.g., buy or not). Figure 11.4 illustrates the structure.

The customer-product matrix can be analyzed in two dimensions: the *customer* dimension and the *product* dimension. Treating each customer as an object and products as attributes, *AllElectronics* can find customer groups that have similar preferences or purchase patterns. Using products as objects and customers as attributes, *AllElectronics* can mine product groups that are similar in customer interest.

Moreover, *AllElectronics* can mine clusters in both customers and products simultaneously. Such a cluster contains a subset of customers and involves a subset of products. For example, *AllElectronics* is highly interested in finding a group of customers who all like the same group of products. Such a cluster is a submatrix in the customer-product matrix, where all elements have a high value. Using such a cluster, *AllElectronics* can make recommendations in two directions. First, the company can recommend products to new customers who are similar to the customers in the cluster. Second, the company can recommend to customers new products that are similar to those involved in the cluster. ■

As with biclusters in a gene expression data matrix, the biclusters in a customer-product matrix usually have the following characteristics:

- Only a small set of customers participate in a cluster.
- A cluster involves only a small subset of products.
- A customer can participate in multiple clusters, or may not participate in any cluster.

	Products			
	w_{11}	w_{12}	$\cdots$	w_{1m}
Customers	w_{21}	w_{22}	$\cdots$	w_{2m}
	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	w_{n1}	w_{n2}	$\cdots$	w_{nm}

Figure 11.4 Customer–product matrix.

- A product may be involved in multiple clusters, or may not be involved in any cluster.

Biclustering can be applied to customer-product matrices to mine clusters satisfying these requirements.

Types of Biclusters

"How can we model biclusters and mine them?" Let's start with some basic notation. For the sake of simplicity, we will use "genes" and "conditions" to refer to the two dimensions in our discussion. Our discussion can easily be extended to other applications. For example, we can simply replace "genes" and "conditions" by "customers" and "products" to tackle the customer-product biclustering problem.

Let $A = \{a_1, \ldots, a_n\}$ be a set of genes and $B = \{b_1, \ldots, b_m\}$ be a set of conditions. Let $E = [e_{ij}]$ be a gene expression data matrix, that is, a gene-condition matrix, where $1 \leq i \leq n$ and $1 \leq j \leq m$. A submatrix $I \times J$ is defined by a subset $I \subseteq A$ of genes and a subset $J \subseteq B$ of conditions. For example, in the matrix shown in Figure 11.5, $\{a_1, a_{33}, a_{86}\} \times \{b_6, b_{12}, b_{36}, b_{99}\}$ is a submatrix.

A bicluster is a submatrix where genes and conditions follow consistent patterns. We can define different types of biclusters based on such patterns.

- As the simplest case, a submatrix $I \times J$ $(I \subseteq A, J \subseteq B)$ is a **bicluster with constant values** if for any $i \in I$ and $j \in J$, $e_{ij} = c$, where c is a constant. For example, the submatrix $\{a_1, a_{33}, a_{86}\} \times \{b_6, b_{12}, b_{36}, b_{99}\}$ in Figure 11.5 is a bicluster with constant values.
- A bicluster is interesting if each row has a constant value, though different rows may have different values. A **bicluster with constant values on rows** is a submatrix $I \times J$ such that for any $i \in I$ and $j \in J$, $e_{ij} = c + \alpha_i$, where α_i is the adjustment for row i. For example, Figure 11.6 shows a bicluster with constant values on rows.

  Symmetrically, a **bicluster with constant values on columns** is a submatrix $I \times J$ such that for any $i \in I$ and $j \in J$, $e_{ij} = c + \beta_j$, where β_j is the adjustment for column j.

	⋯	b_6	⋯	b_{12}	⋯	b_{36}	⋯	b_{99} ⋯
a_1	⋯	60	⋯	60	⋯	60	⋯	60 ⋯
⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯ ⋯
a_{33}	⋯	60	⋯	60	⋯	60	⋯	60 ⋯
⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯ ⋯
a_{86}	⋯	60	⋯	60	⋯	60	⋯	60 ⋯
⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯ ⋯

Figure 11.5 Gene-condition matrix, a submatrix, and a bicluster.

- More generally, a bicluster is interesting if the rows change in a synchronized way with respect to the columns and vice versa. Mathematically, a **bicluster with coherent values** (also known as **a pattern-based cluster**) is a submatrix $I \times J$ such that for any $i \in I$ and $j \in J$, $e_{ij} = c + \alpha_i + \beta_j$, where α_i and β_j are the adjustment for row i and column j, respectively. For example, Figure 11.7 shows a bicluster with coherent values.

  It can be shown that $I \times J$ is a bicluster with coherent values if and only if for any $i_1, i_2 \in I$ and $j_1, j_2 \in J$, then $e_{i_1j_1} - e_{i_2j_1} = e_{i_1j_2} - e_{i_2j_2}$. Moreover, instead of using addition, we can define a bicluster with coherent values using multiplication, that is, $e_{ij} = c \cdot \alpha_i \cdot \beta_j$. Clearly, biclusters with constant values on rows or columns are special cases of biclusters with coherent values.

- In some applications, we may only be interested in the up- or down-regulated changes across genes or conditions without constraining the exact values. A **bicluster with coherent evolutions on rows** is a submatrix $I \times J$ such that for any $i_1, i_2 \in I$ and $j_1, j_2 \in J$, $(e_{i_1j_1} - e_{i_1j_2})(e_{i_2j_1} - e_{i_2j_2}) \geq 0$. For example, Figure 11.8 shows a bicluster with coherent evolutions on rows. Symmetrically, we can define biclusters with coherent evolutions on columns.

Next, we study how to mine biclusters.

10	10	10	10	10
20	20	20	20	20
50	50	50	50	50
0	0	0	0	0

Figure 11.6 Bicluster with constant values on rows.

10	50	30	70	20
20	60	40	80	30
50	90	70	110	60
0	40	20	60	10

Figure 11.7 Bicluster with coherent values.

10	50	30	70	20
20	100	50	1000	30
50	100	90	120	80
0	80	20	100	10

Figure 11.8 Bicluster with coherent evolutions on rows.

Biclustering Methods

The previous specification of the types of biclusters only considers ideal cases. In real data sets, such perfect biclusters rarely exist. When they do exist, they are usually very small. Instead, random noise can affect the readings of e_{ij} and thus prevent a bicluster in nature from appearing in a perfect shape.

There are two major types of methods for discovering biclusters in data that may come with noise. **Optimization-based methods** conduct an iterative search. At each iteration, the submatrix with the highest significance score is identified as a bicluster. The process terminates when a user-specified condition is met. Due to cost concerns in computation, greedy search is often employed to find local optimal biclusters. **Enumeration methods** use a tolerance threshold to specify the degree of noise allowed in the biclusters to be mined, and then tries to enumerate all submatrices of biclusters that satisfy the requirements. We use the δ-Cluster and MaPle algorithms as examples to illustrate these ideas.

Optimization Using the δ-Cluster Algorithm

For a submatrix, $I \times J$, the mean of the ith row is

$$e_{iJ} = \frac{1}{|J|} \sum_{j \in J} e_{ij}. \tag{11.16}$$

Symmetrically, the mean of the jth column is

$$e_{Ij} = \frac{1}{|I|} \sum_{i \in I} e_{ij}. \tag{11.17}$$

The mean of all elements in the submatrix is

$$e_{IJ} = \frac{1}{|I||J|} \sum_{i \in I, j \in J} e_{ij} = \frac{1}{|I|} \sum_{i \in I} e_{iJ} = \frac{1}{|J|} \sum_{j \in J} e_{Ij}. \tag{11.18}$$

The quality of the submatrix as a bicluster can be measured by the *mean-squared residue* value as

$$H(I \times J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2. \tag{11.19}$$

Submatrix $I \times J$ is a **δ-bicluster** if $H(I \times J) \le \delta$, where $\delta \ge 0$ is a threshold. When $\delta = 0$, $I \times J$ is a perfect bicluster with coherent values. By setting $\delta > 0$, a user can specify the tolerance of average noise per element against a perfect bicluster, because in Eq. (11.19) the residue on each element is

$$\text{residue}(e_{ij}) = e_{ij} - e_{iJ} - e_{Ij} + e_{IJ}. \tag{11.20}$$

A *maximal δ-bicluster* is a δ-bicluster $I \times J$ such that there does not exist another δ-bicluster $I' \times J'$, and $I \subseteq I'$, $J \subseteq J'$, and at least one inequality holds. Finding the

maximal δ-bicluster of the largest size is computationally costly. Therefore, we can use a heuristic greedy search method to obtain a local optimal cluster. The algorithm works in two phases.

- In the *deletion phase*, we start from the whole matrix. While the mean-squared residue of the matrix is over δ, we iteratively remove rows and columns. At each iteration, for each row i, we compute the *mean-squared residue* as

$$d(i) = \frac{1}{|J|} \sum_{j \in J} (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2. \tag{11.21}$$

  Moreover, for each column j, we compute the *mean-squared residue* as

$$d(j) = \frac{1}{|I|} \sum_{i \in I} (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2. \tag{11.22}$$

  We remove the row or column of the largest mean-squared residue. At the end of this phase, we obtain a submatrix $I \times J$ that is a δ-bicluster. However, the submatrix may not be maximal.

- In the *addition phase*, we iteratively expand the δ-bicluster $I \times J$ obtained in the deletion phase as long as the δ-bicluster requirement is maintained. At each iteration, we consider rows and columns that are not involved in the current bicluster $I \times J$ by calculating their mean-squared residues. A row or column of the smallest mean-squared residue is added into the current δ-bicluster.

This greedy algorithm can find one δ-bicluster only. To find multiple biclusters that do not have heavy overlaps, we can run the algorithm multiple times. After each execution where a δ-bicluster is output, we can replace the elements in the output bicluster by random numbers. Although the greedy algorithm may find neither the optimal biclusters nor all biclusters, it is very fast even on large matrices.

Enumerating All Biclusters Using MaPle

As mentioned, a submatrix $I \times J$ is a bicluster with coherent values if and only if for any $i_1, i_2 \in I$ and $j_1, j_2 \in J$, $e_{i_1j_1} - e_{i_1j_2} = e_{i_2j_1} - e_{i_2j_2}$. For any 2×2 submatrix of $I \times J$, we can define a *p-score* as

$$p\text{-score}\begin{pmatrix} e_{i_1j_1} & e_{i_1j_2} \\ e_{i_2j_1} & e_{i_2j_2} \end{pmatrix} = |(e_{i_1j_1} - e_{i_2j_1}) - (e_{i_1j_2} - e_{i_2j_2})|. \tag{11.23}$$

A submatrix $I \times J$ is a **δ-pCluster** (for *p*attern-based *cluster*) if the *p*-score of every 2×2 submatrix of $I \times J$ is at most δ, where $\delta \geq 0$ is a threshold specifying a user's tolerance of noise against a perfect bicluster. Here, the *p*-score controls the noise on every element in a bicluster, while the mean-squared residue captures the average noise.

An interesting property of δ-pCluster is that if $I \times J$ is a δ-pCluster, then every $x \times y$ $(x, y \geq 2)$ submatrix of $I \times J$ is also a δ-pCluster. This monotonicity enables

us to obtain a succinct representation of nonredundant δ-pClusters. A δ-pCluster is maximal if no more rows or columns can be added into the cluster while maintaining the δ-pCluster property. To avoid redundancy, instead of finding all δ-pClusters, we only need to compute all maximal δ-pClusters.

MaPle is an algorithm that enumerates all maximal δ-pClusters. It systematically enumerates every combination of conditions using a set enumeration tree and a depth-first search. This enumeration framework is the same as the pattern-growth methods for frequent pattern mining (Chapter 6). Consider gene expression data. For each condition combination, J, MaPle finds the maximal subsets of genes, I, such that $I \times J$ is a δ-pCluster. If $I \times J$ is not a submatrix of another δ-pCluster, then $I \times J$ is a maximal δ-pCluster.

There may be a huge number of condition combinations. MaPle prunes many unfruitful combinations using the monotonicity of δ-pClusters. For a condition combination, J, if there does not exist a set of genes, I, such that $I \times J$ is a δ-pCluster, then we do not need to consider any superset of J. Moreover, we should consider $I \times J$ as a candidate of a δ-pCluster only if for every $(|J|-1)$-subset J' of J, $I \times J'$ is a δ-pCluster. MaPle also employs several pruning techniques to speed up the search while retaining the completeness of returning all maximal δ-pClusters. For example, when examining a current δ-pCluster, $I \times J$, MaPle collects all the genes and conditions that may be added to expand the cluster. If these candidate genes and conditions together with I and J form a submatrix of a δ-pCluster that has already been found, then the search of $I \times J$ and any superset of J can be pruned. Interested readers may refer to the bibliographic notes for additional information on the MaPle algorithm (Section 11.7).

An interesting observation here is that the search for maximal δ-pClusters in MaPle is somewhat similar to mining frequent closed itemsets. Consequently, MaPle borrows the depth-first search framework and ideas from the pruning techniques of pattern-growth methods for frequent pattern mining. This is an example where frequent pattern mining and cluster analysis may share similar techniques and ideas.

An advantage of MaPle and the other algorithms that enumerate all biclusters is that they guarantee the completeness of the results and do not miss any overlapping biclusters. However, a challenge for such enumeration algorithms is that they may become very time consuming if a matrix becomes very large, such as a customer-purchase matrix of hundreds of thousands of customers and millions of products.

Dimensionality Reduction Methods and Spectral Clustering

Subspace clustering methods try to find clusters in subspaces of the original data space. In some situations, it is more effective to construct a new space instead of using subspaces of the original data. This is the motivation behind dimensionality reduction methods for clustering high-dimensional data.

Example 11.14 Clustering in a derived space. Consider the three clusters of points in Figure 11.9. It is not possible to cluster these points in any subspace of the original space, $X \times Y$, because

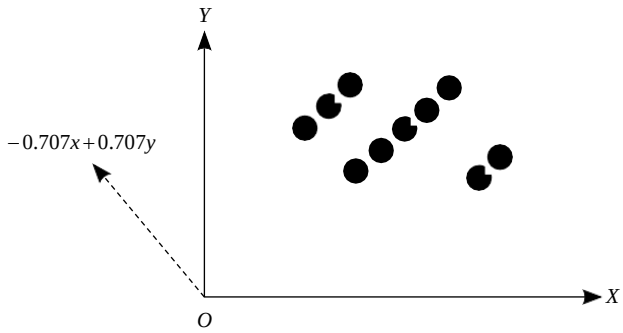


Figure 11.9 Clustering in a derived space may be more effective.

all three clusters would end up being projected onto overlapping areas in the x and y axes. What if, instead, we construct a new dimension, $-\frac{\sqrt{2}}{2}x + \frac{\sqrt{2}}{2}y$ (shown as a dashed line in the figure)? By projecting the points onto this new dimension, the three clusters become apparent. ■

Although Example 11.14 involves only two dimensions, the idea of constructing a new space (so that any clustering structure that is hidden in the data becomes well manifested) can be extended to high-dimensional data. Preferably, the newly constructed space should have low dimensionality.

There are many dimensionality reduction methods. A straightforward approach is to apply feature selection and extraction methods to the data set such as those discussed in Chapter 3. However, such methods may not be able to detect the clustering structure. Therefore, methods that combine feature extraction and clustering are preferred. In this section, we introduce *spectral clustering*, a group of methods that are effective in high-dimensional data applications.

Figure 11.10 shows the general framework for spectral clustering approaches. The Ng-Jordan-Weiss algorithm is a spectral clustering method. Let's have a look at each step of the framework. In doing so, we also note special conditions that apply to the Ng-Jordan-Weiss algorithm as an example.

Given a set of objects, $o_1, \ldots, o_n$, the distance between each pair of objects, $dist(o_i, o_j)$ $(1 \le i, j \le n)$, and the desired number k of clusters, a spectral clustering approach works as follows.

1. Using the distance measure, calculate an *affinity matrix*, W, such that

$$W_{ij} = e^{-\frac{dist(o_i, o_j)}{\sigma^2}},$$

where σ is a scaling parameter that controls how fast the affinity W_{ij} decreases as $dist(o_i, o_j)$ increases. In the Ng-Jordan-Weiss algorithm, W_{ii} is set to 0.

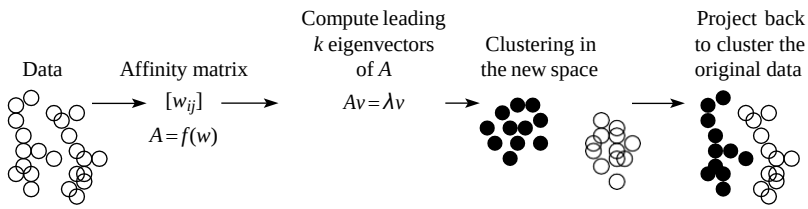


Figure 11.10 The framework of spectral clustering approaches. *Source:* Adapted from Slide 8 at *http://videolectures.net/micued08_azran_mcl/*.

2. Using the affinity matrix W, derive a matrix $A = f(W)$. The way in which this is done can vary. The Ng-Jordan-Weiss algorithm defines a matrix, D, as a diagonal matrix such that D_{ii} is the sum of the ith row of W, that is,

$$D_{ii} = \sum_{j=1}^{n} W_{ij}. \tag{11.24}$$

A is then set to

$$A = D^{-1/2} W D^{-1/2}. \tag{11.25}$$

3. Find the k leading eigenvectors of A. Recall that the *eigenvectors* of a square matrix are the nonzero vectors that remain proportional to the original vector after being multiplied by the matrix. Mathematically, a vector $\boldsymbol{v}$ is an eigenvector of matrix A if $A\boldsymbol{v} = \lambda\boldsymbol{v}$, where λ is called the corresponding *eigenvalue*. This step derives k new dimensions from A, which are based on the affinity matrix W. Typically, k should be much smaller than the dimensionality of the original data.

   The Ng-Jordan-Weiss algorithm computes the k eigenvectors with the largest eigenvalues $x_1, \ldots, x_k$ of A.

4. Using the k leading eigenvectors, project the original data into the new space defined by the k leading eigenvectors, and run a clustering algorithm such as k-means to find k clusters.

   The Ng-Jordan-Weiss algorithm stacks the k largest eigenvectors in columns to form a matrix $X = [x_1 x_2 \cdots x_k] \in \mathbb{R}^{n \times k}$. The algorithm forms a matrix Y by renormalizing each row in X to have unit length, that is,

$$Y_{ij} = \frac{X_{ij}}{\sqrt{\sum_{j=1}^{k} X_{ij}^2}}. \tag{11.26}$$

The algorithm then treats each row in Y as a point in the k-dimensional space $\mathbb{R}^k$, and runs k-means (or any other algorithm serving the partitioning purpose) to cluster the points into k clusters.

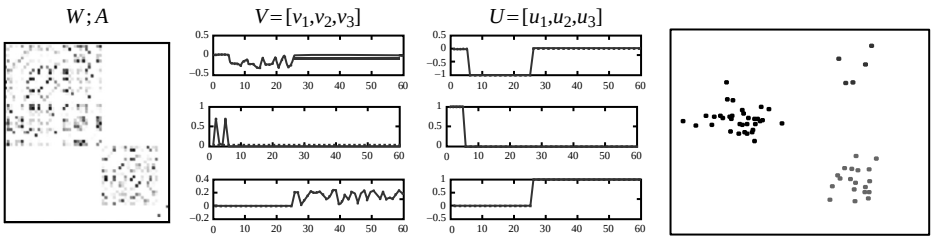


Figure 11.11 The new dimensions and the clustering results of the Ng-Jordan-Weiss algorithm. *Source:* Adapted from Slide 9 at *http://videolectures.net/micued08_azran_mcl/*.

5. Assign the original data points to clusters according to how the transformed points are assigned in the clusters obtained in step 4.

   In the Ng-Jordan-Weiss algorithm, the original object o_i is assigned to the jth cluster if and only if matrix Y's row i is assigned to the jth cluster as a result of step 4.

In spectral clustering methods, the dimensionality of the new space is set to the desired number of clusters. This setting expects that each new dimension should be able to manifest a cluster.

Example 11.15 **The Ng-Jordan-Weiss algorithm.** Consider the set of points in Figure 11.11. The data set, the affinity matrix, the three largest eigenvectors, and the normalized vectors are shown. Note that with the three new dimensions (formed by the three largest eigenvectors), the clusters are easily detected. ■

Spectral clustering is effective in high-dimensional applications such as image processing. Theoretically, it works well when certain conditions apply. Scalability, however, is a challenge. Computing eigenvectors on a large matrix is costly. Spectral clustering can be combined with other clustering methods, such as biclustering. Additional information on other dimensionality reduction clustering methods, such as kernel PCA, can be found in the bibliographic notes (Section 11.7).

Clustering Graph and Network Data

Cluster analysis on graph and network data extracts valuable knowledge and information. Such data are increasingly popular in many applications. We discuss applications and challenges of clustering graph and network data in Section 11.3.1. Similarity measures for this form of clustering are given in Section 11.3.2. You will learn about graph clustering methods in Section 11.3.3.

In general, the terms *graph* and *network* can be used interchangeably. In the rest of this section, we mainly use the term *graph*.

Applications and Challenges

As a customer relationship manager at *AllElectronics*, you notice that a lot of data relating to customers and their purchase behavior can be preferably modeled using graphs.

Example 11.16 **Bipartite graph.** The customer purchase behavior at *AllElectronics* can be represented ina *bipartite graph*. In a bipartite graph, vertices can be divided into two disjoint sets so that each edge connects a vertex in one set to a vertex in the other set. For the *AllElectronics* customer purchase data, one set of vertices represents customers, with one customer per vertex. The other set represents products, with one product per vertex. An edge connects a customer to a product, representing the purchase of the product by the customer. Figure 11.12 shows an illustration.

"What kind of knowledge can we obtain by a cluster analysis of the customer-product bipartite graph?" By clustering the customers such that those customers buying similar sets of products are placed into one group, a customer relationship manager can make product recommendations. For example, suppose Ada belongs to a customer cluster in which most of the customers purchased a digital camera in the last 12 months, but Ada has yet to purchase one. As manager, you decide to recommend a digital camera to her. Alternatively, we can cluster products such that those products purchased by similar sets of customers are grouped together. This clustering information can also be used for product recommendations. For example, if a digital camera and a high-speed flash memory card belong to the same product cluster, then when a customer purchases a digital camera, we can recommend the high-speed flash memory card. ■

Bipartite graphs are widely used in many applications. Consider another example.

Example 11.17 **Web search engines.** In web search engines, search logs are archived to record user queries and the corresponding *click-through information*. (The click-through information tells us on which pages, given as a result of a search, the user clicked.) The query and click-through information can be represented using a bipartite graph, where the two sets

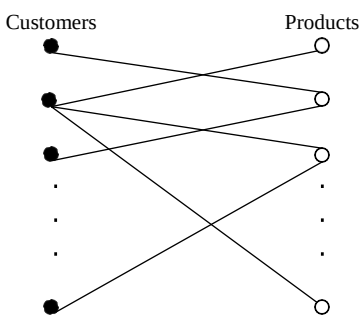


Figure 11.12 Bipartite graph representing customer-purchase data.

of vertices correspond to queries and web pages, respectively. An edge links a query to a web page if a user clicks the web page when asking the query. Valuable information can be obtained by cluster analyses on the query–web page bipartite graph. For instance, we may identify queries posed in different languages, but that mean the same thing, if the click-through information for each query is similar.

As another example, all the web pages on the Web form a directed graph, also known as the *web graph*, where each web page is a vertex, and each hyperlink is an edge pointing from a source page to a destination page. Cluster analysis on the web graph can disclose communities, find hubs and authoritative web pages, and detect web spams. ■

In addition to bipartite graphs, cluster analysis can also be applied to other types of graphs, including general graphs, as elaborated Example 11.18.

Example 11.18 **Social network.** A *social network* is a social structure. It can be represented as a graph, where the vertices are individuals or organizations, and the links are interdependencies between the vertices, representing friendship, common interests, or collaborative activities. *AllElectronics'* customers form a social network, where each customer is a vertex, and an edge links two customers if they know each other.

As customer relationship manager, you are interested in finding useful information that can be derived from *AllElectronics'* social network through cluster analysis. You obtain clusters from the network, where customers in a cluster know each other or have friends in common. Customers within a cluster may influence one another regarding purchase decision making. Moreover, communication channels can be designed to inform the "heads" of clusters (i.e., the "best" connected people in the clusters), so that promotional information can be spread out quickly. Thus, you may use customer clustering to promote sales at *AllElectronics*.

As another example, the authors of scientific publications form a social network, where the authors are vertices and two authors are connected by an edge if they coauthored a publication. The network is, in general, a weighted graph because an edge between two authors can carry a weight representing the strength of the collaboration such as how many publications the two authors (as the end vertices) coauthored. Clustering the coauthor network provides insight as to communities of authors and patterns of collaboration. ■

"Are there any challenges specific to cluster analysis on graph and network data?" In most of the clustering methods discussed so far, objects are represented using a set of attributes. A unique feature of graph and network data is that only objects (as vertices) and relationships between them (as edges) are given. No dimensions or attributes are explicitly defined. To conduct cluster analysis on graph and network data, there are two major new challenges.

- *"How can we measure the similarity between two objects on a graph accordingly?"* Typically, we cannot use conventional distance measures, such as Euclidean distance. Instead, we need to develop new measures to quantify the similarity. Such

measures often are not metric, and thus raise new challenges regarding the development of efficient clustering methods. Similarity measures for graphs are discussed in Section 11.3.2.

- *"How can we design clustering models and methods that are effective on graph and network data?"* Graph and network data are often complicated, carrying topological structures that are more sophisticated than traditional cluster analysis applications. Many graph data sets are large, such as the web graph containing at least tens of billions of web pages in the publicly indexable Web. Graphs can also be sparse where, on average, a vertex is connected to only a small number of other vertices in the graph. To discover accurate and useful knowledge hidden deep in the data, a good clustering method has to accommodate these factors. Clustering methods for graph and network data are introduced in Section 11.3.3.

Similarity Measures

"How can we measure the similarity or distance between two vertices in a graph?" In our discussion, we examine two types of measures: *geodesic distance* and *distance based on random walk*.

Geodesic Distance

A simple measure of the distance between two vertices in a graph is the shortest path between the vertices. Formally, the **geodesic distance** between two vertices is the length in terms of the number of edges of the shortest path between the vertices. For two vertices that are not connected in a graph, the geodesic distance is defined as infinite.

Using geodesic distance, we can define several other useful measurements for graph analysis and clustering. Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, we define the following:

- For a vertex $v \in V$, the **eccentricity** of v, denoted $eccen(v)$, is the largest geodesic distance between v and any other vertex $u \in V - \{v\}$. The eccentricity of v captures how far away v is from its remotest vertex in the graph.
- The **radius** of graph G is the minimum eccentricity of all vertices. That is,

$$r = \min_{v \in V} eccen(v). \tag{11.27}$$

  The radius captures the distance between the "most central point" and the "farthest border" of the graph.
- The **diameter** of graph G is the maximum eccentricity of all vertices. That is,

$$d = \max_{v \in V} eccen(v). \tag{11.28}$$

  The diameter represents the largest distance between any pair of vertices.
- A **peripheral vertex** is a vertex that achieves the diameter.

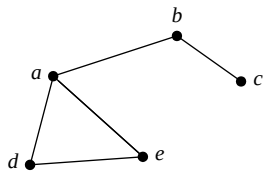


Figure 11.13 A graph, *G*, where vertices *c*, *d*, and *e* are peripheral.

Example 11.19 **Measurements based on geodesic distance.** Consider graph *G* in Figure 11.13. The eccentricity of *a* is 2, that is, *eccen(a)* = 2, *eccen(b)* = 2, and *eccen(c)* = *eccen(d)* = *eccen(e)* = 3. Thus, the radius of *G* is 2, and the diameter is 3. Note that it is not necessary that $d = 2 \times r$. Vertices *c*, *d*, and *e* are peripheral vertices. ■

SimRank: Similarity Based on Random Walkand Structural Context

For some applications, geodesic distance may be inappropriate in measuring the similarity between vertices in a graph. Here we introduce SimRank, a similarity measure based on random walk and on the structural context of the graph. In mathematics, a *random walk* is a trajectory that consists of taking successive random steps.

Example 11.20 **Similarity between people in a social network.** Let's consider measuring the similarity between two vertices in the *AllElectronics* customer social network of Example 11.18. Here, similarity can be explained as the closeness between two participants in the network, that is, how close two people are in terms of the relationship represented by the social network.

"How well can the geodesic distance measure similarity and closeness in such a network?" Suppose Ada and Bob are two customers in the network, and the network is undirected. The geodesic distance (i.e., the length of the shortest path between Ada and Bob) is the shortest path that a message can be passed from Ada to Bob and vice versa. However, this information is not useful for *AllElectronics'* customer relationship management because the company typically does not want to send a specific message from one customer to another. Therefore, geodesic distance does not suit the application.

"What does similarity mean in a social network?" We consider two ways to define similarity:

- Two customers are considered similar to one another if they have similar neighbors in the social network. This heuristic is intuitive because, in practice, two people receiving recommendations from a good number of common friends often make similar decisions. This kind of similarity is based on the local structure (i.e., the *neighborhoods*) of the vertices, and thus is called *structural context–based similarity*.

- Suppose *AllElectronics* sends promotional information to both Ada and Bob in the social network. Ada and Bob may randomly forward such information to their friends (or *neighbors*) in the network. The closeness between Ada and Bob can then be measured by the likelihood that other customers simultaneously receive the promotional information that was originally sent to Ada and Bob. This kind of similarity is based on the random walk reachability over the network, and thus is referred to as *similarity based on random walk*. ■

Let's have a closer look at what is meant by similarity based on structural context, and similarity based on random walk.

The intuition behind similarity based on structural context is that two vertices in a graph are similar if they are connected to similar vertices. To measure such similarity, we need to define the notion of individual neighborhood. In a directed graph $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges, for a vertex $v \in V$, the *individual in-neighborhood* of v is defined as

$$I(v) = \{u \mid (u, v) \in E\}. \tag{11.29}$$

Symmetrically, we define the *individual out-neighborhood* of v as

$$O(v) = \{w \mid (v, w) \in E\}. \tag{11.30}$$

Following the intuition illustrated in Example 11.20, we define SimRank, a structural-context similarity, with a value that is between 0 and 1 for any pair of vertices. For any vertex, $v \in V$, the similarity between the vertex and itself is $s(v, v) = 1$ because the neighborhoods are identical. For vertices $u, v \in V$ such that $u \neq v$, we can define

$$s(u, v) = \frac{C}{|I(u)||I(v)|} \sum_{x \in I(u)} \sum_{y \in I(v)} s(x, y), \tag{11.31}$$

where C is a constant between 0 and 1. A vertex may not have any in-neighbors. Thus, we define Eq. (11.31) to be 0 when either $I(u)$ or $I(v)$ is $\emptyset$. Parameter C specifies the rate of decay as similarity is propagated across edges.

"How can we compute SimRank?" A straightforward method iteratively evaluates Eq. (11.31) until a fixed point is reached. Let $s_i(u, v)$ be the SimRank score calculated at the ith round. To begin, we set

$$s_0(u, v) = \begin{cases} 0 & \text{if } u \neq v \\ 1 & \text{if } u = v. \end{cases} \tag{11.32}$$

We use Eq. (11.31) to compute s_{i+1} from s_i as

$$s_{i+1}(u, v) = \frac{C}{|I(u)||I(v)|} \sum_{x \in I(u)} \sum_{y \in I(v)} s_i(x, y). \tag{11.33}$$

It can be shown that $\lim_{i \to \infty} s_i(u, v) = s(u, v)$. Additional methods for approximating SimRank are given in the bibliographic notes (Section 11.7).

Now, let's consider similarity based on random walk. A directed graph is *strongly connected* if, for any two nodes u and v, there is a path from u to v and another path from v to u. In a strongly connected graph, $G = (V, E)$, for any two vertices, $u, v \in V$, we can define the *expected distance* from u to v as

$$d(u, v) = \sum_{t: u \sim v} P[t] l(t), \tag{11.34}$$

where $u \sim v$ is a path starting from u and ending at v that may contain cycles but does not reach v until the end. For a *traveling tour*, $t = w_1 \to w_2 \to \cdots \to w_k$, its length is $l(t) = k - 1$. The probability of the tour is defined as

$$P[t] = \begin{cases} \prod_{i=1}^{k-1} \frac{1}{|O(w_i)|} & \text{if } l(t) > 0 \\ 0 & \text{if } l(t) = 0. \end{cases} \tag{11.35}$$

To measure the probability that a vertex w receives a message that originated simultaneously from u and v, we extend the expected distance to the notion of *expected meeting distance*, that is,

$$m(u, v) = \sum_{t: (u,v) \sim (x,x)} P[t] l(t), \tag{11.36}$$

where $(u, v) \sim (x, x)$ is a pair of tours $u \sim x$ and $v \sim x$ of the same length. Using a constant C between 0 and 1, we define the *expected meeting probability* as

$$p(u, v) = \sum_{t: (u,v) \sim (x,x)} P[t] C^{l(t)}, \tag{11.37}$$

which is a similarity measure based on random walk. Here, the parameter C specifies the probability of continuing the walk at each step of the trajectory.

It has been shown that $s(u, v) = p(u, v)$ for any two vertices, u and v. That is, SimRank is based on both structural context and random walk.

Graph Clustering Methods

Let's consider how to conduct clustering on a graph. We first describe the intuition behind graph clustering. We then discuss two general categories of graph clustering methods.

To find clusters in a graph, imagine cutting the graph into pieces, each piece being a cluster, such that the vertices within a cluster are well connected and the vertices in different clusters are connected in a much weaker way. Formally, for a graph, $G = (V, E)$,

a **cut**, $C = (S, T)$, is a partitioning of the set of vertices V in G, that is, $V = S \cup T$ and $S \cap T = \emptyset$. The *cut set* of a cut is the set of edges, $\{(u, v) \in E \mid u \in S, v \in T\}$. The *size* of the cut is the number of edges in the cut set. For weighted graphs, the size of a cut is the sum of the weights of the edges in the cut set.

"What kinds of cuts are good for deriving clusters in graphs?" In graph theory and some network applications, a minimum cut is of importance. A cut is *minimum* if the cut's size is not greater than any other cut's size. There are polynomial time algorithms to compute minimum cuts of graphs. Can we use these algorithms in graph clustering?

Example 11.21 **Cuts and clusters.** Consider graph G in Figure 11.14. The graph has two clusters: $\{a, b, c, d, e, f\}$ and $\{g, h, i, j, k\}$, and one outlier vertex, l.

Consider cut $C_1 = (\{a, b, c, d, e, f, g, h, i, j, k\}, \{l\})$. Only one edge, namely, (e, l), crosses the two partitions created by C_1. Therefore, the cut set of C_1 is $\{(e, l)\}$ and the size of C_1 is 1. (Note that the size of any cut in a connected graph cannot be smaller than 1.) As a minimum cut, C_1 does not lead to a good clustering because it only separates the outlier vertex, l, from the rest of the graph.

Cut $C_2 = (\{a, b, c, d, e, f, l\}, \{g, h, i, j, k\})$ leads to a much better clustering than C_1. The edges in the cut set of C_2 are those connecting the two "natural clusters" in the graph. Specifically, for edges (d, h) and (e, k) that are in the cut set, most of the edges connecting d, h, e, and k belong to one cluster. ■

Example 11.21 indicates that using a minimum cut is unlikely to lead to a good clustering. We are better off choosing a cut where, for each vertex u that is involved in an edge in the cut set, most of the edges connecting to u belong to one cluster. Formally, let $deg(u)$ be the degree of u, that is, the number of edges connecting to u. The *sparsity* of a cut $C = (S, T)$ is defined as

$$\Phi = \frac{\text{cut size}}{\min\{|S|, |T|\}}. \tag{11.38}$$

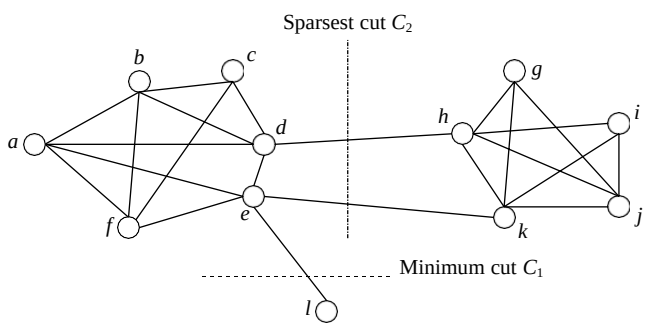


Figure 11.14 A graph G and two cuts.

A cut is *sparsest* if its sparsity is not greater than the sparsity of any other cut. There may be more than one sparsest cut.

In Example 11.21 and Figure 11.14, C_2 is a sparsest cut. Using sparsity as the objective function, a sparsest cut tries to minimize the number of edges crossing the partitions and balance the partitions in size.

Consider a clustering on a graph $G = (V, E)$ that partitions the graph into k clusters. The **modularity** of a clustering assesses the quality of the clustering and is defined as

$$Q = \sum_{i=1}^{k} \left(\frac{l_i}{|E|} - \left(\frac{d_i}{2|E|} \right)^2 \right), \tag{11.39}$$

where l_i is the number of edges between vertices in the ith cluster, and d_i is the sum of the degrees of the vertices in the ith cluster. The modularity of a clustering of a graph is the difference between the fraction of all edges that fall into individual clusters and the fraction that would do so if the graph vertices were randomly connected. The optimal clustering of graphs maximizes the modularity.

Theoretically, many graph clustering problems can be regarded as finding good cuts, such as the sparsest cuts, on the graph. In practice, however, a number of challenges exist:

- **High computational cost:** Many graph cut problems are computationally expensive. The sparsest cut problem, for example, is NP-hard. Therefore, finding the optimal solutions on large graphs is often impossible. A good trade-off between efficiency/scalability and quality has to be achieved.
- **Sophisticated graphs:** Graphs can be more sophisticated than the ones described here, involving weights and/or cycles.
- **High dimensionality:** A graph can have many vertices. In a similarity matrix, a vertex is represented as a vector (a row in the matrix) with a dimensionality that is the number of vertices in the graph. Therefore, graph clustering methods must handle high dimensionality.
- **Sparsity:** A large graph is often sparse, meaning each vertex on average connects to only a small number of other vertices. A similarity matrix from a large sparse graph can also be sparse.

There are two kinds of methods for clustering graph data, which address these challenges. One uses clustering methods for high-dimensional data, while the other is designed specifically for clustering graphs.

The first group of methods is based on generic clustering methods for high-dimensional data. They extract a similarity matrix from a graph using a similarity measure such as those discussed in Section 11.3.2. A generic clustering method can then be applied on the similarity matrix to discover clusters. Clustering methods for

high-dimensional data are typically employed. For example, in many scenarios, once a similarity matrix is obtained, spectral clustering methods (Section 11.2.4) can be applied. Spectral clustering can approximate optimal graph cut solutions. For additional information, please refer to the bibliographic notes (Section 11.7).

The second group of methods is specific to graphs. They search the graph to find well-connected components as clusters. Let's look at a method called **SCAN** (Structural Clustering Algorithm for Networks) as an example.

Given an undirected graph, $G = (V, E)$, for a vertex, $u \in V$, the neighborhood of u is $\Gamma(u) = \{v \mid (u, v) \in E\} \cup \{u\}$. Using the idea of structural-context similarity, SCAN measures the similarity between two vertices, $u, v \in V$, by the normalized common neighborhood size, that is,

$$\sigma(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{|\Gamma(u)||\Gamma(v)|}}. \tag{11.40}$$

The larger the value computed, the more similar the two vertices. SCAN uses a similarity threshold ε to define the cluster membership. For a vertex, $u \in V$, the *ε-neighborhood* of u is defined as $N_\varepsilon(u) = \{v \in \Gamma(u) \mid \sigma(u, v) \geq \varepsilon\}$. The ε-neighborhood of u contains all neighbors of u with a structural-context similarity to u that is at least ε.

In SCAN, a *core vertex* is a vertex inside of a cluster. That is, $u \in V$ is a core vertex if $|N_\varepsilon(u)| \geq \mu$, where μ is a popularity threshold. SCAN grows clusters from core vertices. If a vertex v is in the ε-neighborhood of a core u, then v is assigned to the same cluster as u. This process of growing clusters continues until no cluster can be further grown. The process is similar to the density-based clustering method, DBSCAN (Chapter 10).

Formally, a vertex v can be *directly reached* from a core u if $v \in N_\varepsilon(u)$. Transitively, a vertex v can be *reached* from a core u if there exist vertices $w_1, \ldots, w_n$ such that w_1 can be reached from u, w_i can be reached from w_{i-1} for $1 < i \leq n$, and v can be reached from w_n. Moreover, two vertices, $u, v \in V$, which may or may not be cores, are said to be *connected* if there exists a core w such that both u and v can be reached from w. All vertices in a cluster are connected. A cluster is a maximum set of vertices such that every pair in the set is connected.

Some vertices may not belong to any cluster. Such a vertex u is a *hub* if the neighborhood $\Gamma(u)$ of u contains vertices from more than one cluster. If a vertex does not belong to any cluster, and is not a hub, it is an *outlier*.

The SCAN algorithm is shown in Figure 11.15. The search framework closely resembles the cluster-finding process in DBSCAN. SCAN finds a cut of the graph, where each cluster is a set of vertices that are connected based on the transitive similarity in a structural context.

An advantage of SCAN is that its time complexity is linear with respect to the number of edges. In very large and sparse graphs, the number of edges is in the same scale of the number of vertices. Therefore, SCAN is expected to have good scalability on clustering large graphs.

```
Algorithm: SCAN for clusters on graph data.
Input: a graph G = (V, E), a similarity threshold ε, and a
  population threshold μ
Output: a set of clusters
Method: set all vertices in V unlabeled
  for all unlabeled vertex u do
    if u is a core then
      generate a new cluster-id c
      insert all v ∈ N_ε(u) into a queue Q
      while Q ≠ ∅ do
        w ← the first vertex in Q
        R ← the set of vertices that can be directly reached from w
        for all s ∈ R do
          if s is not unlabeled or labeled as nonmember then
            assign the current cluster-id c to s
          endif
          if s is unlabeled then
            insert s into queue Q
          endif
        endfor
        remove w from Q
      end while
    else
      label u as nonmember
    endif
  endfor
  for all vertex u labeled nonmember do
    if ∃x, y ∈ Γ(u) : x and y have different cluster-ids then
      label u as hub
    else
      label u as outlier
    endif
  endfor
```

Figure 11.15 SCAN algorithm for cluster analysis on graph data.

Clustering with Constraints

Users often have background knowledge that they want to integrate into cluster analysis. There may also be application-specific requirements. Such information can be modeled as clustering constraints. We approach the topic of clustering with constraints in two steps. Section 11.4.1 categorizes the types of constraints for clustering graph data. Methods for clustering with constraints are introduced in Section 11.4.2.

Categorization of Constraints

This section studies how to categorize the constraints used in cluster analysis. Specifically, we can categorize constraints according to the subjects on which they are set, or on how strongly the constraints are to be enforced.

As discussed in Chapter 10, cluster analysis involves three essential aspects: objects as instances of clusters, clusters as groups of objects, and the similarity among objects. Therefore, the first method we discuss categorizes constraints according to what they are applied to. We thus have three types: *constraints on instances*, *constraints on clusters*, and *constraints on similarity measurement*.

Constraints on instances: A *constraint on instances* specifies how a pair or a set of instances should be grouped in the cluster analysis. Two common types of constraints from this category include:

- **Must-link constraints.** If a must-link constraint is specified on two objects x and y, then x and y should be grouped into one cluster in the output of the cluster analysis. These must-link constraints are transitive. That is, if must-link(x, y) and must-link(y, z), then must-link(x, z).
- **Cannot-link constraints.** Cannot-link constraints are the opposite of must-link constraints. If a cannot-link constraint is specified on two objects, x and y, then in the output of the cluster analysis, x and y should belong to different clusters. Cannot-link constraints can be entailed. That is, if cannot-link(x, y), must-link(x, x'), and must-link(y, y'), then cannot-link(x', y').

A constraint on instances can be defined using specific instances. Alternatively, it can also be defined using instance variables or attributes of instances. For example, a constraint,

$$Constraint(x, y) : \text{must-link}(x, y) \text{ if } dist(x, y) \leq \text{‹},$$

uses the distance between objects to specify a must-link constraint.

Constraints on clusters: A *constraint on clusters* specifies a requirement on the clusters, possibly using attributes of the clusters. For example, a constraint may specify the minimum number of objects in a cluster, the maximum diameter of a cluster, or the shape of a cluster (e.g., a convex). The number of clusters specified for partitioning clustering methods can be regarded as a constraint on clusters.

Constraints on similarity measurement: Often, a similarity measure, such as Euclidean distance, is used to measure the similarity between objects in a cluster analysis. In some applications, exceptions apply. A *constraint on similarity measurement* specifies a requirement that the similarity calculation must respect. For example, to cluster people as moving objects in a plaza, while Euclidean distance is used to give

the walking distance between two points, a constraint on similarity measurement is that the trajectory implementing the shortest distance cannot cross a wall.

There can be more than one way to express a constraint, depending on the category. For example, we can specify a constraint on clusters as

$$Constraint_1\text{: the diameter of a cluster cannot be larger than } d.$$

The requirement can also be expressed using a constraint on instances as

$$Constraint_1^{'}\text{ : cannot-link}(x, y) \text{ if } dist(x, y) > d. \tag{11.41}$$

Example 11.22 **Constraints on instances, clusters, and similarity measurement.** *AllElectronics* clusters its customers so that each group of customers can be assigned to a customer relationship manager. Suppose we want to specify that all customers at the same address are to be placed in the same group, which would allow more comprehensive service to families. This can be expressed using a must-link constraint on instances:

$$Constraint_{family}(x, y) \text{ : must-link}(x, y) \text{ if } x.address = y.address.$$

AllElectronics has eight customer relationship managers. To ensure that they each have a similar workload, we place a constraint on clusters such that there should be eight clusters, and each cluster should have at least 10% of the customers and no more than 15% of the customers. We can calculate the spatial distance between two customers using the driving distance between the two. However, if two customers live in different countries, we have to use the flight distance instead. This is a constraint on similarity measurement. ■

Another way to categorize clustering constraints considers how firmly the constraints have to be respected. A constraint is **hard** if a clustering that violates the constraint is unacceptable. A constraint is **soft** if a clustering that violates the constraint is not preferable but acceptable when no better solution can be found. Soft constraints are also called *preferences*.

Example 11.23 **Hard and soft constraints.** For *AllElectronics*, $Constraint_{family}$ in Example 11.22 is a hard constraint because splitting a family into different clusters could prevent the company from providing comprehensive services to the family, leading to poor customer satisfaction. The constraint on the number of clusters (which corresponds to the number of customer relationship managers in the company) is also hard. Example 11.22 also has a constraint to balance the size of clusters. While satisfying this constraint is strongly preferred, the company is flexible in that it is willing to assign a senior and more capable customer relationship manager to oversee a larger cluster. Therefore, the constraint is soft. ■

Ideally, for a specific data set and a set of constraints, all clusterings satisfy the constraints. However, it is possible that there may be no clustering of the data set that

satisfies all the constraints. Trivially, if two constraints in the set conflict, then no clustering can satisfy them at the same time.

Example 11.24 **Conflicting constraints.** Consider these constraints:

$$\text{must-link}(x, y) \text{ if } dist(x, y) < 5$$

$$\text{cannot-link}(x, y) \text{ if } dist(x, y) > 3.$$

If a data set has two objects, x, y, such that $dist(x, y) = 4$, then no clustering can satisfy both constraints simultaneously.

Consider these two constraints:

$$\text{must-link}(x, y) \text{ if } dist(x, y) < 5$$

$$\text{must-link}(x, y) \text{ if } dist(x, y) < 3.$$

The second constraint is redundant given the first. Moreover, for a data set where the distance between any two objects is at least 5, every possible clustering of the objects satisfies the constraints. ■

"How can we measure the quality and the usefulness of a set of constraints?" In general, we consider either their informativeness, or their coherence. The **informativeness** is the amount of information carried by the constraints that is beyond the clustering model. Given a data set, D, a clustering method, A, and a set of constraints, C, the informativeness of C with respect to A on D can be measured by the fraction of constraints in C that are unsatisfied by the clustering computed by A on D. The higher the informativeness, the more specific the requirements and background knowledge that the constraints carry. The **coherence** of a set of constraints is the degree of agreement among the constraints themselves, which can be measured by the redundancy among the constraints.

Methods for Clustering with Constraints

Although we can categorize clustering constraints, applications may have very different constraints of specific forms. Consequently, various techniques are needed to handle specific constraints. In this section, we discuss the general principles of handling hard and soft constraints.

Handling Hard Constraints

A general strategy for handling hard constraints is to strictly respect the constraints in the cluster assignment process. To illustrate this idea, we will use partitioning clustering as an example.

Given a data set and a set of constraints on instances (i.e., must-link or cannot-link constraints), how can we extend the *k*-means method to satisfy such constraints? The **COP-*k*-means algorithm** works as follows:

1. **Generate superinstances for must-link constraints**. Compute the transitive closure of the must-link constraints. Here, all must-link constraints are treated as an equivalence relation. The closure gives one or multiple subsets of objects where all objects in a subset must be assigned to one cluster. To represent such a subset, we replace all those objects in the subset by the mean. The superinstance also carries a weight, which is the number of objects it represents.

   After this step, the must-link constraints are always satisfied.

2. **Conduct modified k-means clustering.** Recall that, in *k*-means, an object is assigned to the closest center. What if a nearest-center assignment violates a cannot-link constraint? To respect cannot-link constraints, we modify the center assignment process in *k*-means to a *nearest feasible center assignment*. That is, when the objects are assigned to centers in sequence, at each step we make sure the assignments so far do not violate any cannot-link constraints. An object is assigned to the nearest center so that the assignment respects all cannot-link constraints.

Because COP-*k*-means ensures that no constraints are violated at every step, it does not require any backtracking. It is a greedy algorithm for generating a clustering that satisfies all constraints, provided that no conflicts exist among the constraints.

Handling Soft Constraints

Clustering with soft constraints is an optimization problem. When a clustering violates a soft constraint, a penalty is imposed on the clustering. Therefore, the optimization goal of the clustering contains two parts: optimizing the clustering quality and minimizing the constraint violation penalty. The overall objective function is a combination of the clustering quality score and the penalty score.

To illustrate, we again use partitioning clustering as an example. Given a data set and a set of soft constraints on instances, the **CVQE** (**Constrained Vector Quantization Error) algorithm** conducts *k*-means clustering while enforcing constraint violation penalties. The objective function used in CVQE is the sum of the distance used in *k*-means, adjusted by the constraint violation penalties, which are calculated as follows.

- **Penalty of a must-link violation.** If there is a must-link constraint on objects *x* and *y*, but they are assigned to two different centers, c_1 and c_2, respectively, then the constraint is violated. As a result, $dist(c_1, c_2)$, the distance between c_1 and c_2, is added to the objective function as the penalty.
- **Penalty of a cannot-link violation.** If there is a cannot-link constraint on objects *x* and *y*, but they are assigned to a common center, *c*, then the constraint is violated.

The distance, $dist(c, c^r)$, between c and c^r is added to the objective function as the penalty.

Speeding up Constrained Clustering

Constraints, such as on similarity measurements, can lead to heavy costs in clustering. Consider the following **clustering with obstacles** problem: To cluster people as moving objects in a plaza, Euclidean distance is used to measure the walking distance between two points. However, a constraint on similarity measurement is that the trajectory implementing the shortest distance cannot cross a wall (Section 11.4.1). Because obstacles may occur between objects, the distance between two objects may have to be derived by geometric computations (e.g., involving triangulation). The computational cost is high if a large number of objects and obstacles are involved.

The clustering with obstacles problem can be represented using a graphical notation. First, a point, $\boldsymbol{p}$, is **visible** from another point, $\boldsymbol{q}$, in the region R if the straight line joining $\boldsymbol{p}$ and $\boldsymbol{q}$ does not intersect any obstacles. A **visibility graph** is the graph, $VG = (V, E)$, such that each vertex of the obstacles has a corresponding node in V and two nodes, $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$, in V are joined by an edge in E if and only if the corresponding vertices they represent are visible to each other. Let $VG^r = (V^r, E^r)$ be a visibility graph created from VG by adding two additional points, $\boldsymbol{p}$ and $\boldsymbol{q}$, in V^r. E^r contains an edge joining two points in V^r if the two points are mutually visible. The shortest path between two points, $\boldsymbol{p}$ and $\boldsymbol{q}$, will be a subpath of VG^r, as shown in Figure 11.16(a). We see that it begins with an edge from $\boldsymbol{p}$ to either $\boldsymbol{v_1}$, $\boldsymbol{v_2}$, or $\boldsymbol{v_3}$, goes through a path in VG, and then ends with an edge from either $\boldsymbol{v_4}$ or $\boldsymbol{v_5}$ to $\boldsymbol{q}$.

To reduce the cost of distance computation between any two pairs of objects or points, several preprocessing and optimization techniques can be used. One method groups points that are close together into microclusters. This can be done by first triangulating the region R into triangles, and then grouping nearby points in the same triangle into microclusters, using a method similar to BIRCH or DBSCAN, as shown in Figure 11.16(b). By processing microclusters rather than individual points, the overall computation is reduced. After that, precomputation can be performed to build two

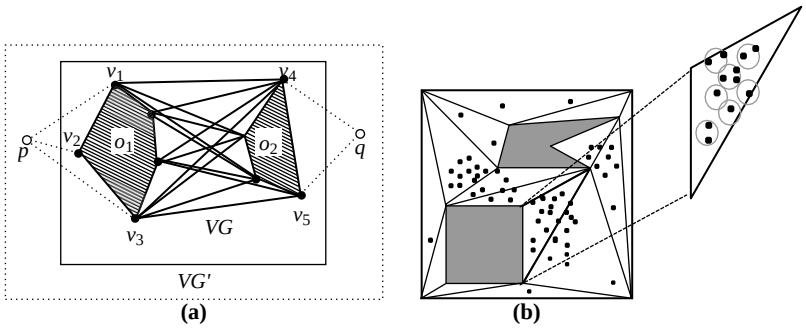


Figure 11.16 Clustering with obstacle objects (o_1 and o_2): (a) a visibility graph and (b) triangulation of regions with microclusters. *Source:* Adapted from Tung, Hou, and Han [THH01].

kinds of join indices based on the computation of the shortest paths: (1) *VV*

indices, for any pair of obstacle vertices, and (2) *MV indices*, for any pair of microcluster and obstacle vertex. Use of the indices helps further optimize the overall performance.

Using such precomputation and optimization strategies, the distance between any two points (at the granularity level of a microcluster) can be computed efficiently. Thus, the clustering process can be performed in a manner similar to a typical efficient *k*-medoids algorithm, such as CLARANS, and achieve good clustering quality for large data sets.