12 ADVANCES IN DATA MINING

Chapter Objectives

- Analyze the characteristics of graph-mining algorithms and introduce some illustrative examples.
- Identify the required changes in data-mining algorithm when temporal and spatial components are introduced.
- Introduce the basic characteristics of distributed data-mining algorithms and specific modifications for distributed Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering.
- Describe the differences between causality and correlation.
- Introduce the basic principles in Bayesian networks modeling.
- Know when and how to include privacy protection in a data-mining process.
- Summarize social and legal aspects of data-mining applications.

Current technological progress permits the storage and access of large amounts of data at virtually no cost. These developments have created unprecedented opportunities for large-scale data-driven discoveries, as well as the potential for fundamental gains in scientific and business understanding. The popularity of the Internet and the Web makes it imperative that the data-mining framework is extended to include distributed, time- and space-dependent information and tools. New complex and distributed systems are supported by enhanced multimedia data sources such as images and signals, and advanced data structures such as graphs. In this environment, data-mining applications have new social and legal challenges, and privacy preservation is one of the priority tasks.

12.1 GRAPH MINING

Traditional data-mining tasks such as association-rule mining, market-basket analysis, and cluster analysis commonly attempt to find patterns in a data set characterized by a collection of independent instances of a single relation. This is consistent with the classical statistical inference problem of trying to identify a model given a random sample from a common underlying distribution. An emerging challenge for data mining is the problem of mining richly structured data sets, where the objects are linked in some way. Many real-world data sets describe a variety of entity types linked via multiple types of relations. These links provide additional context that can be helpful for many data-mining tasks. Yet multi-relational data violate the traditional assumption of independent, identically distributed data instances that provides the basis for many statistical machine-learning algorithms. Naively applying traditional statistical inference procedures, which assume that samples are independent, may lead in many applications to inappropriate conclusions. Care must be taken that potential correlations due to links between samples are handled appropriately. In fact, record linkage is knowledge that should be exploited. Clearly, this is information that can be used to improve the predictive accuracy of the learned models: Attributes of linked objects are often correlated and links are more likely to exist between objects that have some commonality. Relationships between objects represent a rich source of information, and ultimately knowledge. Therefore, new approaches that can exploit the dependencies across the attribute and link structure are needed. Certainly, as a general data structure, a graph can meet the demands of modeling complicated relations among data.

Graph-based data mining represents a collection of techniques for mining the relational aspects of data represented as a graph. It has the task of finding novel, useful, and understandable graph-theoretic patterns in a graph representation of data. Graph mining has become an important topic of research recently because of numerous applications to a wide variety of data-mining problems in computational biology, chemical data analysis, drug discovery, and communication networking. Some examples of graph-represented data are presented in Figure 12.1. Traditional data-mining and management algorithms such as clustering, classification, frequent-pattern mining, and indexing have now been extended to the graph scenario. While the field of graph mining has been a relatively recent development in the data-mining community, it has been studied under different names by other groups of researchers. This is because research on graphs has a long history in mathematics, but most notably important results are obtained by sociologists in the field of a social network analysis. However, there are

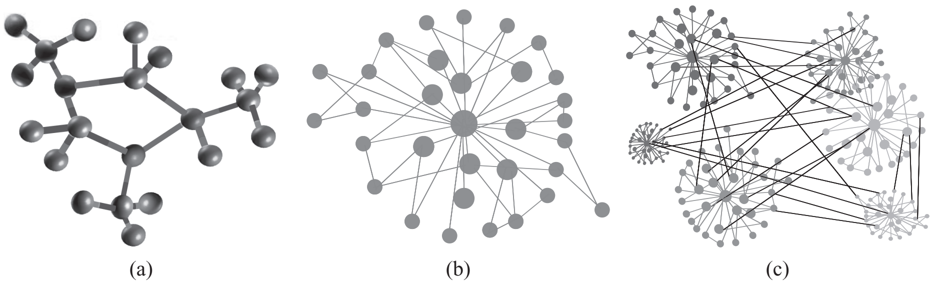


Figure 12.1. Graph representation of data. (a) Chemical compound; (b) social network; (c) genome co-expression network.

important differences, and the primary one is that of network size. Social networks are, in general, small, with the larger studies considering a few hundred nodes. On the other hand, graph-mining data sets in new application domains may typically consist of hundreds of thousands of nodes and millions of edges.

Many data sets of interest today are best described as a linked collection of interrelated objects. These may represent homogeneous networks, in which there is a single-object type and a single-link type, or richer, heterogeneous networks, in which there may be multiple object and link types, and possibly other semantic information. Examples of homogeneous networks include single-mode social networks, such as people connected by friendship links, or the World Wide Web (WWW), a collection of linked Web pages. Examples of heterogeneous networks include those in medical domains describing patients, diseases, treatments, and contacts, or in bibliographic domains describing publications, authors, and venues. Graph-mining techniques explicitly consider these links when building predictive or descriptive models of the linked data.

The requirement of different applications with graph-based data sets is not very uniform. Thus, graph models and mining algorithms that work well in one domain may not work well in another. For example, chemical data is often represented as graphs in which the nodes correspond to atoms, and the links correspond to bonds between the atoms. The individual graphs are quite small although there are significant repetitions among the different nodes. Biological data are modeled in a similar way as chemical data. However, the individual graphs are typically much larger. Protein interaction networks link proteins that must work together to perform some particular biological functions. A single biological network could easily contain thousands of nodes. In the case of computer networks and the Web, the number of nodes in the underlying graph may be massive. Computer networks consist of routers/computers representing nodes, and the links between them. Since the number of nodes is massive, this can lead to a very large number of distinct edges. Social networks may be modeled with large graphs that are defined by people who appear as nodes, and links that correspond to communications or relationships between these different people. The links in the social network can be used to determine relevant communities, members with particular expertise sets, and the flow of information in the social network. For example, the problem of community detection in social networks is related to the problem of node clustering of very large graphs. In this case, we wish to determine dense clusters of nodes based on the underlying linkage structure. It is clear that the design of a particular mining algorithm depends upon the application domain at hand.

Before introducing some illustrative examples of graph-mining techniques, some basic concepts from graph theory will be summarized. Graph theory provides a vocabulary that can be used to label and denote many structural properties in data. Also, graph theory gives us mathematical operations and ideas with which many of these properties can be quantified and measured.

A graph G = G(N, L) consists of two sets of information: a set of nodes N = {n_1, n_2, . . . , n_k} and a set of links between pairs of nodes L = {l_1, l_2,…, l_m}. A graph with nodes and without links is called an empty graph, while the graph with only one node is a trivial graph. Two nodes n_i and n_j are *adjacent* if there is a link between them. A graph G(N, L) can be presented as a diagram as in Figure 12.2a in which points depict nodes, and lines between two points are links. A graph G′(N′, L′) is a subgraph of G(N, L) if N′ $\subseteq$ N and L′ $\subseteq$ L.

An induced subgraph of a graph *G* has a subset of the nodes of *G,* and the same links between pairs of nodes as in *G*. For example, the subgraph (b) in Figure 12.3 is an induced subgraph of the graph (a), but the subgraph (c) is a general subgraph but not an induced subgraph of G, since the incoming link L_1 of the node labeled *N*2 in (a) is not retained in (c) while the node labeled *N*2 is included.

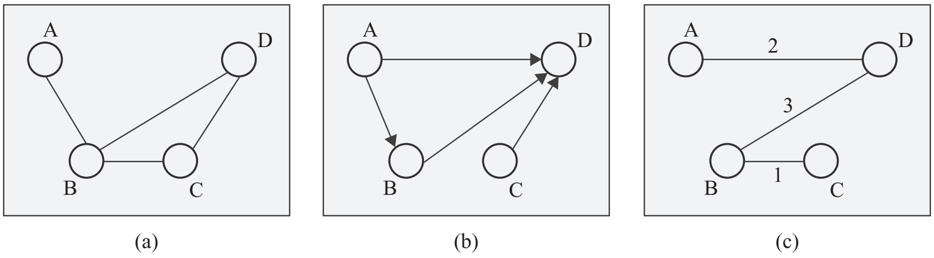


Figure 12.2. Undirected, directed, and weighted graphs.

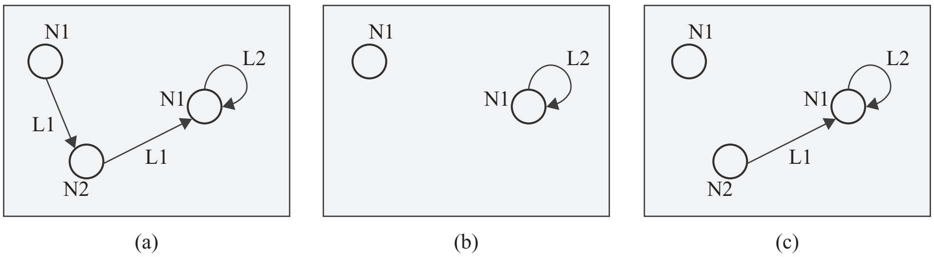


Figure 12.3. An induced subgraph and a general graph.

The degree of a node, denoted by $d(n_i)$, is the number of links connected to the given node. Equivalently, the degree of a node is the number of nodes adjacent to it. For example, in Figure 12.2a the degree d(B) = 3. The degree of a node ranges from a minimum of 0 if no nodes are adjacent to a given node, and to a maximum of k − 1 if a given node is adjacent to all the other nodes in the graph. The degrees are very easy to compute, and yet they can be very informative for many applications. For some applications, it is useful to summarize the degrees of all nodes in the graph. The mean nodal degree is a statistic that reports the average degree of the nodes in the graph:

$$d_{av} = \frac{\sum_{i=1}^{k} d(n_i)}{k}$$

For the graph in Figure 12.2a, $d_{av} = (1 + 3 + 2 + 2)/4 = 2$. One might also be interested in the variability of the nodal degrees. If all the degrees of all nodes are equal, the graph is said to be d-regular, and its variability is equal to 0. If the nodes differ in degrees, the variance is calculated as a measure for variability:

$$SD^2 = \frac{\sum_{i=1}^{k} (d(n_i) - d_{av})^2}{k}$$

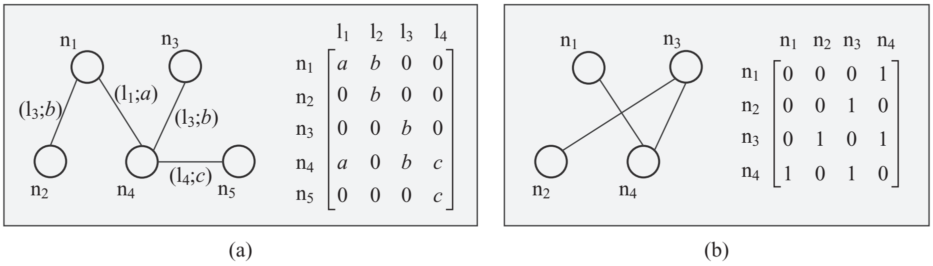
The maximum number of links in the graph is defined by the number of nodes. Since there are k nodes in the graph, and if we exclude the loops as links, there are k(k − 1)/2 possible links that could be presented in the graph. If all links are present, then all nodes are adjacent, and the graph is said to be complete. Consider now what proportion of these links is actually present. The density of a graph is the proportion of actual links in the graph compared with the maximum possible number of links. The density of a graph goes from 0, when there are no links in a graph, to 1, when all possible links are presented.

We can also analyze the paths between a pair of nodes, and they are represented by multiple links. We define a path from $s \in N$ to $t \in N$ as an alternating sequence of nodes and links, beginning with node s and ending with node t, such that each link connects its preceding with its succeeding node. It is likely that there are several paths between a given pair of nodes, and that these paths are differ in lengths (number of links included in the path). The shortest path between two nodes is referred to as a *geodesic*. The geodesic distance d_G or simply the distance between two nodes is defined as the length of a geodesic between them, and it represents the length of the shortest path. By definition, $d_G(s; s) = 0$ for every node $s \in N$, and $d_G(s; t) = d_G(t; s)$ for each pair of nodes s, $t \in V$. For example, the paths between nodes A and D in Figure 12.2a are A-B-D and A-B-C-D. The shortest one is A-B-D, and therefore the distance d(A,D) = 2. If there is no path between two nodes, then the distance is infinite (or undefined). The *diameter* of a connected graph is the length of the largest geodesic between any pairs of nodes. It represents the largest nodal eccentricity. The diameter of a graph can range from 1, if the graph is complete, to a maximum of k − 1. If a graph is not connected, its diameter is infinite. For the graph in Figure 12.2a the diameter is 2.

These basic definitions are introduced for non-labeled and nondirected graphs such as the one in Figure 12.2a. A *directed graph*, or digraph G(N, L) consists of a set of nodes N and a set of directed links L. Each link is defined by an order pair of nodes (sender, receiver). Since each link is directed, there are k(k − 1) possible links in the graph. A *labeled graph* is a graph in which each link carries some value. Therefore, a labeled graph G consists of three sets of information: G(N,L,V), where the new component $V = \{v_1, v_2, \ldots, v_t\}$ is a set of values attached to links. An example of a directed graph is given in Figure 12.2b, while the graph in Figure 12.2c is a labeled graph. Different applications use different types of graphs in modeling linked data. In this chapter the primary focus is on undirected and unlabeled graphs although the reader still has to be aware that there are numerous graph-mining algorithms for directed and/or labeled graphs.

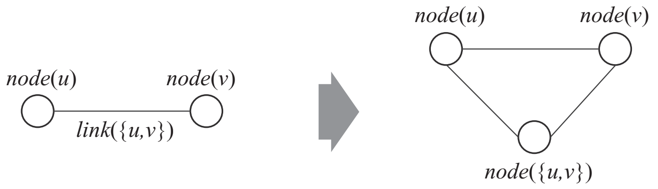
Besides a graphical representation, each graph may be presented in the form of the *incidence matrix* I(G) where nodes are indexing rows and links are indexing columns. The matrix entry in the position (i,j) has value *a* if node n_i is incident with *a*, the link l_j. The other matrix representation of a graph (in the case of undirected and unlabeled graphs) is the k × k adjacency matrix, where both rows and columns are defined by nodes. The graph-structured data can be transformed without much computational effort into an *adjacency matrix*, which is a well-known representation of a graph in mathematical graph theory. On the intersection (i,j) is the value of 1 if the nodes n_i and n_j are connected by a link; otherwise it is 0 value (Fig. 12.4).

If a graph has labeled links, the following conversion of the graph to a new graph that has labels at its nodes only is applied. This transformation reduces the ambiguity in the adjacency matrix representation. Given a node pair (*u*, *v*) and a directed or undirected link {*u, v*} between the nodes, that is, *node*(*u*)–*link*({*u, v*})–*node*(*v*) where *node*() and *link*() stand for the labels of a node and a link, respectively. The *link*() label information can be removed and transformed into a new *node*(*u, v*), and the following triangular graph may be deduced where the original information of the node pair and the link between them are preserved.



$$\begin{array}{c|cccc} & l_1 & l_2 & l_3 & l_4 \\ \hline n_1 & a & b & 0 & 0 \\ n_2 & 0 & b & 0 & 0 \\ n_3 & 0 & 0 & b & 0 \\ n_4 & a & 0 & b & c \\ n_5 & 0 & 0 & 0 & c \end{array} \qquad \begin{array}{c|cccc} & n_1 & n_2 & n_3 & n_4 \\ \hline n_1 & 0 & 0 & 0 & 1 \\ n_2 & 0 & 0 & 1 & 0 \\ n_3 & 0 & 1 & 0 & 1 \\ n_4 & 1 & 0 & 1 & 0 \end{array}$$

Figure 12.4. Matrix representations of graphs. (a) Incidence matrix:nodes × links; (b) adjancency matrix:nodes ×nodes.



This operation on each link in the original graph can convert the graph into another graph representation having no labels on the links while preserving the topological information of the original graph.

Accordingly, the adjacency matrix of Figure 12.5a is converted to that of Figure 12.5b as follows.

$$\begin{array}{c|ccc} & N_1 & N_1 & N_2 \\ \hline N_1 & L_2 & 0 & 0 \\ N_1 & 0 & 0 & L_1 \\ N_2 & L_1 & 0 & 0 \end{array} \Rightarrow \begin{array}{c|cccccc} & N_1 & N_1 & N_2 & L_1 & L_1 & L_2 \\ \hline N_1 & 1 & 0 & 0 & 0 & 0 & 1 \\ N_1 & 0 & 0 & 1 & 0 & 1 & 0 \\ N_2 & 1 & 0 & 0 & 1 & 0 & 0 \\ L_1 & 1 & 0 & 0 & 0 & 0 & 0 \\ L_1 & 0 & 0 & 1 & 0 & 0 & 0 \\ L_2 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}$$

The aforementioned explanation was for directed graphs. But the identical preprocessing may be applied to undirected graphs. The difference from the case of directed graphs is that the adjacency matrix becomes diagonally symmetric. The adjacency matrix of Figure 12.6a is represented in Figure 12.6b.

For the sake not only of efficiency in memory consumption but also of efficiency in computations with graphs, we define a code representation of an adjacency matrix as follows. In the case of an undirected graph, the code of an adjacency matrix X_k, that is, $code(X_k)$, is represented by a binary number. It is obtained by scanning the upper triangular elements of the matrix along each column until diagonal elements (for an

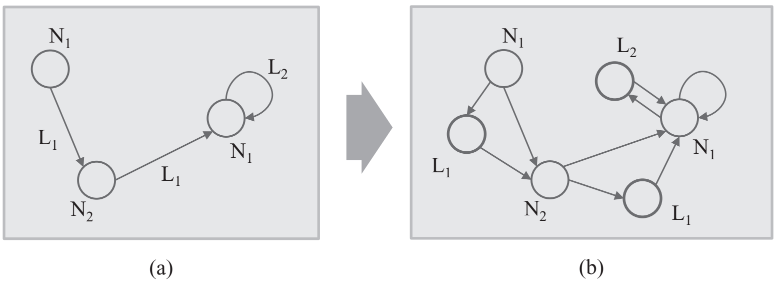
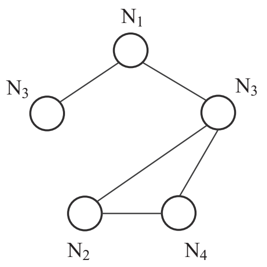


Figure 12.5. Preprocessing of labeled links and self-looped nodes in a graph.

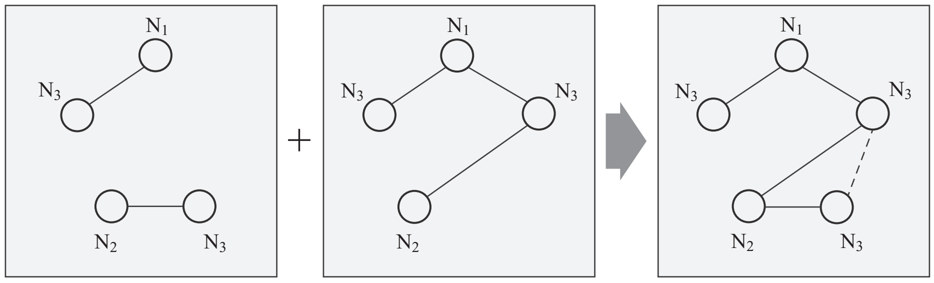


(a)

$$\begin{array}{c|ccccc} & N_1 & N_2 & N_3 & N_3 & N_4 \\ \hline N_1 & 0 & 0 & 1 & 1 & 0 \\ N_2 & 0 & 0 & 0 & 1 & 1 \\ N_3 & 1 & 0 & 0 & 0 & 0 \\ N_3 & 1 & 1 & 0 & 0 & 1 \\ N_4 & 0 & 1 & 0 & 1 & 0 \end{array}$$

(b)

Figure 12.6. Undirected graph representations.



(a) (b) (c)

$$\mathbf{X_4} = \begin{array}{c|cccc} & N_1 & N_2 & N_3 & N_3 \\ \hline N_1 & 0 & 0 & 0 & 1 \\ N_2 & 0 & 0 & 1 & 0 \\ N_3 & 0 & 1 & 0 & 1 \\ N_3 & 1 & 0 & 1 & 0 \end{array} \qquad \mathbf{Y_4} = \begin{array}{c|cccc} & N_1 & N_2 & N_3 & N_3 \\ \hline N_1 & 0 & 0 & 1 & 1 \\ N_2 & 0 & 0 & 0 & 1 \\ N_3 & 1 & 0 & 0 & 0 \\ N_3 & 1 & 1 & 0 & 0 \end{array} \Rightarrow \mathbf{Z_5} = \begin{array}{c|ccccc} & N_1 & N_2 & N_3 & N_3 & N_3 \\ \hline N_1 & 0 & 0 & 1 & 0 & 1 \\ N_2 & 0 & 0 & 0 & 1 & 1 \\ N_3 & 1 & 0 & 0 & 0 & 0 \\ N_3 & 0 & 1 & 0 & 0 & ? \\ N_3 & 1 & 1 & 0 & ? & 0 \end{array}$$

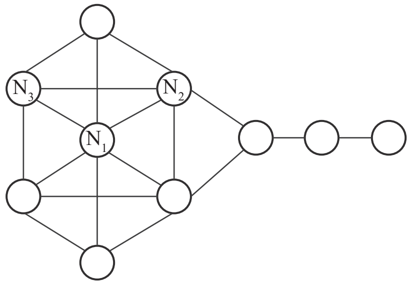
(d) Matrix representation of a join operation

Figure 12.7. An example of join operation between graphs (a) and (b).

undirected graph). For example, the code of the adjacency matrix in Figure 12.6b is given by

$$Code\,(X_k) = 0101100101$$

A variety of operations is defined in graph theory, and corresponding algorithms are developed to efficiently perform these operations. The algorithms work with graphical, matrix, or code representations of graphs. One of the very important operations is the joining of two graphs to form a new, more complex graph. This operation is used in many graph-mining algorithms, including frequent-pattern mining in graphs. The join operation is demonstrated through the example depicted in Figure 12.7. The



	Degree	Closeness
N_1	6	15/9 = 1.66
N_2	5	14/9 = 1.55
N_3	4	17/9 = 1.88

Figure 12.8. Degree and closeness parameters of the graph.

examples of the adjacency matrices X_4, Y_4, and Z_5 are given in (d) representing the graphs at (a), (b), and (c).

Graph analysis includes a number of parameters that describe the important characteristics of a graph, and they are used as fundamental concepts in developing graph-mining algorithms. Over the years, graph-mining researchers have introduced a large number of *centrality indices*, measures of the importance of the nodes in a graph according to one criterion or another.

Perhaps the simplest centrality measure is *degree*, which is the number of links for a given node. Degree is a measure in some sense of the "popularity" of a node in the presented graph. Nodes with a high degree are considered to be more central. However, this weights a node only by its immediate neighbors and not by, for example, its two-hop and three-hop neighbors. A more sophisticated centrality measure is *closeness*, which is the mean geodesic (i.e., shortest path) distance between a vertex and all other vertices reachable from it. Examples of computations for both measures are given in Figure 12.8. Closeness can be regarded as a measure of how long it will take for information to spread from a given node to others in the graph. Nodes that have a low distance to all other nodes in the graph have high closeness centrality.

Another important class of centrality measures is the class of *betweenness* measures. Betweenness is a measure of the extent to which a vertex lies on the paths between others. The simplest and most widely used betweenness measure is *shortest path betweenness*, or simply *betweenness*. The betweenness of a node *i* is defined to be the fraction of shortest paths between any pair of vertices in a graph that passes through node *i*. This is, in some sense, a measure of the influence a node has over the spread of connections through the network. Nodes with high betweenness values occur on a larger number of shortest paths and are presumably more important than nodes with low betweenness. The parameter is costly to compute especially when the graphs are complex with a large number of nodes and links. Currently, the fastest known algorithms require $O(n^3)$ time complexity and $O(n^2)$ space complexity, where n is the number of nodes in the graph.

Illustrative examples of centrality measures and their interpretations are given in Figure 12.9. Node *X* has importance because it bridges the structural hole between the two clusters of interconnected nodes. It has the highest betweenness measure compared with all the other nodes in the graph. Such nodes get lots of brokerage opportunities and can control the flow in the paths between subgraphs. On the other hand, node *Y* is

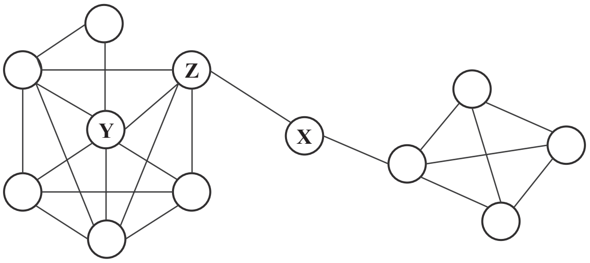


Figure 12.9. Different types of a node's importance in a graph.

in the middle of a dense web of nodes that provides easy, short path access to neighboring nodes; thus, *Y* also has a good central position in the subgraph. This characteristic of the node Y is described with the highest degree measure.

The vertex-betweenness index reflects the amount of control exerted by a given vertex over the interactions between the other vertices in the graph. The other approach to measure betweenness is to concentrate on links in the graph instead of nodes. Edge-betweenness centrality is related to the frequency of an edge placed on the shortest paths between all pairs of vertices. The betweenness centrality of an edge in a network is given by the sum of the edge-betweenness values for all pairs of nodes in the graph going through the given edge. The edges with highest betweenness values are most likely to lie between subgraphs, rather than inside a subgraph. Consequently, successively removing edges with the highest edge-betweenness will eventually isolate subgraphs consisting of nodes that share connections only with other nodes in the same subgraph. This gives the edge-betweenness index a central role in graph-clustering algorithms where it is necessary to separate a large graph into smaller highly connected subgraphs. Edge- (also called link-) betweenness centrality is traditionally determined in two steps:

1. Compute the length and number of shortest paths between all pairs of nodes through the link.
2. Sum all link dependencies.

The overall betweenness centrality of a link *v* is obtained by summing up its partial betweenness values for this link, calculated using the graph transformation on breadth first strategy from each node. For example, at the beginning it is given a graph in Figure 12.10a and it is necessary to find link betweenness measures for all links in the graph. In the first step, we build a "modified" graph that starts with node A, and specifies all links in the graph, layer by layer: first neighbors, second neighbors, and so on. The resulting graph is given in Figure 12.10b. This graph is the starting point for *partial* betweenness computation. The total betweeness will be the sum of partial scores obtained for transformed graphs with root nodes A to K. The process with each transformed graph in (b) consists of a forward phase and a backward phase, and will be illustrated with activities on the graph in Figure 12.10b. In the forward phase, the count of shortest paths from A to all other nodes of the network is determined. The

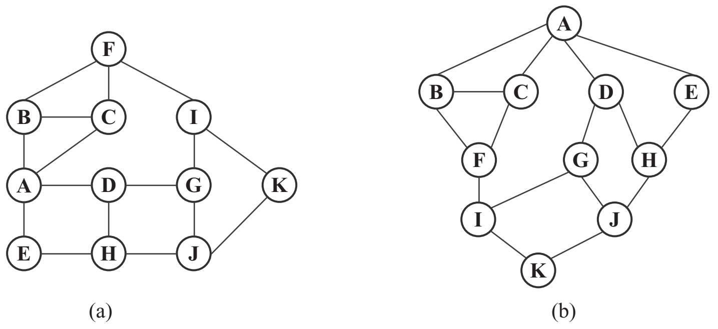


Figure 12.10. Preparatory phase for link-betweenness computation. (a) Initial graph; (b) transformed graph with the root in node A.

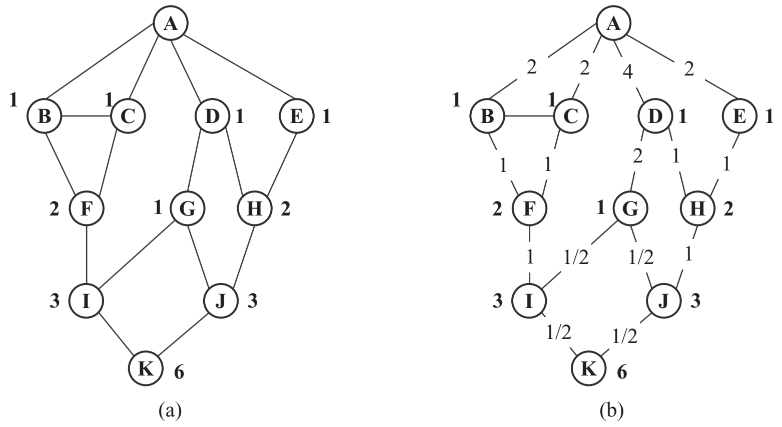


Figure 12.11. Computation of the partial link-betweenness measure. (a) Forward phase; (b) backward phase.

computation is performed iteratively, layer by layer. For example, the number of shortest paths from the initial node A to the node I is computed based on the number of shortest paths from node A to the nodes F and G.

The result of the completed forward phase is given in Figure 12.11a. Each node is labeled with the number of shortest paths from the root node A. For example, the node J has three shortest paths, two of them through the node H (ADHJ and AEHJ) and one through the node G (ADGJ).

The backward phase starts from the bottom of the layered graph structure, in our example, from node K. If there are multiple paths from the given node up, count betweenness measure of each link fractionally. The proportion is determined by the number of shortest paths to these nodes on the previous layer. What is the amount we are splitting between these links? The amount is defined as *1 + sum of all betweenness measures entering into the node from below*. For example, from node K there are two paths toward nodes I and J, and because both nodes have the same number of shortest paths (3), the amount we are splitting is 1 + 0 = 1, and the partial betweenness measure for links IK and JK is 0.5. In a similar way, we may compute betweennes measures for node G. Total betweenness value for splitting is 1 + 0.5 + 0.5 = 2. There is only one node up; it is D, and the link-betweenness measure for GD is 2.

When we compute betweenness measures for all links in the graph, the procedure should be repeated for the other nodes in the graph, such as the root nodes, until each node of the network is explored. Finally, all partial link scores determined for different graphs should be added to determine the final link-betweenness score.

Graph-mining applications are far more challenging to implement because of the additional constraints that arise from the structural nature of the underlying graph. The problem of frequent pattern mining has been widely studied in the context of mining transactional data. Recently, the techniques for frequent-pattern mining have also been extended to the case of graph data. This algorithm attempts to find interesting or commonly occurring subgraphs in a set of graphs. Discovery of these patterns may be the sole purpose of the systems, or the discovered patterns may be used for graph classification or graph summarization. The main difference in the case of graphs is that the process of determining support is quite different. The problem can be defined in different ways, depending upon the application domain. In the first case, we have a group of graphs, and we wish to determine all the patterns that support a fraction of the corresponding graphs. In the second case, we have a single large graph, and we wish to determine all the patterns that are supported at least a certain number of times in this large graph. In both cases, we need to account for the isomorphism issue in determining whether one graph is supported by another or not. However, the problem of defining the support is much more challenging if overlaps are allowed between different embeddings. Frequently occurring subgraphs in a large graph or a set of graphs could represent important motifs in the real-world data.

Apriori-style algorithms can be extended to the case of discovering frequent subgraphs in graph data, by using a similar level-wise strategy of generating (k + 1) candidates from k-patterns. Various measures to mine substructure frequencies in graphs are used similarly in conventional data mining. The selection of the measures depends on the objective and the constraints of the mining approach. The most popular measure in graph-based data mining is a “support” parameter whose definition is identical with that of market-basket analysis. Given a graph data set D, the support of the subgraph Gs, sup(G_s), is defined as

$$\sup(G_s) = \frac{\text{number of graphs including } G_s \text{ in } D}{\text{total number of graphs in } D}$$

By specifying a “minimum support” value, subgraphs Gs, whose support values are above threshold, are mined as candidates or components of candidates for maximum frequent subgraphs. The main difference in an a priori implementation is that we need to define the join process of two subgraphs a little differently. Two graphs of size k can be joined if they have a structure of size (k − 1) in common. The *size of this structure* could be defined in terms of either nodes or edges. The algorithm starts by finding all

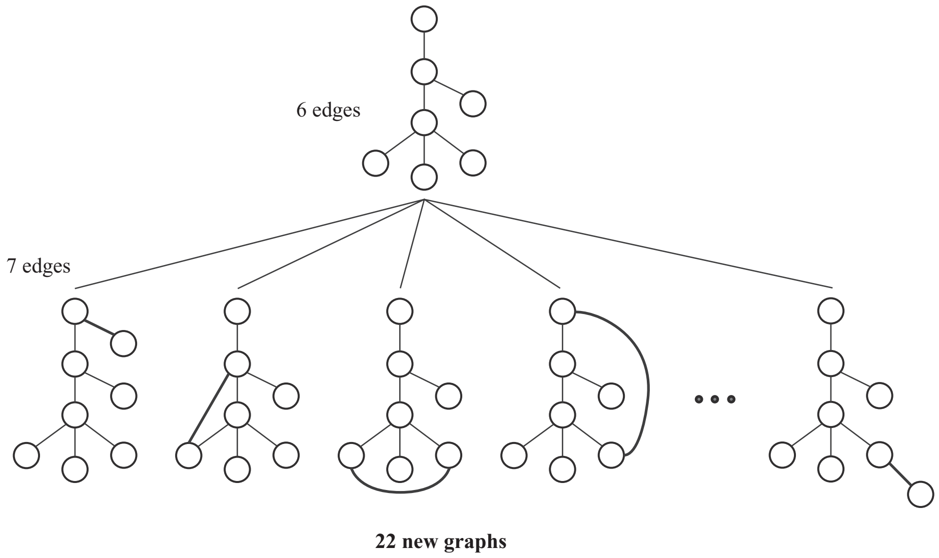


Figure 12.12. Free extensions in graphs.

frequent single- and double-link subgraphs. Then, in each iteration, it generates candidate subgraphs by expanding the subgraphs found in the previous iteration by one edge. The algorithm checks how many times the candidate subgraph with the extension occurs within an entire graph or set of graphs. The candidates whose frequency is below a user-defined level are pruned. The algorithm returns all subgraphs occurring more frequently than the given threshold. A naïve approach of a subgraph extension from kk − 1 size to size k is computationally very expensive as illustrated in Figure 12.12. Therefore, the candidate generation of a frequently induced subgraph is done with some constraints. Two frequent graphs are joined only when the following conditions are satisfied to generate a candidate of frequent graph of size k + 1. Let X_k and Y_k be adjacency matrices of two frequent graphs $G(X_k)$ and $G(Y_k)$ of size k. If both $G(X_k)$ and $G(Y_k)$ have equal elements of the matrices except for the elements of the *k*th row and the *k*th column, then they may be joined to generate Z_{k+1} as an adjacency matrix for a candidate graph of size k + 1.

$$X_k = \begin{pmatrix} X_{k-1} & x_1 \\ x_2^T & 0 \end{pmatrix}, Y_k = \begin{pmatrix} X_{k-1} & y_1 \\ y_2^T & 0 \end{pmatrix}, Z_{k+1} = \begin{pmatrix} X_{k-1} & x_1 & y_1 \\ x_2^T & 0 & z_{k,k+1} \\ y_2^T & z_{k+1,k} & 0 \end{pmatrix}$$

In this matrix representations, X_{k-1} is the common adjacency matrix representing the graph whose size is kk − 1, while x_i and y_i (i = 1, 2) are (kk − 1) × 1 column vectors. These column vectors represent the differences between two graphs prepared for the join operation.

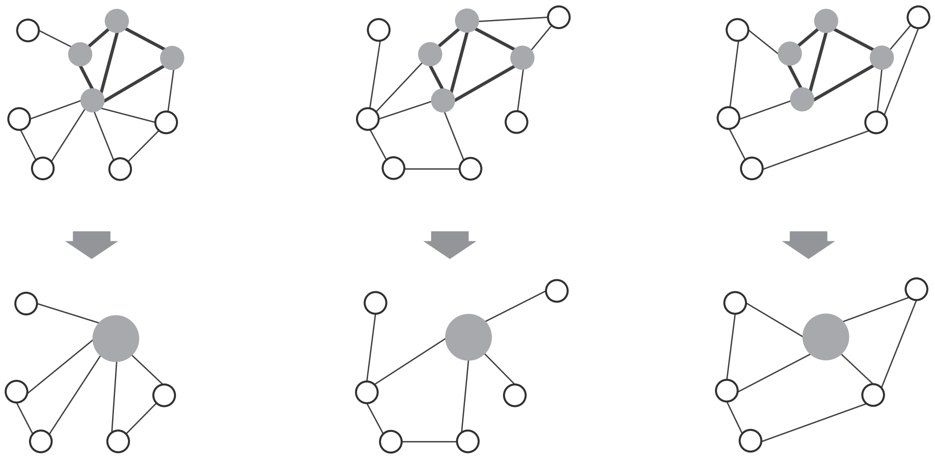


Figure 12.13. Graph summarization through graph compression.

The process suffers from computational intractability when the graph size becomes too large. One problem is subgraph *isomorphism,* with NP complexity, as a core step in graph matching. Also, all frequent patterns may not be equally relevant in the case of graphs. In particular, patterns that are highly connected (which means dense subgraphs) are much more relevant. This additional analysis requires more computations. One possible application of discovering frequent subgraphs is a summarized representation of larger, complex graphs. After extracting common subgraphs, it is possible to simplify large graphs by condensing these subgraphs into new nodes. An illustrative example is given in Figure 12.13, where a subgraph of four nodes is replaced in the set of graphs with a single node. The resulting graphs represent summarized representation of the initial graph set.

In recent years, significant attention has focused on studying the structural properties of networks such as the WWW, online social networks, communication networks, citation networks, and biological networks. Across these large networks, an important characteristic is that they can be characterized by the nature of the underlying graphs and subgraphs, and clustering is an often used technique for miming these large networks. The problem of graph clustering arises in two different contexts: a single large graph or large set of smaller graphs. In the first case, we wish to determine dense node clusters in a *single large graph* minimizing the intercluster similarity for a fixed number of clusters. This problem arises in the context of a number of applications such as graph partitioning and the minimum-cut problem. The determination of dense regions in the graph is a critical problem from the perspective of a number of different applications in social networks and Web-page summarization. Top-down clustering algorithms are closely related to the concept of *centrality analysis* in graphs where central nodes are typically key members in a network that is well connected to other members of the community. Centrality analysis can also be used in order to determine the central points in information flows. Thus, it is clear that the same kind of structural-analysis algorithm

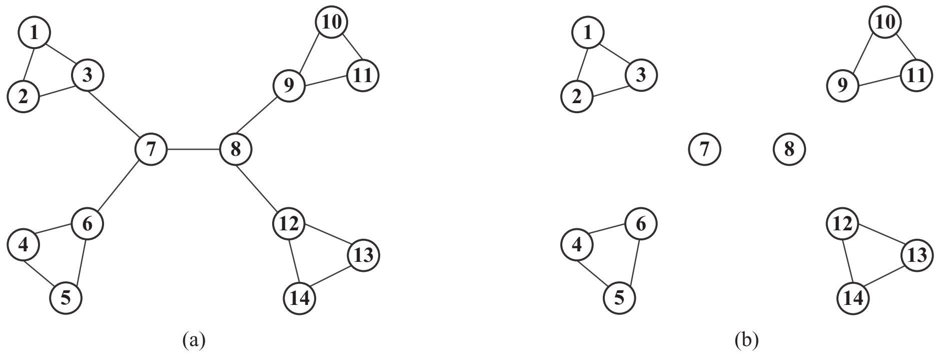


Figure 12.14. Graph clustering using betweenness measure. (a) Initial graph; (b) subgraphs after elimination of links with maximum betweenness.

can lead to different kinds of insights in graphs. For example, if the criterion for separating a graph into subgraphs is a maximum measure of link betweenness, then the graph in Figure 12.14a may be transformed into six subgraphs as presented in Figure 12.14b. In this case the maximum betweenness of 49 was for the link (7, 8), and elimination of this link defines two clusters on the highest level of hierarchy. The next value of betweenness, 33, was found for links (3,7), (8, 9), (6, 7), and (8, 12). After elimination of these links on the second level of hierarchy, the graph is decomposed into six dense subgraphs of clustered nodes.

The second case of cluster analysis assumes multiple graphs, each of which may possibly be of modest size. These large number of graphs need to be clustered based on their underlying structural behavior. The problem is challenging because of the need to match the structures of the underlying graphs, and these structures are used for clustering purposes. The main idea is that we wish to cluster graphs as objects, and the distance between graphs is defined based on a structural similarity function such as the edit distance. This clustering approach makes it an ideal technique for applications in areas such as scientific-data exploration, information retrieval, computational biology, Web-log analysis, forensics analysis, and blog analysis.

Link analysis is an important field that has received a lot of attention recently when advances in information technology enabled mining of extremely large networks. The basic data structure is still a graph, only the emphasis in analysis is on links and their characteristics: labeled or unlabeled, directed or undirected. There is an inherent ambiguity with respect to the term "link" that occurs in many circumstances, but especially in discussions with people whose background and research interests are in the database community. In the database community, especially the subcommunity that uses the well-known entity-relationship (ER) model, a "link" is a connection between two records in two different tables. This usage of the term "link" in the database community differs from that in the intelligence community and in the artificial intelligence (AI) research community. Their interpretation of a "link" typically refers to some real world connection between two entities. Probably the most famous example of exploiting link structure in the graph is the use of links to improve information retrieval results. Both, the well-known PageRank measure and hubs, and authority scores are based on the link structure of the Web. Link analysis techniques are used in law enforcement, intelligence analysis, fraud detection, and related domains. It is sometimes described using the metaphor of "connecting the dots" because link diagrams show the connections between people, places, events, and things, and represent invaluable tools in these domains.

12.2 TEMPORAL DATA MINING

Time is one of the essential natures of data. Many real-life data describe the property or status of some object at a particular time instance. Today time-series data are being generated at an unprecedented speed from almost every application domain, for example, daily fluctuations of stock market, traces of dynamic processes and scientific experiments, medical and biological experimental observations, various readings obtained from sensor networks, Web logs, computer-network traffic, and position updates of moving objects in location-based services. Time series or, more generally, temporal sequences, appear naturally in a variety of different domains, from engineering to scientific research, finance, and medicine. In engineering matters, they usually arise with either sensor-based monitoring, such as telecommunication control, or log-based systems monitoring. In scientific research they appear, for example, in spatial missions or in the genetics domain. In health care, temporal sequences have been a reality for decades, with data originated by complex data-acquisition systems like electrocardiograms (ECGs), or even simple ones like measuring a patient's temperature or treatment effectiveness. For example, a supermarket transaction database records the items purchased by customers at some time points. In this database, every transaction has a time stamp in which the transaction is conducted. In a telecommunication database, every signal is also associated with a time. The price of a stock at the stock market database is not constant, but changes with time as well.

Temporal databases capture attributes whose values change with time. Temporal data mining is concerned with data mining of these large data sets. Samples related with the temporal information present in this type of database need to be treated differently from static samples. The accommodation of time into mining techniques provides a window into the temporal arrangement of events and, thus, an ability to suggest cause and effect that are overlooked when the temporal component is ignored or treated as a simple numeric attribute. Moreover, temporal data mining has the ability to mine the behavioral aspects of objects as opposed to simply mining rules that describe their states at a point in time. Temporal data mining is an important extension as it has the capability of mining activities rather than just states and, thus, inferring relationships of contextual and temporal proximity, some of which may also indicate a cause–effect association.

Temporal data mining is concerned with data mining of large sequential data sets. By sequential data, we mean data that are ordered with respect to some index. For example, a time series constitutes a popular class of sequential data where records are indexed by time. Other examples of sequential data could be text, gene sequences, protein sequences, Web logs, and lists of moves in a chess game. Here, although there is no notion of time as such, the ordering among the records is very important and is central to the data description/modeling. Sequential data include:

1. *Temporal Sequences.* They represent ordered series of nominal symbols from a particular alphabet (e.g., a huge number of relatively short sequences in Web-log files or a relatively small number of extremely long gene expression sequences). This category includes ordered but not time stamped collections of samples. The sequence relationships include before, after, meet, and overlap.
2. *Time Series.* It represents a time-stamped series of continuous, real-valued elements (e.g., a relatively small number of long sequences of multiple sensor data or monitoring recordings from digital medical devices). Typically, most of the existing work on time series assumes that time is discrete. Formally, time-series data are defined as a sequence of pairs $T = ([p_1, t_1], [p_2, t_2], \ldots, [p_n, t_n])$, where $t_1 < t_2 < \ldots < t_n$. Each p_i is a data point in a d-dimensional data space, and each t_i is the time stamp at which p_i occurs. If the sampling rate of a time series is constant, one can omit the time stamps and consider the series as a sequence of d-dimensional data points. Such a sequence is called the raw representation of the time series.

Traditional analyses of temporal data require a statistical approach because of noise in raw data, missing values, or incorrect recordings. They include (1) long-term trend estimation, (2) cyclic variations, for example, business cycles, (3) seasonal patterns, and (4) irregular movements representing outliers. Examples are given in Figure 12.15. The discovery of relations in temporal data requires more emphasis in a data-mining

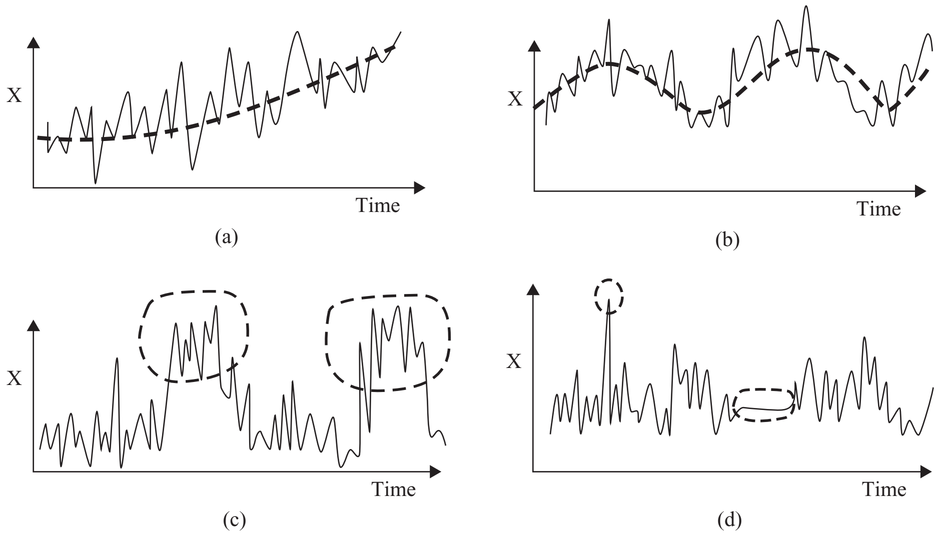


Figure 12.15. Traditional statistical analyses of time series. (a) Trend; (b) cycle; (c) seasonal; (d) outliers.

process on the following three steps: (1) the representation and modeling of the data sequence in a suitable form; (2) the definition of similarity measures between sequences; and (3) the application of variety of new models and representations to the actual mining problems.

12.2.1 Temporal Data Representation

The representation of temporal data is especially important when dealing with large volumes of data since direct manipulation of continuous, high-dimensional data in an efficient way is extremely difficult. There are a few ways this problem can be addressed.

Original Data or with Minimal Preprocessing Use data as they are without or with minimal preprocessing. We preserve the characteristics of each data point when a model is built. The main disadvantage of this process is that it is extremely inefficient to build data-mining models with millions of records of temporal data, all of them with different values.

Windowing and Piecewise Approximations There is well-known psychological evidence that the human eye segments smooth curves into piecewise straight lines. Based on this theory, there are a number of algorithms for segmenting a curve that represents a time series. Figure 12.16 shows a simple example where it replaces the original nonlinear function with several piecewise linear functions. As shown in the figure, the initial real-valued elements (time series) are partitioned into several segments. Finding the required number of segments that best represent the original sequence is not trivial. An easy approach is to predefine the number of segments. A more realistic approach may be to define them when a change point is detected in the original sequence. Another technique based on the same idea segments a sequence by iteratively merging two similar segments. Segments to be merged are selected based on the squared-error minimization criteria. Even though these methods have the advantage of ability to reduce the impact of noise in the original sequence, when it comes to

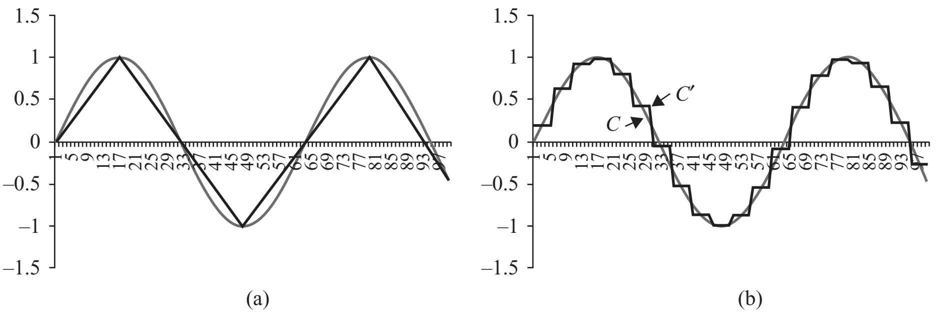


Figure 12.16. Simplified representation of temporal data. (a) Piecewise linear approximation; (b) piecewise aggregate approximation.

real-world applications (e.g., sequence matching) differences in amplitudes (scaling) and time-axis distortions are not addressed easily.

To overcome these drawbacks, *Piecewise Aggregate Approximation* (PAA) technique was introduced. It approximates the original data by segmenting the sequences into same-length sections and recording the mean value of these sections. A time-series **C** of length n is represented as $C = \{c_1, c_2, \ldots, c_n\}$. The task is to represent C as C' in a w-dimensional space ($w < n$) by mean values of c_is in w equal-sized segments. The ith element of C' is calculated as a mean of all values in the segment:

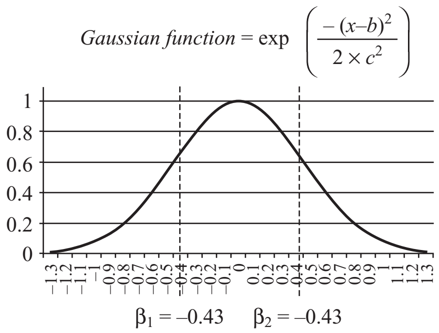
$$C'_i = \sum_{j=w\times(i-1)+1}^{w\times i} c^i_j, 1 \le i \le \textit{the number of segments}$$

For example, if the original sequence is C = {−2, −4, −3, −1, 0, 1, 2, 1, 1, 0}, where n = |C| = 10, and we decide to represent C in two sections of the same length, then

$$C' = \{mean(-2, -4, -3, -1, 0), mean(1, 2, 1, 1, 0)\}$$
$$C' = \{-2, 1\}$$

Usually, PAA is visualized as a linear combination of box bases functions as illustrated in Figure 12.16b, where a continuous function is replaced with 10 discrete averages for each interval.

A modified PAA algorithm, which is called *Symbolic Aggregate Approximation* (SAX), is proposed with the assumption that the original normalized time series, C, has a Gaussian distribution of PAA values. SAX defines "break points" in the Gaussian curve that will produce equal-sized areas below the curve. Formally, break points are a sorted list of numbers $B = \beta_1, \beta_2, \beta_3, \ldots, \beta_{\alpha-1}$ such that the areas under the Gaussian curve from β_i to β_{i+1} are equal to 1/α, and they are constant. α is a parameter of the methodology representing the number of intervals. These break points can be determined in a statistical table. For example, Figure 12.17 gives the break points for α values from 3 to 10.



(a)

β_i \ a	3	4	5	6	7	8	9	10
β_1	−0.43	−0.67	−0.84	−0.97	−1.07	−1.15	−1.22	−1.28
β_2	0.43	0	−0.25	−0.43	−0.57	−0.67	−0.76	−0.84
β_3		0.67	0.52	0	−0.18	−0.32	−0.43	−0.52
β_4			0.84	0.43	0.18	0	−0.14	−0.25
β_5				0.97	0.57	0.32	0.14	0
β_6					1.07	0.67	0.43	0.25
β_7						1.15	0.76	0.52
β_8							1.22	0.84
β_9								1.28

(b)

Figure 12.17. SAX look-up table. (a) Break points in the Gaussian curve ($b = 0, c^2 = 0.2$) when $\alpha = 3$; (b) SAX look-up table.

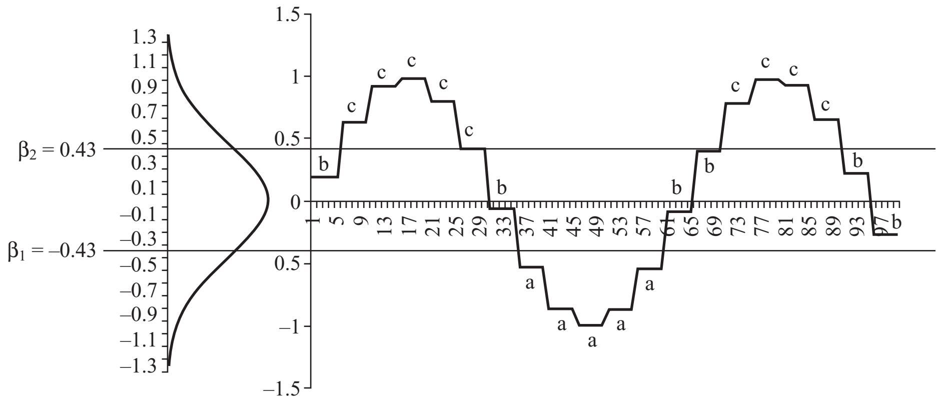


Figure 12.18. Transformation of PAA to SAX.

Once the break points are defined along with the corresponding coding symbols for each interval, the sequence is discretized as follows:

1. Obtain the PAA averages of the time series.
2. All PAA averages in the given interval (β_i, β_{i+1}) are coded with a specific symbol for this interval. For example, if $\alpha = 3$, all PAA averages less than the smallest break point (–0.43) are mapped to "*a*." All PAA coefficients less than the second break point but greater than first breakpoint (–0.43, 0.43) are mapped to "*b*," and all average values larger than the second break point (0.43) are mapped to "*c*." This process is illustrated in Figure 12.18.

It is assumed that the symbols "*a*," "*b*," and "*c*" are approximately equiprobable symbols in the representation of a time series. The original sequence is then represented as a concatenation of these symbols, which is known as a "word." For example, the mapping from PAA (C') to a word C'' is represented as C'' = (*bcccccbaaaaabbccccbb*). The main advantage of the SAX method is that 100 different discrete numerical values in an initial discrete time series C is first reduced to 20 different (average) values using PAA, and then they are transformed into only three different categorical values using SAX.

$$C = \{c_1, c_2, \ldots, c_{100}\} \rightarrow C'' = \{c'_1, c'_2, \ldots, c'_{20}\} \rightarrow C'' = f\{a, b, c\}$$

The proposed approach is intuitive and simple, yet a powerful methodology in a simplified representation of a large number of different values in time series. The method is fast to compute and supports different distance measures. Therefore, it is applicable as a data-reduction step for different data-mining techniques.

Transformation-Based Representations The main idea behind transformation-based techniques is to map the original data into a point of a more manageable domain.

One of the widely used methods is the *Discrete Fourier Transformation* (DFT). It transforms a sequence in the time domain to a point in the frequency domain. This is done by selecting the top-K frequencies and representing each sequence as a point in the K-dimensional space. One important property that is worth noting is that Fourier coefficients do not change under the shift operation. One problem with DFT is that it misses the important feature of time localization. To avoid this problem *Piecewise Fourier Transform* was proposed, but the size of the pieces introduced new problems. Large pieces reduce the power of multi-resolution, while modeling of low frequencies with small pieces does not always give expected representations. The *Discrete Wavelet Transformation* (DWT) has been introduced to overcome the difficulties in DFT. The DWT transformation technique, analogously to fast Fourier transformation, turns a discrete vector of function values with the length N into a vector of N wavelet coefficients. Wavelet transformation is a linear operation and it is usually implemented as a recursion. The advantage of using DWT is its ability for multi-resolution representation of signals. It has the time-frequency localization property. Therefore, signals represented by wavelet transformations bear more information than that of DFT.

In some applications we need to retrieve objects from a database of certain shapes. Trends can often be reflected by specifying shapes of interest such as steep peaks, or upward and downward changes. For example, in a stock-market database we may want to retrieve stocks whose closing price contains a *head-and-shoulders* pattern, and we should be able to represent and recognize this shape. Pattern discovery can be driven by a template-based mining language in which the analyst specifies the shape that should be looked for. *Shape Definition Language* (SDL) was proposed to translate the initial sequence with real-valued elements occurring in historical data into a sequence of symbols from a given alphabet. SDL is capable of describing a variety of queries about the shapes found in the database. It allows the user to create his or her own language with complex patterns in terms of primitives. More interestingly, it performs well on approximate matching, where the user cares only about the overall shape of the sequence but not about specific details. The first step in the representation process is defining the alphabet of symbols and then translating the initial sequence to a sequence of symbols. The translation is done by considering sample-to-sample transitions, and then assigning a symbol of the described alphabet to each transition.

A significantly different approach is to convert a sequence into discrete representation by using *clustering*. A sliding window of width *w* is used to generate subsequences from the original sequence. These subsequences are then clustered, considering the pattern similarity between subsequences, using a suitable clustering method, for example, the *k*-nearest neighbor method. A different symbol is then assigned to each cluster. The discrete version of the time series is obtained by using cluster identities corresponding to the subsequence. For example, the original time sequence is defined with integer values given in time: (1, 2, 3, 2, 3, 4, 3, 4, 3, 4, 5, 4, 5) as represented in Figure 12.19a. The Window width is defined by three consecutive values, and samples of primitives are collected through the time series. After simplified clustering, the final set of three "frequent" primitive shapes, representing cluster centroids, is given in Figure 12.19b. Assigning symbolic representation for these shapes, a_1, a_2, and a_3, the final symbolic representation of the series will be (a_3, a_2, a_1, a_1, a_3, a_2).

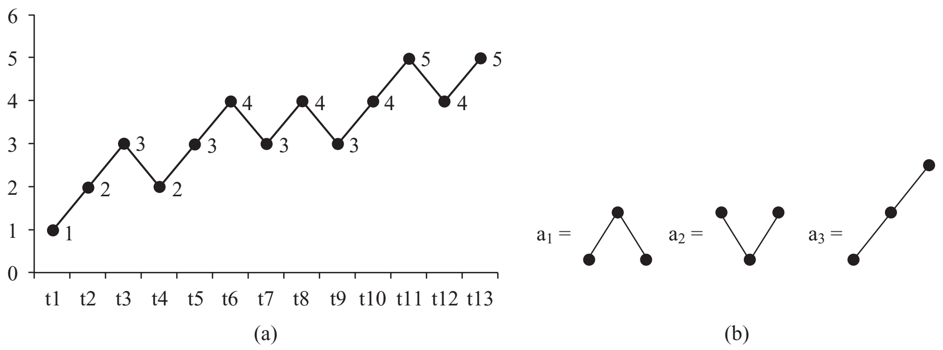


Figure 12.19. Clustering approach in primitive shapes discovery. (a) Time series; (b) primitive shapes after clustering.

12.2.2 Similarity Measures between Sequences

The individual elements of the sequences may be vectors of real numbers (e.g., in applications involving speech or audio signals) or they may be symbolic data (e.g., in applications involving gene sequences). After representing each sequence in a suitable form, it is necessary to define a similarity measure between sequences, in order to determine if they match. Given two sequences T_1 and T_2, we need to define an appropriate similarity function *Sim*, which calculates the closeness of the two sequences, denoted by *Sim*(T_1, T_2). Usually, the similarity measure is expressed in terms of inverse distance measure, and for various types of sequences and applications, we have numerous distance measures. An important issue in measuring similarity between two sequences is the ability to deal with outlying points, noise in the data, amplitude differences causing *scaling problems*, and the existence of gaps and other time-distortion problems. The most straightforward distance measure for time series is the Euclidean distance and its variants, based on the common Lp-norms. It is used in time-domain continuous representations by viewing each sub-sequence with *n* discrete values as a point in R_n. Besides being relatively straightforward and intuitive, Euclidean distance and its variants have several other advantages. The complexity of evaluating these measures is linear; they are easy to implement, indexable with any access method, and, in addition, are parameter-free. Furthermore, the Euclidean distance is surprisingly competitive with other more complex approaches, especially if the size of the training set/database is relatively large. However, since the mapping between the points of two time series is fixed, these distance measures are very sensitive to noise and misalignments in time, and are unable to handle local-time shifting, that is, similar segments that are out of phase.

When a sequence is represented as a sequence of discrete symbols of an alphabet, the similarity between two sequences is achieved most of the time by comparing each element of one sequence with the corresponding one in the other sequence. The best known such distance is the longest common subsequence (LCS) similarity, utilizing the search for the LCS in both sequences we are comparing, and normalized with the length of the longer sequence. For example, if two sequences X and Y are given as X = {10, 5, 6, 9, 22, 15, 4, 2} and Y = {6, 5, 10, 22, 15, 4, 2, 6, 8}, then the LCS is

$$\text{LCS (X, Y)} = \{22, 15, 4, 2\}$$

and normalized similarity measure is

$$\text{LCS similarity (X, Y)} = \text{LCS(X, Y)}/\max\{X, Y\} = 4/9$$

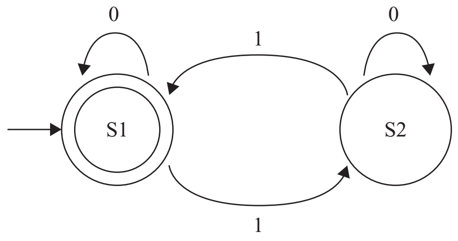
In order to deal with noise, scaling, approximate values, and translation problems, a simple improvement consists of determining the pairs of sequence portions that agree in both sequences after some linear transformations are applied. It consists of determining if there is a linear function *f*, such that one sequence can be approximately mapped into the other. In most applications involving determination of similarity between pairs of sequences, the sequences would be of different lengths. In such cases, it is not possible to blindly accumulate distances between corresponding elements of the sequences. This brings us to the second aspect of sequence matching, namely, sequence alignment. Essentially, we need to properly insert "gaps" in the two sequences or decide which should be corresponding elements in the two sequences. There are many situations in which such symbolic sequence matching problems find applications. For example, many biological sequences such as genes and proteins can be regarded as sequences over a finite alphabet. When two such sequences are similar, it is expected that the corresponding biological entities have similar functions because of related biochemical mechanisms. The approach includes a similarity measure for sequences based on the concept of the edit distance for strings of discrete symbols. This distance reflects the amount of work needed to transform a sequence to another, and is able to deal with different sequences length and gaps existence. Typical edit operations are insert, delete, and replace, and they may be included in the measure with the same or with different weights (costs) in the transformation process. The distance between two strings is defined as the least sum of edit operation costs that needs to be performed to transform one string into another. For example, if two sequences are given: X = {a, b, c, b, d, a, b, c} and Y = { b, b, b, d, b}, the following operations are applied to transform X into Y: delete (a), replace (c,b), delete (a), delete (c). The total number of operations in this case is four, and it represents non-normalized distance measure between two sequences.

12.2.3 Temporal Data Modeling

A *model* is a global, high-level, and often abstract representation of data. Typically, models are specified by a collection of model parameters that can be estimated from a given data set. It is possible to classify models as predictive or descriptive depending on the task they are performing. In contrast to the (global) model structure, a *temporal pattern* is a local model that makes a specific statement about a few data samples in time. Spikes, for example, are patterns in a real-valued time series that may be of interest. Similarly, in symbolic sequences, regular expressions represent well-defined patterns. In bioinformatics, genes are known to appear as local patterns interspersed between chunks of noncoding DNA. Matching and discovery of such patterns are very

Condition	Current State	
	S1	S2
Input 0	S1	S2
Input 1	S2	S1

(a)



(b)

Figure 12.20. Finite-state machine. (a) State-transition table; (b) state-transition diagram.

useful in many applications, not only in bioinformatics. Due to their readily interpretable structure, patterns play a particularly dominant role in data mining. There have been many techniques used to model global or local temporal events. We will introduce only some of the most popular modeling techniques.

Finite State Machine (FSM) has a set of states and a set of transitions. A state may have transitions to other states that are caused by fulfilling some conditions within the state. An FSM must have an initial state, usually drawn with an arrow, and it is a state that provides a starting point of the model. Inputs to the states, in our case representing symbols in a sequence, act as triggers for the transition from one state to another state. An accept state, which is also known as final state, is usually represented by a double circle in a graph representation. The machine reaches the final state when it has performed the procedure successfully, or in our case recognized a sequence pattern. An FSM can be represented using a state-transition table or state-transition diagram. Figures 12.20a,b shows both of these representations for a modeling recognition of a binary number with an even number of ones. FSM does not work very well when the transitions are not precise and does not scale well when the set of symbols for sequence representation is large.

Markov Model (MM) extends the basic idea behind FSM. Both FSM and MM are directed graphs. As with FSM, MM always has a current state. Start and end nodes are drawn for illustrative purposes and need not be present. Unlike in FSM, transitions are not associated with specific input values. Arcs carry a probability value for transition from one state to another. For example, the probability that transition from state "Start" to "S1" is 0.4 and the probability of staying in the "Start" state is 0.6. The sum of the probability values coming out of each node should be 1. MM shows only transitions with probability greater than 0. If a transition is not shown, it is assumed to have a probability of 0. The probabilities are combined to determine the final probability of the pattern produced by the MM. For example, with the MM shown in Figure 12.21, the probability that the MM takes the horizontal path from starting node to S2 is $0.4 \times 0.7 = 0.28$.

MM is derived based on the memoryless assumption. It states that given the current state of the system, the future evolution of the system is independent of its history. MMs have been used widely in speech recognition and natural language processing.

Hidden Markov Model (HMM) is an extension to MM. Similar to MM, HMM consists of a set of states and transition probabilities. In a regular MM, the states are

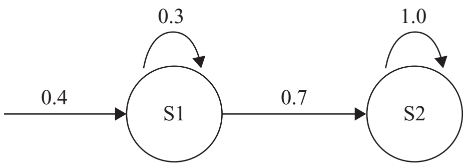


Figure 12.21. A simple Markov Model.

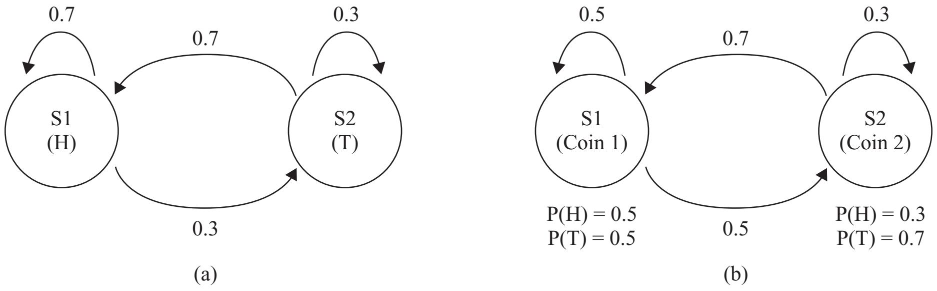


Figure 12.22. Markov Model versus hidden Markov Model. (a) One-coin model; (b) two-coin model.

visible to the observer, and the state-transition probabilities are the only parameters. In HMM, each state is associated with a state-probability distribution. For example, assume that we were given a sequence of events in a coin toss: O = (HTTHTHH), where H = Head and T = Tail. But additional information is necessary. What is not given is the sequence generated with one or two coins. According to the above definitions, Figure 12.22 shows two possible models. Figure 12.22a assumes that only one coin was tossed. We can model this system as an MM with a two-state model, where Head and Tail are the two states with the same initial probabilities. The probability of the sequence O is $P(O) = 0.5 \times 0.7 \times 0.3 \times 0.7 \times 0.3 \times 0.7 \times 0.7 = 0.0108$.

Another possibility for explaining the observed sequence is shown in Figure 12.22b. There are again two states in this model, and each state corresponds to a separate biased coin being tossed. Each state has its own probability distribution of Heads and Tails, and therefore the model is represented as an HMM. Obviously, in this model we have several "paths" to determine the probability of the sequence. In other words, we can start with tossing one or another coin, and continue with this selection. In all these cases, composite probability will be different. In this situation, we may search for the maximum probability of the sequence O in the HMM. HMM may be formalized as a directed graph with *V* vertices and *A* arcs. Set $V = \{v_1, v_2, \ldots, v_n\}$ represents states, and matrix $A = \{a_{ij}\}$ represents transition-probability distribution, where a_{ij} is the transitional probability from state i to state j. Given a set of possible observations $O = \{o_1, o_2, \ldots, o_m\}$ for each state v_i, the probability of seeing each observation in the sequence is given by $Bi = \{o_{i1}, o_{i2}, \ldots, o_{im}\}$. The initial-state distribution is represented as σ, which determines the starting state at time $t = 0$.

12.2.4 Mining Sequences

Temporal data-mining tasks include prediction, classification, clustering, search and retrieval, and pattern discovery. The first four have been investigated extensively in traditional time-series analysis, pattern recognition, and information retrieval. We will concentrate in this text on illustrative examples of algorithms for pattern discovery in large databases, which are of more recent origin and showing wide applicability. The problem of pattern discovery is to find and evaluate all "interesting" patterns in the data. There are many ways of defining what constitutes a pattern in the data and we shall discuss some generic approaches. There is no universal notion for interestingness of a pattern either. However, one concept that is found very useful in data mining is that of frequent patterns. A frequent pattern is one that occurs many times in the data. Much of the data-mining literature is concerned with formulating useful pattern structures and developing efficient algorithms for discovering all patterns that occur frequently in the data.

A pattern is a local structure in the database. In the sequential-pattern framework, we are given a collection of sequences, and the task is to discover sequences of items called sequential patterns that occur in sufficiently many of those sequences. In the frequent episodes analysis, the data set may be given in a single long sequence or in a large set of shorter sequences. An *event sequence* is denoted by $\{(E_1, t_1), (E_2, t_2), \ldots, (E_{n}, t_n)\}$, where E_i takes values from a finite set of event types E, and t_i is an integer denoting the time stamp of the i^{th} event. The sequence is ordered with respect to the time stamps so that $t_i \le t_{i+1}$ for all $i = 1, 2, \ldots, n$. The following is an example event sequence S with 10 events in it:

$$S = \{(A, 2), (B, 3), (A, 7), (C, 8), (B, 9), (D, 11), (C, 12), (A, 13), (B, 14), (C, 15)\}$$

An episode is a partially ordered set of events. When the order among the events of an episode is total, it is called a *serial* episode, and when there is no order at all, the episode is called a *parallel* episode. For example, $(A \rightarrow B \rightarrow C)$ is a three-node serial episode. The arrows in our notation serve to emphasize the total order. In contrast, parallel episodes are somewhat similar to itemsets, and so, we can denote a three-node parallel episode with event types A, B, and C, as (ABC).

An episode is said to *occur* in an event sequence if there exist events in the sequence occurring with exactly the same order as that prescribed in the episode. For instance, in the example the events $(A, 2)$, $(B, 3)$ and $(C, 8)$ constitute an occurrence of the serial episode $(A \rightarrow B \rightarrow C)$ while the events $(A, 7)$, $(B, 3)$, and $(C, 8)$ do not, because for this serial episode to occur, A must occur before B and C. Both these sets of events, however, are valid occurrences of the parallel episode (ABC), since there are no restrictions with regard to the order in which the events must occur for parallel episodes. Let α and β be two episodes. β is said to be a *sub-episode* of α if all the event types in β appear in α as well, and if the partial order among the event types of β is the same as that for the corresponding event types in α. For example, $(A \rightarrow C)$ is a two-node sub-episode of the serial episode $(A \rightarrow B \rightarrow C)$ while $(B \rightarrow A)$ is not. In case of parallel episodes, this order constraint is not required.

The sequential pattern-mining framework may extend the frequent itemsets idea described in the chapter on association rules with temporal order. The database *D of itemsets* is considered no longer just some unordered collection of transactions. Now, each transaction in *D* carries a time stamp as well as a customer ID. Each transaction, as earlier, is simply a collection of items. The transactions associated with a single customer can be regarded as a sequence of itemsets ordered by time, and *D* would have one such transaction sequence corresponding to each customer. Consider an example database with five customers whose corresponding transaction sequences are as follows:

Customer ID	Transaction Sequence
1	$(\{A,B\}\{A,C,D\}\{B,E\})$
2	$(\{D,G\}\ \{A,B,E,H\})$
3	$(\{A\}\{B,D\}\{A,B,E,F\}\{G,H\})$
4	$(\{A\}\{F\})$
5	$(\{A,D\}\ \{B,E,G,H\}\ \{F\})$

Each customer's transaction sequence is enclosed in angular braces, while the items bought in a single transaction are enclosed in round braces. For example, customer 3 made four visits to the supermarket. In his/her first visit he/she bought only item *A*, in the second visit items *B* and *D*, and so on.

The temporal patterns of interest are sequences of itemsets. A sequence *S* of itemsets is denoted by $\{s_1\ s_2 \cdots s_n\}$, where s_j is an itemset. Since *S* has *n* itemsets, it is called an *n*-sequence. A sequence $A = \{a_1\ a_2 \cdots a_n\}$ is said to be *contained in* another sequence $B = \{b_1\ b_2 \cdots b_m\}$ if there exist integers $i_1 < i_2 < \cdots < i_n$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}, \ldots, a_n \subseteq b_{in}$. That is, an *n*-sequence *A* is contained in a sequence *B* if there exists an *n*-length subsequence in *b*, in which each itemset contains the corresponding itemsets of *a*. For example, the sequence {*(A)(BC)*} is contained in {*(AB) (F) (BCE) (DE)*} but not in {*(BC) (AB) (C) (DEF)*}. Further, a sequence is said to be *maximal* in a set of sequences, if it is not contained in any other sequence. In the set of example customer-transaction sequences listed above, all are maximal (with respect to the given set of sequences) except the sequence of customer 4, which is contained in transaction sequences of customers 3 and 5.

The Apriori algorithm described earlier can be used to find frequent sequences, except that there is a small difference in the definition of support. Earlier, the support of an itemset was defined as the fraction of *all* transactions that contained the itemset. Now, the *support* for any arbitrary sequence *A* is the fraction of customer transaction sequences in the database *D*, which contains *A*. For our example database, the sequence {*(D)(GH)*} has a support of 0.4, since it is contained in two out of the five transaction sequences (namely, that of customer 3 and customer 5). The user specifies a minimum support threshold. Any sequence of itemsets with support greater than or equal to the threshold value is called a *large* sequence. If a sequence *A* is large and maximal, then it is regarded as a *sequential pattern*. The process of frequent episode discovery is an

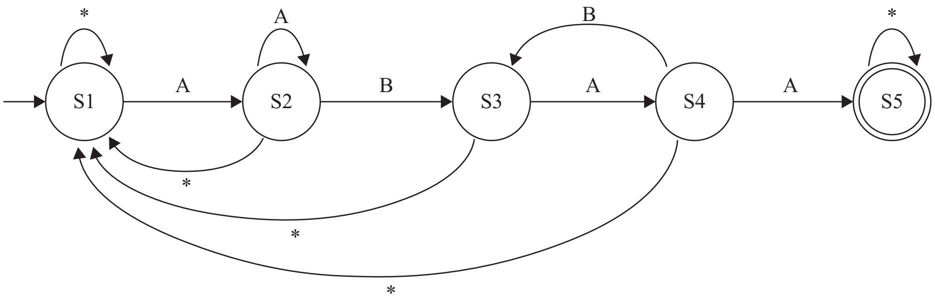


Figure 12.23. FSA for the sequence A → B → A → A. *, any other symbol.

Apriori-style iterative algorithm that starts with discovering frequent one-element sequences. These are then combined to form candidate two-element sequences, and then by counting their frequencies, two-element frequent sequences are obtained. This process is continued till frequent sequences of all lengths are found. The task of a sequence mining is to systematically discover all sequential patterns in database *D*.

Counting frequencies of parallel itemsets is straightforward and described in traditional algorithms for frequent itemsets detection. Counting serial itemsets, on the other hand, requires more computational resources. For example, unlike for parallel itemsets, we need finite-state automata to recognize serial episodes. More specifically, an appropriate *l*-state automaton can be used to recognize occurrences of an *l*-node serial sequence. For example, for the sequence $(A \rightarrow B \rightarrow A \rightarrow A)$, there would be a five-state automaton (FSA) given in Figure 12.23. It transits from its first state on seeing an event of type *A* and then waits for an event of type *B* to transit to its next state, and so on. We need such automata for each episode whose frequency is being counted.

While we described the framework using an example of mining a database of customer transaction sequences for temporal buying patterns, this concept of sequential patterns is quite general and can be used in many other situations as well. Indeed, the problem of motif discovery in a database of protein sequences can also be easily addressed in this framework. Another example is Web-navigation mining. Here the database contains a sequence of Web sites that a user navigates through in each browsing session. Sequential pattern mining can be used to discover sequences of Web sites that are frequently visited. Temporal associations are particularly appropriate as candidates for causal rules' analysis in temporally related medical data, such as in the histories of patients' medical visits. Patients are associated with both static properties, such as gender, and temporal properties, such as age, symptoms, or current medical treatments. Adapting this method to deal with temporal information leads to some different approaches. A possible extension is a new meaning for a typical association rule X ≥ Y. It states now that if X occurs, then Y will occur within time T. Stating a rule in this new form allows for controlling the impact of the occurrence of one event to the other event occurrence, within a specific time interval. In case of the sequential patterns framework some generalizations are proposed to incorporate minimum and maximum time-gap constraints between successive elements of a sequential pattern.

Mining continuous data streams is a new research topic related to temporal data mining that has recently received significant attention. The term "data stream" pertains to data arriving over time, in a nearly continuous fashion. It is often a fast-changing stream with a huge number of multidimensional data (Fig. 12.24). Data are collected close to their source, such as sensor data, so they are usually with a low level of abstraction. In streaming data-mining applications, the data are often available for mining only once, as it flows by. That causes several challenging problems, including how to aggregate the data, how to obtain scalability of traditional analyses in massive, heterogeneous, nonstationary data environment, and how to incorporate incremental learning into a data-mining process. Linear, single-scan algorithms are still rare in commercial data-mining tools, but also still challenged in a research community. Many applications, such as network monitoring, telecommunication applications, stock market analysis, bio-surveillance systems, and distribute sensors depend critically on the efficient processing and analysis of data streams. For example, a frequent itemset-mining algorithm over data stream is developed. It is based on an incremental algorithm to maintain the FP stream, which is a tree data structure to represent the frequent itemsets and their dynamics in time.

Ubiquitous Data Mining (UDM) is an additional new field that defines a process of performing analysis of data on mobile, embedded, and ubiquitous devices. It represents the next generation of data-mining systems that will support the intelligent and time-critical information needs of mobile users and will facilitate "anytime, anywhere" data mining. It is the next natural step in the world of ubiquitous computing. The underlying focus of UDM systems is to perform computationally intensive mining

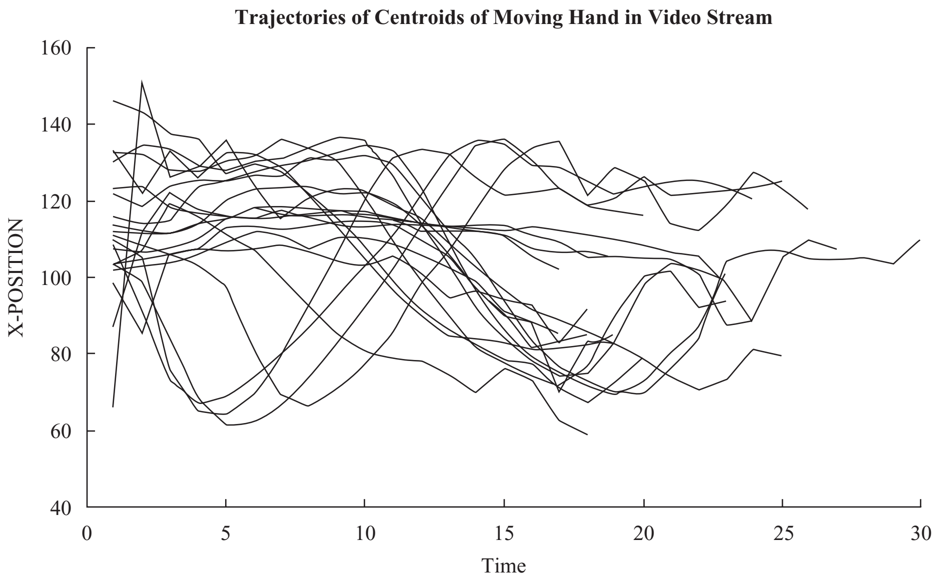


Figure 12.24. Multidimensional streams.

techniques in mobile environments that are constrained by limited computational resources and varying network characteristics. Additional technical challenges are as follows: How to minimize energy consumption of the mobile device during the data mining process; how to present results on relatively small screens; and how to transfer data mining results over a wireless network with a limited bandwidth?

12.3 SPATIAL DATA MINING (SDM)

SDM is the process of discovering interesting and previously unknown but potentially useful information from large spatial data sets. Spatial data carries topological and/or distance information, and it is often organized in databases by spatial indexing structures and accessed by spatial access methods. The applications covered by SDM include geomarketing, environmental studies, risk analysis, remote sensing, geographical information systems (GIS), computer cartography, environmental planning, and so on. For example, in geomarketing, a store can establish its trade area, that is, the spatial extent of its customers, and then analyze the profile of those customers on the basis of both their properties and the area where they live. Simple illustrations of SDM results are given in Figure 12.25, where (a) shows that a fire is often located close to a dry tree and a bird is often seen in the neighborhood of a house, while (b) emphasizes a significant trend that can be observed for the city of Munich, where the average rent decreases quite regularly when moving away from the city. One of the main reasons for developing a large number of spatial data-mining applications is the enormous amount of special data that are collected recently at a relatively low price. High spatial and spectral resolution remote-sensing systems and other environmental monitoring devices gather

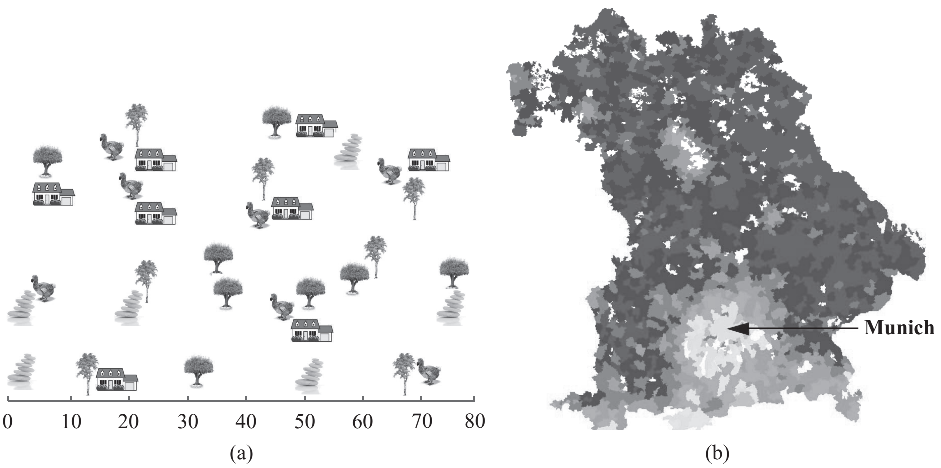


Figure 12.25. Illustrative examples of spatial data-mining results. (a) Example of collocation spatial data mining (Shekhar and Chawla, 2003); (b) average rent for the communities of Bavaria (Ester et al., 1997).

	Traditional Data Mining	Spatial Data Mining
Input	Simple types Explicit relationship	Complex types Implicit relationships
Statistical foundation	Independence of samples	Spatial autocorrelation
Output	Set-based interest measures e.g., classification accuracy	Spatial interest measures e.g., spatial accuracy

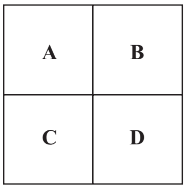
Figure 12.26. Main differences between traditional data mining and spatial data mining.

vast amounts of geo-referenced digital imagery, video, and sound. The complexity of spatial data and intrinsic spatial relationships limits the usefulness of conventional data-mining techniques for extracting spatial patterns.

One of the fundamental assumptions of data-mining analysis is that the data samples are independently generated. However, in the analysis of spatial data, the assumption about the independence of samples is generally false. In fact, spatial data tends to be highly self-correlated. Extracting interesting and useful patterns from spatial data sets is more difficult than extracting corresponding patterns from traditional numeric and categorical data due to the complexity of spatial data types, spatial relationships, and spatial autocorrelation. The spatial attributes of a spatial object most often include information related to spatial locations, for example, longitude, latitude and elevation, as well as shape. Relationships among nonspatial objects are explicit in data inputs, for example, arithmetic relation, ordering, is an instance of, subclass of, and membership of. In contrast, relationships among spatial objects are often implicit, such as overlap, intersect, close, and behind. Proximity can be defined in highly general terms, including distance, direction and/or topology. Also, spatial heterogeneity or the nonstationarity of the observed variables with respect to location is often evident since many space processes are local. Omitting the fact that nearby items tend to be more similar than items situated apart causes inconsistent results in the spatial data analysis. In summary, specific features of spatial data that preclude the use of general-purpose data-mining algorithms are: (1) rich data types (e.g., extended spatial objects), (2) implicit spatial relationships among the variables, (3) observations that are not independent, and (4) spatial autocorrelation among the features (Fig. 12.26).

One possible way to deal with implicit spatial relationships is to materialize the relationships into traditional data input columns and then apply classical data-mining techniques. However, this approach can result in loss of information. Another way to capture implicit spatial relationships is to develop models or techniques to incorporate spatial information into the spatial data-mining process. A concept within statistics devoted to the analysis of spatial relations is called spatial autocorrelation. Knowledge-discovery techniques, which ignore spatial autocorrelation, typically perform poorly in the presence of spatial data.

The spatial relationship among locations in a spatial framework is often modeled via a contiguity matrix. A simple contiguity matrix may represent a neighborhood relationship defined using adjacency. Figure 12.27a shows a gridded spatial framework with four locations, A, B, C, and D. A binary matrix representation of a four-



(a)

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & 1 & 0 \\ B & 1 & 0 & 0 & 1 \\ C & 1 & 0 & 0 & 1 \\ D & 0 & 1 & 1 & 0 \end{array}$$

(b)

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 0.5 & 0.5 & 0 \\ B & 0.5 & 0 & 0 & 0.5 \\ C & 0.5 & 0 & 0 & 0.5 \\ D & 0 & 0.5 & 0.5 & 0 \end{array}$$

(c)

Figure 12.27. Spatial framework and its four-neighborhood contiguity matrix.

neighborhood relationship is shown in Figure 12.27b. The row-normalized representation of this matrix is called a contiguity matrix, as shown in Figure 12.27c. The essential idea is to specify the pairs of locations that influence each other along with the relative intensity of interaction.

SDM consists of extracting knowledge, spatial relationships, and any other properties that are not explicitly stored in the database. SDM is used to find implicit regularities, and relations between spatial data and/or nonspatial data. In effect, a spatial database constitutes a spatial continuum in which properties concerning a particular place are generally linked and explained in terms of the properties of its neighborhood. In this section, we introduce as illustrations of SDM two important characteristics and often used techniques: (1) spatial autoregressive (SAR) modeling, and (2) spatial outliers' detection using variogram-cloud technique.

1. *The SAR model* is a classification technique that decomposes a classifier into two parts, spatial autoregression and logistic transformation. Spatial dependencies are modeled using the framework of logistic regression analysis. If the spatially dependent values y_i are related to each other, then the traditional regression equation can be modified as

$$y = \rho W y + X\beta + \varepsilon$$

   where W is the neighborhood relationship contiguity matrix and ρ is a parameter that reflects the strength of the spatial dependencies between the elements of the dependent variable. After the correction term ρWy is introduced, the components of the residual error vector ε are then assumed to be generated from independent and identical standard normal distributions. As in the case of classical regression, the proposed equation has to be transformed via the logistic function for binary dependent variables, and we refer to this equation as the SAR model. Notice that when $\rho = 0$, this equation collapses to the classical regression model. If the spatial autocorrelation coefficient is statistically significant, then SAR will quantify the presence of spatial autocorrelation in the classification model. It will indicate the extent to which variations in the dependent variable (y) are influenced by the average of neighboring observation values.
2. *A spatial outlier* is a spatially referenced object whose nonspatial attribute values differ significantly from those of other spatially referenced objects in its

Sample	X-C	Y-C	AT-1	AT-2	AT-3
S1	1	2	8	4	1
S2	3	4	2	6	4
S3	2	1	4	2	4
S4	5	3	3	2	5
S5	2	2	7	4	2
S6	1	1	6	5	1

(a)

Samples compared	Spatial distance	Distance of samples
S3 – S6	1.0	4.69
S3 – S5	1.0	4.12
S1 – S3	1.41	5.39

(b)

Figure 12.28. An example of a variogram-cloud graph. (a) Spatial data set; (b) a critical sample's relations in a variogram-cloud.

spatial neighborhood. This kind of outlier shows a local instability in values of nonspatial attributes. It represents spatially referenced objects whose nonspatial attributes are extreme relative to its neighbors, even though the attributes may not be significantly different from the entire population. For example, a new house in an old neighborhood of a growing metropolitan area is a spatial outlier based on the nonspatial attribute house age.

A variogram-cloud technique displays data points related by neighborhood relationships. For each pair of samples, the square-root of the absolute difference between attribute values at the locations versus the Euclidean distance between the locations is plotted. In data sets exhibiting strong spatial dependence, the variance in the attribute differences will increase with increasing distance between locations. Locations that are near to one another, but with large attribute differences, might indicate a spatial outlier, even though the values at both locations may appear to be reasonable when examining the dataset nonspatially. For example, the spatial data set is represented with six five-dimensional samples given in Figure 12.28a. Traditional nonspatial analysis will not discover any outliers especially because the number of samples is relatively small. However, after applying a variogram-cloud technique, assuming that the first two attributes are X-Y spatial coordinates, and the other three are characteristics of samples, the conclusion could be significantly changed. Figure 12.29 shows the variogram-cloud for this data set. This plot has some pairs of points that are out of main dense region of common distances.

Computation of spatial distances and distances of samples, as a part of a variogram technique, shows that there is a sample spatially relatively close to a group of other samples (small space distances) but with very high distances in other nonspatial attributes. This is the sample S3, which is spatially close to samples S1, S5, and S6. Coordinates of these samples and corresponding distances are given in Figure 12.28b, selecting S3 as a candidate for an outlier. Visualization of these and other relations between samples through a variogram shows the same results.

12.4 DISTRIBUTED DATA MINING (DDM)

The emergence of tremendous data sets creates a growing need for analyzing them across geographical lines using distributed systems. These developments have created

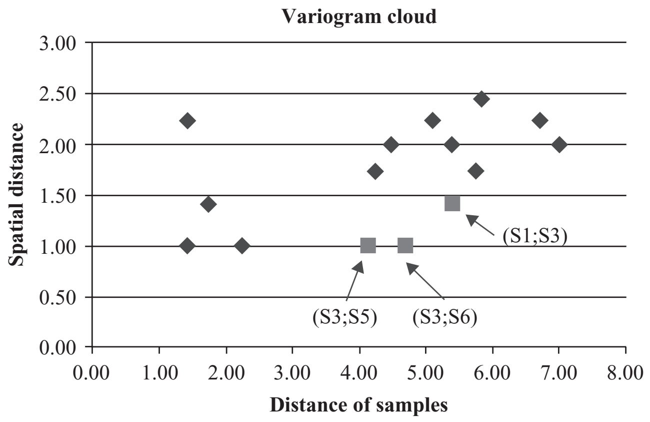


Figure 12.29. A variogram-cloud technique discovers an outlier.

unprecedented opportunities for a large-scale data-driven knowledge discovery, as well as the potential for fundamental gains in scientific and business understanding. Implementations of data-mining techniques on high-performance distributed computing platforms are moving away from centralized computing models for both technical and organizational reasons. In some cases, centralization is hard because it requires these multi-terabyte data sets to be transmitted over very long distances. In others, centralization violates privacy legislation, exposes business secrets, or poses other social challenges. Common examples of such challenges arise in medicine, where relevant data might be spread among multiple parties, in commercial organizations such as drug companies or hospitals, government bodies such as the U.S. Food and Drug Administration, and nongovernment organizations such as charities and public-health organizations. Each organization is bound by regulatory restrictions, such as privacy legislation, or corporate requirements on proprietary information that could give competitors a commercial advantage. Consequently, a need exists for developing algorithms, tools, services, and infrastructure that let us mine data distributed across organizations while preserving privacy.

This shift toward intrinsically distributed, complex environments has prompted a range of new data-mining challenges. The added dimension of distributed data significantly increases the complexity of the data-mining process. Advances in computing and communication over wired and wireless networks have resulted in many pervasive distributed computing environments. Many of these environments deal with different distributed sources of voluminous data, multiple compute nodes, and distributed user community. Analyzing and monitoring these distributed data sources require a new data-mining technology designed for distributed applications. The field of DDM deals with these problems—mining distributed data by paying careful attention to the distributed resources. In addition to data being distributed, the advent of the Internet has led to increasingly complex data, including natural-language text, images, time series, sensor data, and multi-relational and object data types. To further complicate matters, systems with distributed streaming data need incremental or online mining tools that require a complete process whenever a change is made to the underlying data. Data-mining techniques involved in such a complex environment must encounter great dynamics due to changes in the system, and it can affect the overall performance of the system. Providing support for all these features in DDM systems requires novel solutions.

The Web architecture, with layered protocols and services, provides a sound framework for supporting DDM. The new framework embraces the growing trend of merging computation with communication. DDM accepts the fact that data may be inherently distributed among different loosely coupled sites, often with heterogeneous data, and connected by a network. It offers techniques to discover new knowledge through distributed data analysis and modeling using minimal communication of data. Also, interactions in a distributed system need to be implemented in a reliable, stable, and scalable way. Ultimately, systems must be able to hide this technological complexity from users.

Today, the goods that are able to be transacted through e-services are not restricted to real entities such as electronics, furniture, or plane tickets. The Internet and the WWW evolve to include also resources such as software, computation abilities, or useful data sets. These new resources are potentially able to be sold or rented to clients as services for Internet users. Data mining is emerging as intuitively suitable for being delivered as an e-service because the approach reduces the high cost of setting up and maintaining infrastructure of supporting technologies. To efficiently and effectively deliver data mining as a service in the WWW, Web-service technologies are introduced to provide layers of abstractions and standards above existing software systems. These layers are capable of bridging any operating system, hardware platform, or programming language, just as the Web does. The natural extension for these services is grid computing. The grid is a distributed computing infrastructure that enables coordinated resource sharing within dynamic organizations consisting of individuals, institutions, and resources. The main aim of grid computing is to give organizations and application developers the ability to create distributed computing environments that can utilize computing resources on demand. Grid computing can leverage the computing power of a large numbers of server computers, desktop PCs, clusters, and other kinds of hardware. Therefore, it can help increase efficiencies and reduce the cost of computing networks by decreasing data processing time and optimizing resources and distributing workloads. Grid allows users to achieve much faster results on large operations and at lower costs. Recent development and applications show that the grid technology represents a critical infrastructure for high-performance DDM and knowledge discovery. This technology is particularly suitable for applications that typically deal with very a large amount of distributed data such as retail transactions, scientific simulation, or telecommunication data that cannot be analyzed on traditional machines in acceptable times. As the grid is becoming a well-accepted computing infrastructure in science and industry, it provides more general data-mining services, algorithms, and applications. This framework helps analysts, scientists, organizations, and professionals to leverage grid capacity in supporting high-performance distributed computing for solving their data-mining problem in a distributed way. The creation of the so-called *Knowledge Grids* on top of data and computational grids is the condition for meeting the challenges posed by the increasing demand for power and abstractions coming from complex data-mining scenarios in business, science, and engineering.

It is not only that DDM infrastructure is changing by offering new approaches through Web services together with the grid technology. Basic data-mining algorithms also need changes in a distributed environment. Most off-the-shelf data-mining systems are designed to work as a monolithic centralized application. They normally download the relevant data to a centralized location and then perform the data-mining operations. This centralized approach does not work well in many of the emerging distributed, ubiquitous, possibly privacy-sensitive data-mining applications. A primary goal of DDM algorithms is to achieve the same or similar data-mining result as a centralized solution without moving data from their original locations. The distributed approach assumes that local computation is done on each of the sites, and either a central site communicates with each distributed site to compute the global model, or a peer-to-peer architecture is used. In the latter case, individual nodes perform most of the tasks by communicating with neighboring nodes by message passing over an asynchronous network. Illustrative examples are networks of independent and intelligent sensors that are connected to each other in an ad hoc fashion. Some features of a distributed mining scenario are as follows:

- The system consists of multiple independent sites of data and computation.
- Sites exchange their results by communicating with other sites, often through message passing.
- Communication between the sites is expensive and often represents a bottleneck.
- Sites have resource constraints, for example, battery power in distributed sensors systems.
- Sites have privacy and/or security concerns.
- The system should have the ability to efficiently scale up because distributed systems today may consist of millions of nodes.
- The system should have the ability to function correctly in the presence of local site failures, and also missing or incorrect data.

Obviously, the emphasis in DDM algorithms is on local computation and communication. Local algorithms for DDM can be broadly classified under two categories:

- *Exact Local Algorithms.* These algorithms guarantee to always terminate with precisely the same result that would have to be found by a centralized algorithm. Exact local algorithms are obviously more desirable but are more difficult to develop, and in some cases seemingly not possible.
- *Approximate Local Algorithms.* These algorithms cannot guarantee accuracy by centralized solutions. They make a balance between quality of solution and system's responses.

Selection of a type of a local algorithm depends on the data-mining problem and application domain, including the amount of data and their dynamics. In general, approximate approaches are used in cases when the balance between accuracy and efficiency is important, and communications between sites represent a bottleneck. We will illustrate this balance between local computation and communication with a simple approximate algorithm useful in many data-mining applications. For example, if we want to compare the data vectors observed at different sites, the centralized approach will collect these vectors to the central computer and then compare the vectors using whatever metric is appropriate for the domain. DDM technology offers more efficient solutions for the problem using a simple randomized technique.

Vectors $a = (a_1, a_2, \ldots, a_m)$ and $b = (b_1, b_2, \ldots, b_m)$ are given at two distributed sites A and B, respectively. We want to approximate the Euclidean distance between them using a small number of messages and reduced data transfer between sites A and B. Centralized solution requires that one vector is transferred to the other site, that is, *m* components of one vector are transferred. How does one obtain the same result with less than *m* data transfer? Note that the problem of computing the Euclidean distance between a pair of vectors *a* and *b* can be represented as the problem of computing the inner products as follows:

$$d^2(a,b) = (a \bullet a) + (b \bullet b) - 2\,(a \bullet b)$$

where $(a \bullet b)$ represents a inner product between vectors *a* and *b* defined as $\Sigma\, a_i\, b_i$, and $(a \bullet a)$ is a special case of the inner product representing square of the magnitude of the vector *a*. The reader can easily check the previous relation. If, for example, the vectors a and b are $a = (1,2,3)$ and $b = (2,1,2)$, then the Euclidean distance may be calculated as $d^2 = 14 + 9 - 2{\times}10 = 3$. While products $(a \bullet a)$ and $(b \bullet b)$ can be computed locally, and each result is a single value, the core challenge is to develop an algorithm for distributed inner product computation $(a \bullet b)$. A simple, communication-efficient randomized technique for computing this inner product between two vectors observed at two different sites may consist of the following steps:

1. Vectors *a* and *b* are given on two sites, A and B, respectively. Site A sends to the site B a random number generator seed. (*This is only one passed message*.)
2. Both sites A and B cooperatively generate a random matrix R with dimensions $k \times m$, where $k << m$. Each entry in matrix R is generated independently and identically from some fixed distribution with mean 0 and a finite variance.
3. Based on matrix R, sites A and B compute their own local matrix products: ^a = R a and ^b = R b.

   Dimensions of new local vectors ^a and ^b are k, and that means significantly lower than initial lengths of m.
4. Site A sends the resulting vector ^a to the site B. (*This represents k passed messages*.)
5. Site B computes approximate inner product $(a \bullet b) = (\hat{}a^T \bullet \hat{}b)/k$

SITE 1

City	Humidity	Temperature
Louisville	85%	83 °F
Cincinnati	77%	81 °F
Nashville	89%	85 °F

SITE 2

City	Humidity	Temperature
Seattle	67%	73 °F
Miami	77%	91 °F
Huston	56%	95 °F

(a)

SITE 1

Patient ID	Temperature	Hearth rate
1	97 °F	75
2	98.3 °F	68
3	99.9 °F	72
4	101 °F	80

SITE 2

Patient ID	Red cells	White cells
1	4.2×10^3	4×10^9
2	6.2×10^3	7.2×10^9
3	6.7×10^3	6.1×10^9
4	5.1×10^3	4.8×10^9

(b)

Figure 12.30. Horizontally versus vertically partitioned data. (a) Horizontally partitioned data; (b) vertically partitioned data.

So, instead of sending an m-dimensional vector to the other site, the algorithm sends only a (k + 1)-dimensional vector where k << m (k is a user-defined parameter). The inner product of vectors can still be estimated accurately with lower communication load.

In the DDM literature, one of two assumptions is commonly adopted as to how data are distributed across sites: (1) homogeneously or horizontally partitioned, or (2) heterogeneously or vertically partitioned. Both viewpoints assume that the data tables at each distributed site are partitions of a single global table. It is important to stress that the global table viewpoint is strictly conceptual. It is not necessarily assumed that such a table was physically realized and partitioned to form the tables at each site. In the homogeneous case, the global table is horizontally partitioned. The tables at each site are subsets of the global table; they have exactly the same attributes. Figure 12.30a illustrates the homogeneously distributed case using an example from weather data where both tables use the same three attributes. In the heterogeneous case, the table is vertically partitioned where each site contains a subset of columns. That means sites do not have the same attributes. However, samples at each site are assumed to contain a unique identifier to facilitate matching, and Figure 12.30b illustrates this case. The tables at distributed sites have different attributes, and samples are linked through a unique identifier, Patient ID.

DDM technology supports different data-mining tasks including classification, prediction, clustering, market-basket analysis, and outliers' detection. A solution for each of these tasks may be implemented with a variety of DDM algorithms. For example, distributed Apriori has several versions for frequent itemset generation in distributed transactional database. They usually require multiple synchronizations and communication steps. Most of these implementations assume that platforms are homogeneous and therefore the data sets are partitioned evenly among the sites. However, in practice, both the data sets and the processing platforms are more likely to be heterogeneous, running multiple and different systems and tools. This leads to unbalanced data-set distributions and workloads causing additional problems in implementation.

One recent trend is online-mining technology used for monitoring in distributed sensor networks. This is because deployments of large-scale distributed sensor networks are now possible owing to hardware advances and increasing software support. Online data mining, also called *data-stream mining*, is concerned with extracting patterns, detecting outliers, or developing dynamic models of a system's behavior from continuous data streams such as those generated by sensor networks. Because of the massive amount of data and the speed of which the data are generated, many data-mining applications in sensor networks require in-network processing such as aggregation to reduce sample size and communication overhead. Online data mining in sensor networks offers many additional challenges, including:

- limited communication bandwidth,
- constraints on local computing resources,
- limited power supply,
- need for fault tolerance, and
- asynchronous nature of the network.

Obviously, data-mining systems have evolved in a short period of time from stand-alone programs characterized by single algorithms with little support for the entire knowledge-discovery process to integrated systems incorporating several mining algorithms, multiple users, communications, and various and heterogeneous data formats and distributed data sources. Although many DDM algorithms are developed and deployed in a variety of applications, the trend will be illustrated in this book with only one example of a distributed clustering algorithm.

12.4.1 Distributed DBSCAN Clustering

Distributed clustering assumes that samples to be clustered reside on different sites. Instead of transmitting all samples to a central site where we can apply one of the standard clustering algorithms to analyze the data locally, the data are clustered independently on the distributed local sites. Then, in a subsequent step, the central site tries to establish a global clustering model based on the downloaded local models, that is, summarized representatives of local data. Distributed clustering is carried out on two different levels, that is, the local level and the global level (Fig. 12.31). On the local level, all sites carry out clustering independently from each other. Communication with the central site and determining a global model should reflect an optimum trade-off between complexity and accuracy of the algorithm.

Local models consist of a set of representatives for each locally found cluster. A representative is a good approximation for samples residing on the corresponding local site. The local model is transferred to a central site, where the local models are merged in order to form a global model. The representation of local models should be simple

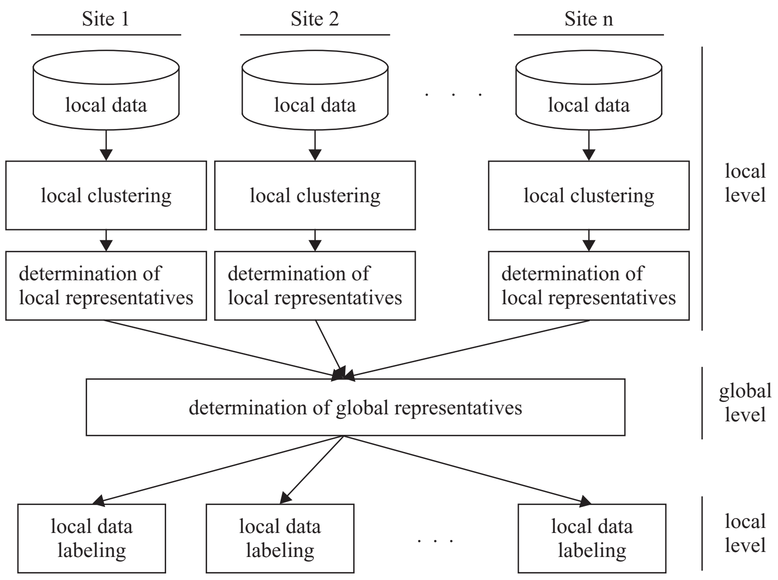


Figure 12.31. System architecture for distributed clustering.

enough so there will be no overload in communications. At the same time, local models should be informative enough to support a high quality of approximate global clustering. The global model is created by analyzing and integrating local representatives. The resulting global clustering is sent back, at the end of the process, to all local sites.

This global-distributed framework may be more precisely specified when we implement a specific clustering algorithm. The density-based clustering algorithm DBSCAN is a good candidate, because it is robust to outliers, easy to implement, supports clusters of different shapes, and allows incremental, online implementation. The main steps of the algorithm are explained in Chapter 9, and the same process is applied locally. To find local clusters, DBSCAN starts with an arbitrary core object *p*, which is not yet clustered and retrieves all objects density reachable from *p*. The retrieval of density-reachable objects is performed in iterations until all local samples are analyzed. After having clustered the data locally, we need a small number of representatives that will describe the local clustering result accurately. For determining suitable representatives of the clusters, the concept of *specific core points* is introduced.

Let C be a local cluster with respect to the given DBSCAN parameters ε and *MinPts*. Furthermore, let $Cor_C \subseteq C$ be the set of core points belonging to this cluster. Then $Scor_C \subseteq C$ is called a *complete set of specific core points of C* iff the following conditions are true:

- $Scor_C \subseteq Cor_C$
- $\forall s_i, s_j \subseteq Scor_C: s_i \notin Neighborhood_\varepsilon (s_j)$
- $\forall c \in Cor_C, \exists s \in Scor_C: c \in Neighborhood_\varepsilon (s)$

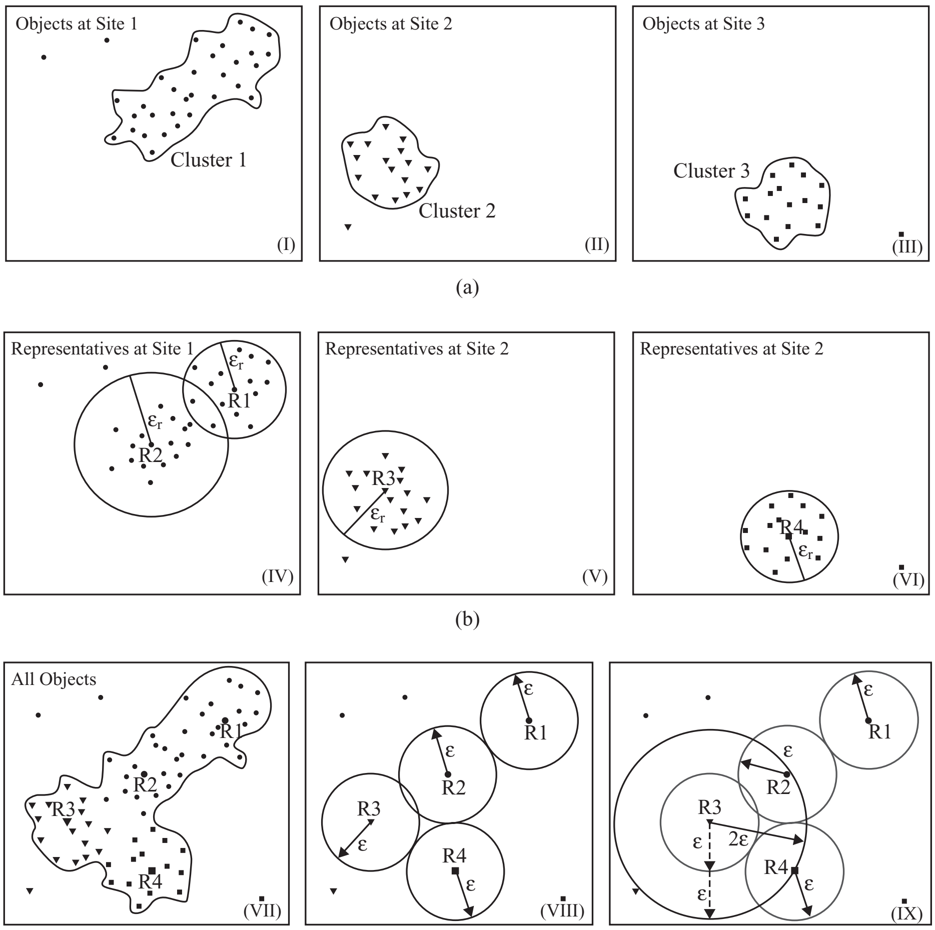


Figure 12.32. Distributed DBSCAN clustering (Januzaj et al., 2003). (a) Local clusters; (b) local representatives; (c) global model with $\varepsilon_{global} = 2\varepsilon_{local}$.

The $Scor_C$ set of points consists of a very small number of *specific core points* that describe the cluster *C*. For example, in Figure 12.32a, sites 2 and 3 have only one *specific core point*, while site 1, because of the cluster shape, has two *specific core points*. To further simplify the representation of local clusters, the number of specific core points, $|Scor_C| = K$, is used as an input parameter for a further local "clustering step" with an adapted version of *K-means*. For each cluster *C* found by *DBSCAN*, k-means use $Scor_C$ points as starting points. The result is $K = |Scor_C|$ subclusters and centroids within *C*.

Each local model $LocalModel_k$ consists of a set of m_k pairs: a representative *r* (complete specific core point), and an ε radius value. The number *m* of pairs transmitted from each site *k* is determined by the number *n* of clusters C_i found on site *k*. Each of these pairs (r, ε_r) represents a subset of samples that are all located in a corresponding local cluster. Obviously, we have to check whether it is possible to merge two or more of these clusters, found on different sites, together. That is the main task of a global modeling part. To find such a global model, the algorithm continues with the density-based clustering algorithm DBSCAN again but only for collected representatives from local models. Because of characteristics of these representative points, the parameter $MinPts_{global}$ is set to 2, and radius ε_{global} value should be set generally close to $2\varepsilon_{local}$.

In Figure 12.32, an example of distributed DBSCAN for $\varepsilon_{global} = 2\varepsilon_{local}$ is depicted. In Figure 12.32a the independently detected clusters on site 1, 2, and 3 are represented. The cluster on site 1 is represented using K-means by two representatives, R_1 and R_2, whereas the clusters on site 2 and site 3 are only represented by one representative as shown in Figure 12.32b. Figure 12.32c illustrates that all four local clusters from the different sites are merged together in one large cluster. This integration is obtained by using an ε_{global} parameter equal to $2\varepsilon_{local}$. Figure 12.32c also makes clear that an $\varepsilon_{global} = \varepsilon_{local}$ is insufficient to detect this global cluster. When the final global model is obtained, the model is distributed to local sites. This model makes corrections comparing previously found local models. For example, in the local clustering some points may be left as outliers, but with the global model they may be integrated into modified clusters.

12.5 CORRELATION DOES NOT IMPLY CAUSALITY

An associational concept is any relationship that can be defined in terms of a frequency-based joint distribution of observed variables, while a causal concept is any relationship that cannot be defined from the distribution alone. Even simple examples show that the associational criterion is neither necessary nor sufficient for causality confirmation. For example, data mining might determine that males with income between \$50,000 and \$65,000 who subscribe to certain magazines are likely purchasers of a product you want to sell. While you can take advantage of this pattern, say by aiming your marketing at people who fit the pattern, you should not assume that any of these factors (income, type of magazine) *cause* them to buy your product. The predictive relationships found via data mining are not necessarily *causes* of an action or behavior.

The research questions that motivate many studies in the health, social, and behavioral sciences are not statistical but causal in nature. For example, what is the efficacy of a given drug in a given population, or what fraction of past crimes could have been avoided by a given policy? The central target of such studies is to determine cause–effect relationships among variables of interests, for example, treatments–diseases or policies–crime, as precondition–outcome relationships. In order to express causal assumptions mathematically, certain extensions are required in the standard mathematical language of statistics, and these extensions are not generally emphasized in the mainstream literature and education.

The aim of standard statistical analysis, typified by regression and other estimation techniques, is to infer parameters of a distribution from samples drawn from that distribution. With the help of such parameters, one can infer associations among variables, or estimate the likelihood of past and future events. These tasks are managed well by standard statistical analysis so long as experimental conditions remain the same. Causal analysis goes one step further; its aim is to infer aspects of the data-generation process. Associations characterize static conditions, while causal analysis deals with changing conditions. There is nothing in the joint distribution of symptoms and diseases to tell us that curing the former would or would not cure the latter.

Drawing analogy to visual perception, the information contained in a probability function is analogous to a geometrical description of a three-dimensional object; it is sufficient for predicting how that object will be viewed from any angle outside the object, but it is insufficient for predicting how the object will be deformed if manipulated and squeezed by external forces. The additional information needed for making predictions such as the object's resilience or elasticity is analogous to the information that causal assumptions provide. These considerations imply that the slogan "correlation does not imply causation" can be translated into a useful principle: One cannot substantiate causal claims from associations alone, even at the population level. Behind every causal conclusion there must lie some causal assumptions that are not testable in observational studies.

Any mathematical approach to causal analysis must acquire a new notation for expressing causal assumptions and causal claims. To illustrate, the syntax of probability calculus does not permit us to express the simple fact that "symptoms do not cause diseases," let alone draw mathematical conclusions from such facts. All we can say is that two events are dependent—meaning that if we find one, we can expect to encounter the other, but we cannot distinguish statistical dependence, quantified by the conditional probability *P*(*disease/symptom*) from causal dependence, for which we have no expression in standard probability calculus. Symbolic representation for the relation "symptoms cause disease" is distinct from the symbolic representation of "symptoms are associated with disease."

The need to adopt a new notation, foreign to the province of probability theory, has been traumatic to most persons trained in statistics partly because the adaptation of a new language is difficult in general, and partly because statisticians—this author included—have been accustomed to assuming that all phenomena, processes, thoughts, and modes of inference can be captured in the powerful language of probability theory. Causality formalization requires new mathematical machinery for cause–effect analysis and a formal foundation for counterfactual analysis including concepts such as "path diagrams," "controlled distributions," causal structures, and causal models.

12.5.1 Bayesian Networks

One of the powerful aspects of graphical models is that a specific graph can make probabilistic statements for a broad class of distributions. In order to motivate the use of directed graphs to describe probability distributions, consider first an arbitrary joint distribution p(a, b, c) over three variables a, b, and c. By application of the product rule of probability, we can write the joint distribution in the form

$$p(a, b, c) = p(c \mid a, b)p(a, b) = p(c \mid a, b)p(b \mid a)p(a)$$

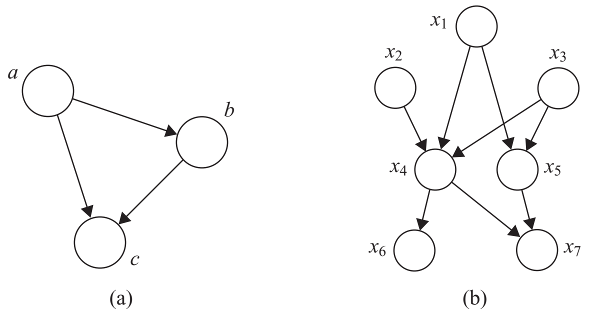


Figure 12.33. A directed graphical model representing the joint probability distribution over a set of variables. (a) Fully connected; (b) partially connected.

We now represent the right-hand side of the equation in terms of a simple graphical model as follows. First, we introduce a node for each of the random variables a, b, and c and associate each node with the corresponding conditional distribution on the right-hand side of the equation. Then, for each conditional distribution we add directed links, (arrows) to the graph from the nodes corresponding to the variables on which the distribution is conditioned. Thus, for the factor p(c|a, b), there will be links from nodes a and b to node c, whereas for the factor p(a) there will be no incoming links, as presented in Figure 12.33a. If there is a link going from a node a to a node b, then we say that node *a* is the *parent* of node b, and we say that node b is the *child* of node *a*.

For given K variables, we can again represent a joint probability distribution as a directed graph having K nodes, one for each conditional distribution, with each node having incoming links from all lower numbered nodes. We say that this graph is *fully connected* because there is a link between every pair of nodes. Consider now the graph shown in Figure 12.33b, which is not a fully connected graph because, for instance, there is no link from x_1 to x_2 or from x_3 to x_7. We may transform this graph to the corresponding representation of the joint probability distribution written in terms of the product of a set of conditional distributions, one for each node in the graph. The joint distribution of all seven variables is given by

$$p(x_1, x_2, \ldots, x_7) = p(x_1)p(x_2)p(x_3)p(x_4 \mid x_1, x_2, x_3)p(x_5 \mid x_1, x_3)p(x_6 \mid x_4)p(x_7 \mid x_4, x_5)$$

Any joint distribution can be represented by a corresponding graphical model. It is the *absence* of links in the graph that conveys interesting information about the properties of the class of distributions that the graph represents. We can interpret such models as expressing the processes by which the observed data arose, and in many situations we may draw conclusions about new samples from a given probability distribution. The directed graphs that we are considering are subject to an important restriction, that is, that there must be no *directed cycles*. In other words, there are no closed paths within the graph such that we can move from node to node along links following the direction of the arrows and end up back at the starting node. Such graphs are also called *Directed Acyclic Graphs* (*DAGs*).

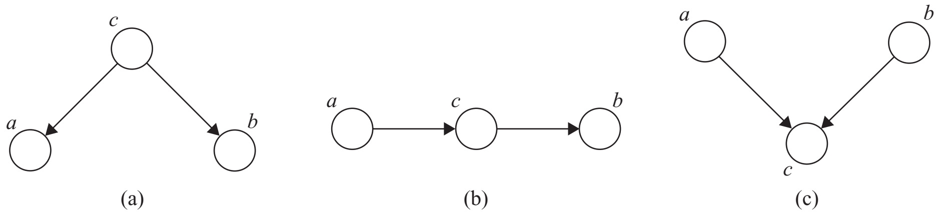


Figure 12.34. Joint probability distributions show different dependencies between variables *a*, *b*, and *c*.

An important concept for probability distributions over multiple variables is that of *conditional independence*. Consider three variables a, b, and c, and suppose that the conditional distribution of a, given b and c, is such that it does not depend on the value of b, so that

$$p(a \mid b, c) = p(a \mid c)$$

We say that *a* is conditionally independent of *b* given *c*. This can be extended in a slightly different way if we consider the joint distribution of *a* and *b* conditioned on *c*, which we can write in the form

$$p(a, b \mid c) = p(a \mid b, c)p(b \mid c) = p(a \mid c)p(b \mid c)$$

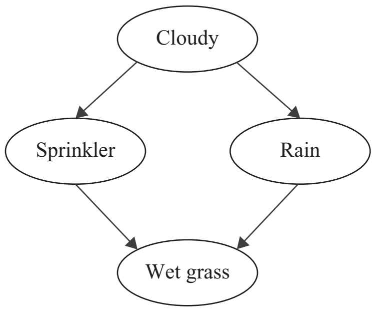
The joint distribution of *a* and *b*, conditioned on *c*, may be factorized into the product of the marginal distribution of *a* and the marginal distribution of *b* (again both conditioned on *c*). This says that the variables *a* and *b* are statistically independent, given *c*. This independence may be presented in a graphical form in Figure 12.34a. The other typical joint distributions may be graphically interpreted. The distribution for Figure 12.34b represents the case

$$p(b, c \mid a) = p(c \mid a)p(b \mid c)$$

while for Figure 12.34c the probability p(c| a, b) is under the assumption that variables *a* and *b* are independent *p*(*a, b*) = *p*(*a*) p(*b*).

In general, graphical models may capture the *causal* processes by which the observed data were generated. For this reason, such models are often called *generative* models. We could make previous models in Figure 12.33 generative by introducing a suitable prior distribution p(x) for all input variables (these are variables—nodes without input links). For the case in Figure 12.33a this is a variable *a*, and for the case in Figure 12.33b these are variables: x_1, x_2, and x_3. In practice, producing synthetic observations from a generative model can prove informative in understanding the form of the probability distribution represented by that model.

This preliminary analysis about joint probability distributions brings us to the concept of Bayesian networks (BN). *BN* are also called belief networks or probabilistic networks in the literature. The nodes in a BN represent variables of interest (e.g., the



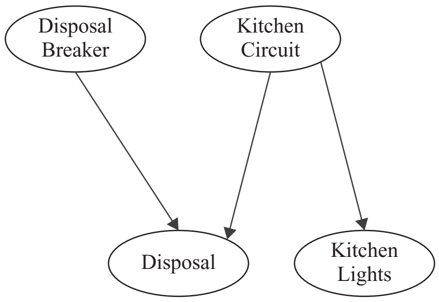


Figure 12.35. Two examples of Bayesian network architectures.

temperature of a device, the gender of a patient, the price of a product, the occurrence of an event), and the links represent dependencies among the variables. Each node has *states*, or a set of probable values for each variable. For example, the weather could be cloudy or sunny, an enemy battalion could be near or far, symptoms of a disease are present or not present, and the garbage disposal is working or not working. Nodes are connected with an arrow to show causality and also indicate the direction of influence. These arrows are called *edges*. The dependencies are quantified by conditional probabilities for each node given its parents in the network. Figure 12.35 presents some BN architectures, initially without probabilities distributions. In general, we can formally describe a BN as a graph in which the following holds:

1. A set of random variables makes up the nodes of the network.
2. A set of directed links connects pairs of nodes. The intuitive meaning of an arrow from node X to node Y is that X has a *direct influence* on Y.
3. Each node has a *conditional probability table* (CPT) that quantifies the effects that the parents have on the node. The parents of a node X are all those nodes that have arrows pointing to X.
4. The graph has no directed cycles (hence is a DAG).

Each node in the BN corresponds to a random variable X, and has a probability distribution of the variable P(X). If there is a directed arc from node X to node Y, this indicates that X has a direct influence on Y. The influence is specified by the conditional probability P(Y|X). Nodes and arcs define a *structure* of the BN. Probabilities are *parameters* of the structure.

We turn now to the problem of inference in graphical models, in which some of the nodes in a graph are clamped to observed values, and we wish to compute the posterior distributions of one or more subsets of other nodes. The network supports the computation of the probabilities of any subset of variables given evidence about any other subset. We can exploit the graphical structure both to find efficient algorithms for inference and to make the structure of those algorithms transparent. Specifically, many inference-based algorithms can be expressed in terms of the propagation of local

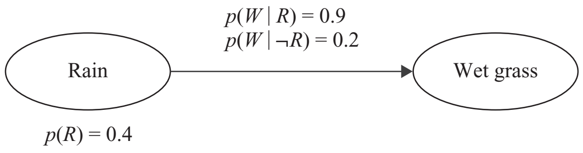


Figure 12.36. Simple causal graph.

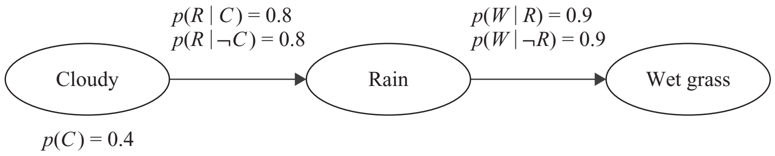


Figure 12.37. An extended causal graph.

probabilities around the graph. A BN can be considered as a probabilistic graph in which the probabilistic knowledge is represented by the topology of the network and the conditional probabilities at each node. The main purpose of building knowledge on probabilities is to use it for inference, that is, for computing the answer for particular cases about the domain.

For example, we may assume that rain causes the grass to get wet. Causal graph in Figure 12.36 explains the cause–effect relation between these variables, including corresponding probabilities. If P(Rain) = P(R) = 0.4 is given, that also means P(¬R) = 0.6. Also, note that the sum of presented conditional probabilities is not equal to 1. If you analyze the relations between probabilities, P(W|R) + P(¬W|R) = 1, and also P(W|¬R) + P(¬W|¬R) = 1, not the sum of given probabilities. In these expressions R means "Rain," and W means "Wet grass." Based on the given BN, we may check the probability of "Wet grass":

$$P(W) = P(W \mid R)\ P(R) + P(W \mid R)\ P(R) = 0.9 \times 0.4 + 0.2 \times 0.6 = 0.48 \text{ (or } 48\%)$$

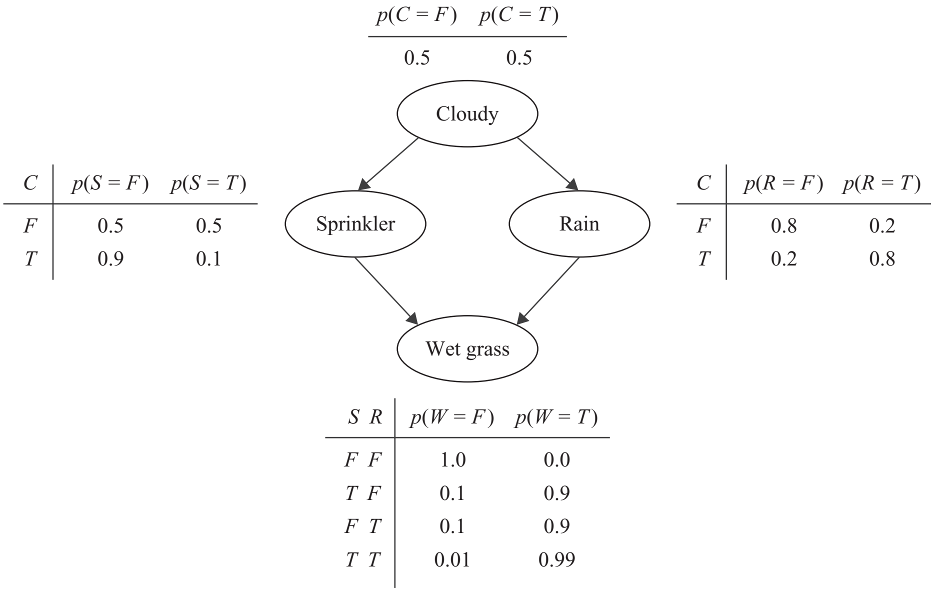
Bayes' rule allows us to invert the dependencies, and obtain probabilities of parents in the graph based on probabilities of children. That could be useful in many applications, such as determining probability of a diagnosis based on symptoms. For example, based on the BN in Figure 12.36, we may determine conditional probability P(Rain|Wet grass) = P(R|W). We know that

$$P(R, W) = P(W \mid R)\ P(R) = P(R \mid W)\ P(W)$$

and therefore

$$P(R \mid W) = (P[W \mid R]\ P[R])/P(W) = (P[W \mid R]\ P[R])/(P[W \mid R]\ P[R] + P[W \mid \neg R]\ P[\neg R]) = 0.9 * 0.4/(0.9 * 0.4 + 0.2 * 0.6) = 0.75$$

Let us include now more complex problems, and the more complex BN represented in Figure 12.37. In this case we have three nodes, and they are connected serially, often



$p(C = F)$	$p(C = T)$
0.5	0.5

C	$p(S = F)$	$p(S = T)$
F	0.5	0.5
T	0.9	0.1

C	$p(R = F)$	$p(R = T)$
F	0.8	0.2
T	0.2	0.8

S	R	$p(W = F)$	$p(W = T)$
F	F	1.0	0.0
T	F	0.1	0.9
F	T	0.1	0.9
T	T	0.01	0.99

Figure 12.38. Four-node architecture of a Bayesian network.

called head-to-tail connections of three events. Now an additional event, "Cloudy," with yes and no values is included as a variable at the beginning of the network. The R node blocks a path from C to W; it separates them. If the R node is removed, there is no path from C to W. Therefore, the relation between conditional probabilities in the graph are given as: P(C, R, W) = P(C) * P(R|C) * P(W|R).

In our case, based on the BN in Figure 12.37, it is possible to determine and use "forward" and "backward" conditional probabilities as represented in the previous BN. We are starting with:

$$\mathrm{P(W \mid C) = P(W \mid R) * P(R \mid C) + P(W \mid R) * P(\neg R \mid C)} = 0.9 * 0.8 + 0.2 * 0.2 = 0.76$$

Then, we may use Bayes' rule for inverted conditional probabilities:

$$\mathrm{P(C \mid W) = (P[W \mid C] * P[C])/P(W)} = 0.65 \quad (\mathrm{P(W)}\ \textit{requires detailed computation})$$

More complex connections may be analyzed in BN. The following Figure 12.38 shows the graph structure and the assumed input parameters.

The parameters of a graphical model are represented by the conditional probability distributions in a form of CPT tables for each node, given its parents. The simplest form of a formalized distribution, a CPT table, is suitable when the nodes are discrete-valued. All nodes in Figure 12.38 are represented with a discrete set of states, and corresponding CPTs. For example, "Sprinkler" node (S) may be "on" and "off," and is represented in the table with T and F values. Sprinkler CPT table is generated including input discrete values for node the "Cloudy" (C). Many algorithms for BN analysis may be expressed in terms of the propagation of probabilities through the graph.

All probabilistic models, no matter how refined and accurate, Bayesian included, describe a distribution over possible observed events, but say nothing about what will happen if a certain intervention occurs. For example, what if I turn on the sprinkler? What effect does that have on the season, or on the connection between wetness and slipperiness? A causal network is a BN with the added property that the parents of each node are its direct causes. In such a network, the result of an intervention is obvious: The sprinkler node is set to "on" and the causal link between the season and the sprinkler is removed. All other causal links and conditional probabilities remain intact. This added property endows the causal network with the capability of representing and responding to external or spontaneous changes. For example, to represent a disabled sprinkler in the story of Figure 12.38, we simply delete from the network all links incident to the node Sprinkler. To represent the policy of turning the sprinkler off if it rains, we simply add a link between Rain and Sprinkler. Such changes would require much greater remodeling efforts if the network were not constructed along the causal direction. This remodeling flexibility may well be cited as the ingredient that manages novel situations instantaneously, without requiring training or adaptation of the model.

12.6 PRIVACY, SECURITY, AND LEGAL ASPECTS OF DATA MINING

An important lesson of the Industrial Revolution was that the introduction of new technologies can have a profound effect on our ethical principles. In today's Information Revolution we strive to adapt our ethics to diverse concepts in cyberspace. The recent emergence of very large databases, and their associated data-mining tools, presents us with yet another set of ethical challenges to consider. The rapid dissemination of data-mining technologies calls for an urgent examination of their social impact. It should be clear that data mining itself is not socially problematic. Ethical challenges arise when it is executed over data of a personal nature. For example, the mining of manufacturing data is unlikely to lead to any consequences of a personally objectionable nature. However, mining clickstreams of data obtained from Web users initiate a variety of ethical and social dilemmas. Perhaps the most significant of these is the invasion of privacy, but that is not the only one.

Thanks to the proliferation of digital technologies and networks such as the Internet, and tremendous advances in the capacity of storage devices and parallel decreases in their cost and physical size, many private records are linked and shared more widely and stored far longer than ever before, often without the individual consumer's knowledge or consent. As more everyday activities move online, digital records contain more detailed information about individuals' behavior. Merchants' record data are no longer only on what individuals buy and how they pay for their purchases. Instead, those data include every detail of what we look at, the books we read, the movies we watch, the music we listen to, the games we play, and the places we visit. The robustness of these records is difficult to overestimate and is not limited to settings involving commercial transactions. More and more computers track every moment of most employees' days.

E-mail and voice mail are stored digitally; even the content of telephone conversations may be recorded. Digital time clocks and entry keys record physical movements. Computers store work product, text messages, and Internet browsing records—often in keystroke-by-keystroke detail, they monitor employee behavior.

The ubiquitous nature of data collection, analysis, and observation is not limited to the workplace. Digital devices for paying tolls, computer diagnostic equipment in car engines, and global positioning services that are increasingly common in passenger vehicles record every mile driven. Cellular telephones and personal digital assistants record not only call and appointment information, but location as well, and transmit this information to service providers. Internet Service providers (ISPs) record online activities; digital cable and satellite record what we watch and when; alarm systems record when we enter and leave our homes; and all of these data are held by third parties. Information on our browsing habits is available to both the employer and the ISP. If an employee buys an airline ticket through an online travel service, such as Travelocity or Expedia, the information concerning that transaction will be available to the employer, the ISP, the travel service, the airline, and the provider of the payment mechanism, at a minimum.

All indications are that this is just the beginning. Broadband Internet access into homes has not only increased the personal activities we now engage in online but also created new and successful markets for remote computer backup and online photo, e-mail, and music storage services. With Voice over Internet Protocol (IP) telephone service, digital phone calls are becoming indistinguishable from digital documents: Both can be stored and accessed remotely. Global positioning technologies are appearing in more and more products, and Radio Frequency Identification Tags are beginning to be used to identify high-end consumer goods, pets, and even people.

Many individuals are unaware of the extent of the personal data stored, analyzed, and used by government institutions, private corporations, and research labs. Usually, it is only when things go wrong that individuals exercise their rights to obtain these data and seek to eliminate or correct it. For many of those whose records are accessed through data mining, we do not know it is happening, and may never find out because nothing incriminating is signaled. But we still know that data mining allows companies to accumulate and analyze vast amounts of information about us, sufficient perhaps to create, with the help of data mining, what some have called personality or psychological "mosaics" of the subjects. One result of the entry into the information age is that faceless bureaucrats (in a company, in government, everywhere) will be able to compile dossiers on anyone and everyone, for any reason or for no reason at all. The possibility, even if slim, that this information could somehow be used to our detriment or simply revealed to others can create a chilling effect on all these activities.

Data and the information derived from those data using data mining are an extremely valuable resource for any organization. Every data-mining professional is aware of this, but few are concentrated on the impact that data mining could have on privacy and the laws surrounding the privacy of personal data. Recent survey showed that data-mining professionals "prefer to focus on the advantages of Web-data mining instead of discussing the possible dangers." These professionals argued that Web-data mining does not threaten privacy. One might wonder why professionals are not aware of or concerned over the possible misuse of their work, and the possible harm it might cause to individuals and society. Part of the reason some professionals are not concerned about the possible misuse of their work and the potential harm it might cause might lie in the explanation that "they are primarily technical professionals, and somebody else should take care of these social and legal aspects." But sensible regulations of data mining depend on the understanding of its many variants and its potential harms. Therefore, technical professionals have to be a part of the team, often leading, which will try to solve privacy challenges.

The key ethical issues in mining personal data are that people are generally:

1. not aware that their personal information is being gathered,
2. do not know to what use the data will be made, and/or
3. have not consented to such collection of data or data use.

In order to alleviate concerns about data privacy, a number of techniques have recently been proposed in order to perform the data-mining tasks in a privacy-preserving way. These techniques for performing privacy-preserving data mining are drawn from a wide array of related topics such as cryptography and information hiding. Most privacy-preserving data-mining methods apply a transformation that reduces the effectiveness of the underlying data when they are applied to data-mining methods or algorithms. In fact, there is a natural trade-off between privacy and accuracy although this trade-off is affected by the particular algorithm that is used for privacy preservation. The key directions in the field of privacy-preserving data mining include:

- *Privacy-Preserving Data Publishing:* These techniques tend to study different transformation methods associated with privacy. They concentrate on how the perturbed data can be used in conjunction with classical data-mining methods.
- *Changing the Results of Data-Mining Applications to Preserve Privacy:* These techniques are concentrated on the privacy of data-mining results where some results are modified in order to preserve the privacy. A classic example of such techniques are association-rule hiding methods, in which some of the association rules are suppressed in order to preserve privacy.
- *Cryptographic Methods for Distributed Privacy:* If the data are distributed across multiple sites, a variety of cryptographic protocols may be used in order to communicate among the different sites so that secure function computation is possible without revealing sensitive information.

Recent research trends propose that issues of privacy protection, currently viewed in terms of data *access,* be reconceptualized in terms of data *use*. From a technology perspective, this requires supplementing legal and technical mechanisms for access control with new mechanisms for *transparency* and *accountability* of data used in a data-mining process. Current technical solutions of the impact of data mining on privacy have generally focused on limiting access to data at the point of collection or storage. Most effort has been put into the application of cryptographic and statistical techniques to construct finely tuned access-limiting mechanisms. Even if privacy-preserving data-mining techniques prove to be practical, they are unlikely to provide sufficient public assurance that data-mining inferences conform to legal restrictions. While privacy-preserving data-mining techniques are certainly necessary in some contexts, they are not a sufficient privacy protection without the transparency and accountability.

In the long run, access restriction alone is not enough to protect privacy or to ensure reliable conclusions, and the best example of these challenges is Web and Web-mining technology. As we leave the well-bounded world of enterprise databases and enter the open, unbounded world of the Web, data users need a new class of tools to verify that the results they see are based on data that are from trustworthy sources and are used according to agreed-upon institutional and legal requirements. The implications of data mining on digital social networks such as Facebook, Myspace, or Twitter may be enormous. Unless it is part of a public record designed for consumption by everyone or describes an activity observed by strangers, the stored information is rarely known outside our families, much less outside our social networks. An expectation that such information and potential derivatives will remain "private" on the Internet is not anymore a reasonable assumption from the social network perspective. One of the major contributors to these controversies is the absence of clear legal standards. Thirty years ago the lack of relevant law was understandable: The technologies were new; their capacity was largely unknown; and the types of legal issues they might raise were novel. Today, it is inexplicable and threatens to undermine both privacy and security. Hence, we must develop technical, legal, and policy foundations for transparency and accountability of large-scale mining across distributed heterogeneous data sources. *Policy awareness* is a property of the Semantic Web still in development that should provide users with accessible and understandable views of the policies associated with resources.

The following issues related to privacy concerns may assist in individual privacy protection during a data-mining process, and should be a part of the best data-mining practices:

- *Whether there is a clear description of a program's collection of personal information, including how the collected information will serve the program's purpose?* In other words, be transparent early on about a data-mining project's purpose. Clearly state up-front the business benefits that will be achieved by data mining. Provide notice of the combining of information from different sources. Companies like Walmart or Kroger store much of their business and customer data in large warehouses. Their customers are not told of the extent of the information that is accumulated on them, how long it will be kept, the uses to which the data will be put, or other users with which data will be shared.
- *Whether information collected for one purpose will then be used for additional, secondary purposes in the future?* Ensure that any new purpose of a project is consistent with the project's original purpose. Maintain oversight of a data-mining project and create audit requirements.
- *Whether privacy protections are built-in to systems in the early developmental stage?* Build in privacy considerations up-front, and bring in all stakeholders at

the beginning, including privacy advocates to get input from them. Ensure the accuracy of data entry.

- *What type of action will be taken on the basis of information discovered through a data-mining process?* Where appropriate, anonymize personal information. Limit the actions that may be taken as a result of unverified findings from data mining.
- *Whether there is an adequate feedback system for individuals to review and correct their personal information that is collected and maintained in order to avoid "false positives" in a data-mining program?* Determine whether an individual should have a choice in the collection of information. Provide notice to individuals about use of their personal information. Create a system where individuals can ensure that any incorrect personal information can be corrected.
- *Whether there are proper disposal procedures for collected and derived personal information that has been determined to be irrelevant?*

Some observers suggest that the privacy issues presented by data mining will be resolved by technologies, not by law or policy. But even the best technological solutions will still require a legal framework in which to operate, and the absence of that framework may not only slow down their development and deployment, but make them entirely unworkable. Although there is no explicit right to privacy of personal data in the Constitution, legislation and court decisions on privacy are usually based on parts of the First, Fourth, Fifth, and Fourteenth Amendments. Except for health-care and financial organizations, and data collected from children, there is no law that governs the collection and use of personal data by commercial enterprises. Therefore, it is essentially up to each organization to decide how they will use the personal data they have accumulated on their customers. In early March 2005, hackers stole the personal information of 32,000 people from the databases of LexisNexis. The stolen data included Social Security numbers and financial information. Although the chief executive officer (CEO) of LexisNexis claimed that the information they collect is governed by the U.S. Fair Credit Reporting Act, members of Congress disagreed. As a result of this and other large-scale identity thefts in recent years, Congress is considering new laws explaining what personal data a company can collect and share. For example, Congress is considering a law to prohibit almost all sales of Social Security numbers.

At the same time, especially since 9/11, government agencies have been eager to experiment with the data-mining process as a way of nabbing criminals and terrorists. Although details of their operation often remain unknown, a number of such programs have come to light since 2001. The Department of Justice (DOJ), through the Federal Bureau of Investigation (FBI), has been collecting telephone logs, banking records, and other personal information regarding thousands of Americans not only in connection with counterterrorism efforts, but also in furtherance of ordinary law enforcement. A 2004 report by the Government Accountability Office (GAO) found 42 federal departments—including every cabinet-level agency that responded to the GAO's survey—engaged in, or were planning to engage in, 122 data-mining efforts involving personal information (U.S. General Accounting Office, *Data Mining: Federal Efforts*

Cover a Wide Range of Uses [GAO-04-548], May 2004, pp. 27–64). Recently, the U.S. Government recognized that sensible regulation of data mining depends on understanding its many variants and its potential harms, and many of these data-mining programs are reevaluated. In the United Kingdom, the problem is being addressed more comprehensively by the Foundation for Information Policy Research, an independent organization examining the interaction between information technology and society with goals to identify technical developments with significant social impact, commission research into public policy alternatives, and promote public understanding and dialog between technologists and policy makers in the United Kingdom and Europe. It combines information technology researchers with people interested in social impacts, and uses a strong media presence to disseminate its arguments and educate the public.

There is one additional legal challenge related specifically to data mining. Today's privacy laws and guidelines, where they exist, protect data that are explicit, confidential, and exchanged between databases. However, there is no legal or normative protection for data that are implicit, nonconfidential, and not exchanged. Data mining can reveal sensitive information that is derived from nonsensitive data and meta-data through the inference process. Information gathered in data mining is usually implicit patterns, models, or outliers in the data, and questionable is the application of privacy regulations primarily written for traditional, explicit data.

In addition to data privacy issues, data mining raises other social concerns. For example, some researchers argue that data mining and the use of consumer profiles in some companies can actually exclude groups of customers from full participation in the marketplace and limit their access to information.

Good privacy protection not only can help build support for data mining and other tools to enhance security, it can also contribute to making those tools more effective. As technology designers, we should provide an information infrastructure that helps society to be more certain that data-mining power is used only in legally approved ways, and that the data that may give rise to consequences for individuals are based on inferences that are derived from accurate, approved, and legally available data. Future data-mining solutions reconciling any social issues must not only be applicable to the ever changing technological environment, but also flexible with regard to specific contexts and disputes.

12.7 REVIEW QUESTIONS AND PROBLEMS

1. What are the benefits in modeling social networks with a graph structure? What kind of graphs would you use in this case?

2. For the given undirected graph G:
   (a) compute the degree and variability parameters of the graph;
   (b) find adjacency matrix for the graph G;
   (c) determine binary *code(G)* for the graph;
   (d) find closeness parameter or each node of the graph; and
   (e) what is the betweeness measure for node number 2?

3. For the graph given in Problem number 2, find partial *betweeness centrality* using modified graph starting with node number 5.

4. Give real-world examples for traditional analyses of temporal data (i.e., trends, cycles, seasonal patterns, outliers).

5. Given the temporal sequence S = {1 2 3 2 4 6 7 5 3 1 0 2}:
   (a) find PAA for four sections of the sequence;
   (b) determine SAX values for solution in (a) if (1) $\alpha = 3$, (2) $\alpha = 4$;
   (c) find PAA for three sections of the sequence; and
   (d) determine SAX values for solution in (c) if (1) $\alpha = 3$, (2) $\alpha = 4$.

6. Given the sequence S = {A B C B A A B A B C B A B A B B C B A C C}:
   (a) Find the longest subsequence with frequency ≥ 3.
   (b) Construct finite-state automaton (FSA) for the subsequence found in (a).

7. Find normalized contiguity matrix for the table of U.S. cities:

Minneapolis	Chicago	New York
Nashville	Louisville	Charlotte

   Make assumption that only neighboring cities (vertical and horizontal) in the table are close.

8. For the BN in Figure 12.38 determine:
   (a) P(C, R, W)
   (b) P($\rceil$C, $\rceil$S, W)

9. Review the latest articles on privacy-preserving data mining that are available on the Internet. Discuss the trends in the field.

10. What are the largest sources of unintended personal data on the Internet? How do we increase awareness of Web users of their personal data that are available on the Web for a variety of data-mining activities?

11. Discuss an implementation of transparency and accountability mechanisms in a data-mining process. Illustrate your ideas with examples of real-world data-mining applications.

12. Give examples of data-mining applications where you would use DDM approach. Explain the reasons.

12.8 REFERENCES FOR FURTHER STUDY

Aggarwal C. C., P. S. Yu, *Privacy-Preserving Data Mining: Models and Algorithms*, Springer, Boston, 2008.

The book proposes a number of techniques to perform the data-mining tasks in a privacy-preserving way. These techniques generally fall into the following categories: data modification techniques, cryptographic methods and protocols for data sharing, statistical techniques for disclosure and inference control, query auditing methods, and randomization and perturbation-based techniques. This edited volume contains surveys by distinguished researchers in the privacy field. Each survey includes the key research content as well as future research directions. *Privacy-Preserving Data Mining: Models and Algorithms* is designed for researchers, professors, and advanced-level students in computer science, and is also suitable for industry practitioners.

Chakrabarti D., C. Faloutsos, Graph Mining: Laws, Generators, and Algorithms, *ACM Computing Surveys*, Vol. 38, March 2006, pp. 1–69.

How does the Web look? How could we tell an abnormal social network from a normal one? These and similar questions are important in many fields where the data can intuitively be cast as a graph; examples range from computer networks to sociology to biology and many more. Indeed, any *M*: *N* relation in database terminology can be represented as a graph. A lot of these questions boil down to the following: "How can we generate synthetic but realistic graphs?" To answer this, we must first understand what patterns are common in real-world graphs and can thus be considered a mark of normality/realism. This survey gives an overview of the incredible variety of work that has been done on these problems. One of our main contributions is the integration of points of view from physics, mathematics, sociology, and computer science. Further, we briefly describe recent advances on some related and interesting graph problems.

Da Silva J. C., et al., Distributed Data Mining and Agents, *Engineering Applications of Artificial Intelligence*, Vol. 18, No. 7, October 2005, pp. 791–807.

Multi-Agent Systems (MAS) offer an architecture for distributed problem solving. DDM algorithms focus on one class of such distributed problem solving tasks—analysis and modeling of distributed data. This paper offers a perspective on DDM algorithms in the context of multi-agent systems. It discusses broadly the connection between DDM and MAS. It provides a high-level survey of DDM, then focuses on distributed clustering algorithms and some potential applications in multi-agent-based problem-solving scenarios. It reviews algorithms for distributed clustering, including privacy-preserving ones. It describes challenges for clustering in sensor-network environments, potential shortcomings of the current algorithms, and future work accordingly. It also discusses confidentiality (privacy preservation) and presents a new algorithm for privacy-preserving density-based clustering.

Laxman S., P. S. Sastry, A Survey of Temporal Data Mining, *Sadhana*, Vol. 31, Part 2, April 2006, pp. 173–198.

Data mining is concerned with analyzing large volumes of (often unstructured) data to automatically discover interesting regularities or relationships that in turn lead to better understanding of the underlying processes. The field of temporal data mining is concerned with such analysis in the case of ordered data streams with temporal interdependencies. Over the last decade many interesting techniques of temporal data mining were proposed and shown to be useful in many applications. Since temporal data mining brings together techniques from different fields such as statistics, machine learning, and databases, the literature is scattered among many different sources. In this article, we present an overview of the techniques of temporal data mining.We mainly concentrate on algorithms for pattern discovery in sequential data streams. We also describe some recent results regarding statistical analysis of pattern discovery methods.

Mitsa T., *Temporal Data Mining*, Chapmann & Hall/CRC Press, Boca Raton, FL, 2010.

From basic data-mining concepts to state-of-the-art advances, *Temporal Data Mining* covers the theory of this subject as well as its application in a variety of fields. It discusses the incorporation of temporality in databases as well as temporal data representation, similarity computation, data classification, clustering, pattern discovery, and prediction. The book also explores the use of temporal data mining in medicine and biomedical informatics, business and industrial applications, Web-usage mining, and spatiotemporal data mining. Along with various state-of-the-art algorithms, each chapter includes detailed references and short descriptions of relevant algorithms and techniques described in other references. In the appendices, the author explains how data mining fits the overall goal of an organization and how these data can be interpreted for the purposes of characterizing a population. She also provides programs written in the Java language that implement some of the algorithms presented in the first chapter.

Pearl, J., *Causality: Models, Reasoning and Inference*, 2nd edition, Cambridge University Press, Cambridge, UK, 2009.

This book fulfills a long-standing need for a rigorous yet accessible treatise on the mathematics of causal inference. Judea Pearl has done a masterful job of describing the most important approaches and displaying their underlying logical unity. The book deserves to be read by all scientists who use nonexperimental data to study causation, and would serve well as a graduate or advanced undergraduate course text. The book should prove invaluable to researchers in AI, statistics, economics, epidemiology, and philosophy, and, indeed, all those interested in the fundamental notion of causality. It may well prove to be one of the most influential books of the next decade.

Zeitouni K., A Survey of Spatial Data Mining Methods: Databases and Statistics Point of View, in *Data Warehousing and Web Engineering*, S. Becker, ed., IRM Press, Hershey, PA, 2002.

This chapter reviews the data-mining methods that are combined with GIS for carrying out spatial analysis of geographic data. We will first look at data-mining functions as applied to such data and then highlight their specificity compared with their application to classical data. We will go on to describe the research that is currently going on in this area, pointing out that there are two approaches: The first comes from learning on spatial databases, while the second is based on spatial statistics. We will conclude by discussing the main differences between these two approaches and the elements they have in common.