CHAPTER 1

Preliminaries and Notation

1.1 Introduction

The work of Euler in 1736 is often recognized as the first study of graphs [260]. The original question Euler addressed was how to cross once and only once the seven bridges in the town of Königsberg (see Figure 1.1).

The first book on graph theory, written by König [472], appeared only 200 years later, although various related studies were published before this book (e.g., [144, 642, 683, 745]). In [360], Hamilton introduced the famous Around the World game, which is currently known as the Traveling Salesman Problem, and it is still under investigation. Since 1936, graph theory has developed rapidly under the leadership of various experts in operations research who have addressed specific problems (e.g., see [279, 481]). For further details on the early history of graph theory, the reader may refer to *Graph Theory* by Biggs [90].

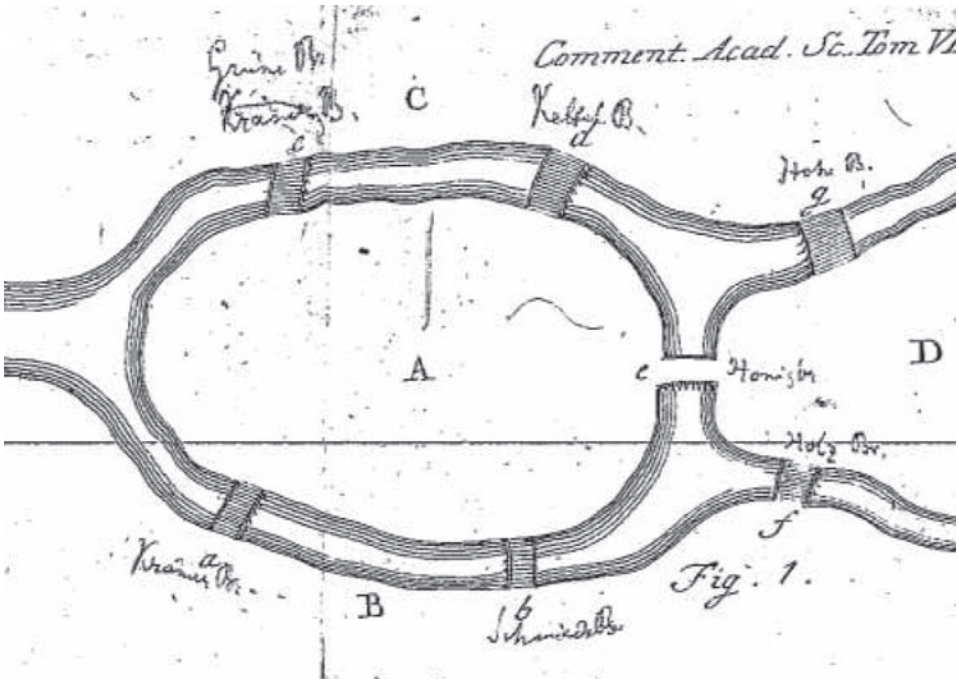

Figure 1.1. Representation of the bridges in the town of Königsberg.

Figure 1.2. Graph representation of the professional network of François Fouss, proposed by inMaps.LinkedInLabs.com, where each node represents an individual and each edge an existing link between the corresponding persons.

More recently, with the development of algorithms and computers, graphs have been used in many areas to model and analyze real interconnected systems, including chemistry, biology, physics, human sciences, and engineering.

Some well-known examples are as follows. In chemistry, graphs can be used to model molecules by assuming that molecules with similar chemical structures will have similar properties. In social networks, graphs can be used to model the links (such as friendship or professional links) between the members of a community (see Figure 1.2), for example, to discover the proximity or similarities between members or the common behaviors of members. In information and communication technology, graphs can be used to model and analyze the Internet, by, for example, mapping the physical connectivity of the Internet (see Figure 1.3).

The common feature of all these examples (and many others) is the use of a graphical structure to model or represent part of the real world. From a mathematical viewpoint, a graph is simply a collection of nodes that correspond to entities in the real world and edges that express links between these entities (interactions, relations, transactions, etc.). According to various examples, these entities can be of various types (web pages, individuals, atoms, etc.), and the links can have various meanings (hyperlinks, friendships, chemical bonds, etc.), and thus they correspond to different relationships depending on the reality modeled by the graph.

The remainder of this chapter introduces some basic graph concepts, algorithms, and associated matrices that are particularly useful in various parts of the book. It starts with a short synopsis of the content of the book.

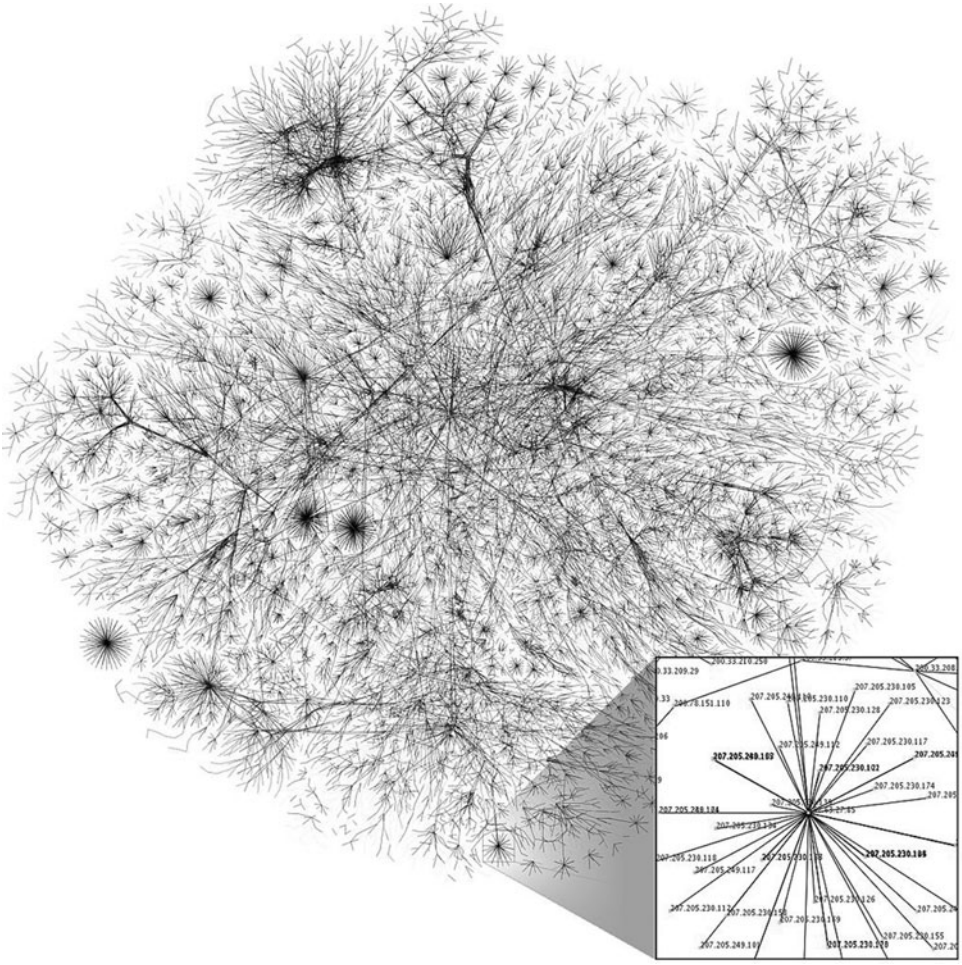

Figure 1.3. Graph representation of the Internet proposed by Internet mapping (see http://en.wikipedia.org/wiki/Internet_Mapping_Project, the Wikipedia page of the Internet Mapping Project).

1.2 Content of the Book

The main focus of the book is the extraction of useful information from static **network data**, observed in real life. Each chapter covers techniques tackling a family of *functional tasks*, such as "Identifying prestigious nodes," "Detecting the most central nodes," "Predicting information associated with the nodes," and "Finding dense communities." Each method is described in depth in a separate section that is – as far as possible – *self-contained*, so that each can be read independently.

The content of the book comprises two levels of analysis for static network data, where the first level (Chapters 2–5) is focused on *characterizing the basic elements of a network* (i.e., nodes and/or edges) and the second level (Chapters 6–10) is focused on *analyzing the global structure of a network*.

In particular, Chapters 2–5 describe methods for answering questions such as, *Should these two nodes be considered as similar/dissimilar?* or *Does this node have a central or key position in the network?*

Chapter 2. In Chapter 2, we introduce various **similarity/dissimilarity** measures between the nodes of a graph. These measures are computed from the structure of a graph and may serve to answer questions like *Who will be your best friend on a social network like Facebook?* In most cases, these measures consider the amount of connectivity between the nodes, that is, two nodes are more similar when the number of direct or indirect paths between them is larger. Several local (i.e., based on the neighborhood of the nodes of interest) or global (i.e., based on the whole graph) measures between nodes in a (generally undirected) graph are presented in this chapter. Two of these global measures are of particular interest in our study: the **shortest path** and the **commute time** distances.

Chapter 3. The shortest path and the commute time distances can be regarded as two extreme ways of defining dissimilarity between graph nodes; that is, the former only considers the length without addressing the connectivity, whereas the latter only considers connectivity without addressing the length. In Chapter 3, we develop **families of dissimilarities** that lie in between these two distances. These quantities depend on a continuous parameter (at one limit of the value of the parameter, they converge to the shortest-path distance, whereas at the other end, they converge to the commute time distance). They thus "interpolate" between the two distances.

After defining the similarity/dissimilarity measures between the nodes of the network, they can be used for several tasks, such as *link prediction* (predicting missing links), *clustering* (finding compact communities), and *finding nearest neighbors*.

Chapter 4. In addition to information about the similarity/dissimilarity between pairs of nodes in a network, we could also be interested in answering questions such as, *What is the most representative, or central, node within a given community? How critical is a given node with respect to the information flow in a network?*, or *Which node is the most peripheral in a social network?* These questions are all focused on **centrality measures** in undirected graphs, covered in this chapter.

Many different measures of the centrality and prestige of a node have been defined in social science, computer science, physics, statistics, and applied mathematics, where these measures are also known as "importance," "standing," "prominence," or "popularity," especially in the case of social networks. In this book, we speak of prestige when the graph is directed, whereas the concept is referred to as centrality in the case of an undirected graph. In Chapter 4, we describe three types of centrality measures: the **closeness centrality** to quantify the extent to which a node (or a group of nodes) is central to the network; the **betweenness centrality** to quantify the extent to which a node (or a group of nodes) is an important intermediary in the network; and the **criticality** to quantify the extent to which a node or an edge is "critical" or "vital" to the graph in terms of communication, movement, or transmission.

Chapter 5. Chapter 5 considers **prestige measures** for quantifying the importance of a node in a directed graph where the edges possess some "endorsement" relationship. For instance, prestige measures are the focus of questions such as, *Does a node in a network have a special or prestigious position if it is chosen by many others? How influential is a given node in a social network?* In this context, the prestige of a node increases as it receives more positive citations or endorsements (incoming links). Numerous measures

have been developed in the social sciences (only the most popular are introduced in this chapter), and this chapter also describes some prestige measures introduced in computer science and applied mathematics. These measures were developed mainly in the context of bibliometrics and search engines, and they are now among the most popular for quantifying node prestige.

The first part of this book is focused on characterizing the elements of the network, whereas the second is devoted to analyzing the global structure of the network. In particular, Chapters 6–10 address the tasks of *labeling nodes*, *clustering nodes*, and *finding dense regions* as well as the *analysis of bipartite graphs* and *graph embedding*.

Chapter 6. In Chapter 6, we introduce some techniques for assigning a class label to an unlabeled node based on knowledge of the class of some labeled nodes and the network structure. A concrete example is, *Is it possible to predict the technological category of patents linked by citations, given that these categories are known only for a few nodes?* This **within-network classification** task conforms to the **semisupervised classification** paradigm, the goal of which is to fit a predictive model using a small number of labeled samples and some (usually a large number of) unlabeled samples (the labels are missing or unobserved for these samples), where it is assumed that combining these two sources of information will yield predictive models that are more accurate than when simply using the labeled samples alone (and thus ignoring the unlabeled samples). Most of the semisupervised classification models described in this chapter are presented in a one-versus-all classification setting (i.e., one model is fitted per class and the resulting models are then used for classification).

Chapter 7. Another well-known task when handling network data involves **clustering** the nodes of the network into a partition, that is, grouping a set of objects into subsets or clusters such that those belonging to the same cluster are more "related" than those belonging to different clusters. Most of the well-known clustering algorithms described in Chapter 7 comprise **top-down divisive techniques** (splitting methods) that start from an initial situation where all the nodes of the graph are contained in only one cluster before trying to split the cluster into pieces, **optimization techniques** that maximize a criterion by measuring the quality of the partition, and **bottom-up agglomerative techniques** that start from a degenerate partition where each node is a cluster by itself before trying to merge the most similar nodes/clusters recursively. Top-down and optimization techniques (described in Chapter 7) produce a partition of the nodes, whereas bottom-up techniques (described in Chapter 8) produce sets of dense clusters, at least at the beginning of the procedure. The algorithms described in this chapter answer questions such as, *Are there highly connected clusters with few links between clusters in the network?*

Chapter 8. It may also be interesting to **identify dense regions** inside the network where, instead of trying to find a partition of the graph, we only seek some subsets of nodes that are highly interconnected (the nodes that are not part of a dense region are simply not assigned to any cluster of reference). An example of a concrete problem is, *Can we identify dense communities of nodes in a mobile network, with a very high calling rate between the members of the community?* In Chapter 8, we first investigate some well-known local density measures to quantify the extent to which a local subset

of nodes centered on a particular node is highly cohesive. We then present some global measures for smoothing the density over the network, measures that tend to be more robust with respect to local variations in the density. Finally, we describe some **bottom-up agglomerative methods**, which allow highly dense regions to be detected by extending them gradually in a sequential manner according to a greedy algorithm.

Chapter 9. Chapter 9 focuses on **bipartite graphs**, where we explore various methods and models for analyzing such graphs. In bipartite graphs, the node set may be partitioned into two disjoint sets, where each edge has one endpoint in the first set of nodes and the other in the second set of nodes. These graphs appear naturally in applications involving two types of objects, or objects that play different roles, including *collaborative recommendation*, *item ranking*, *information retrieval*, or *matching problems*. An example question which the algorithms of this chapter are trying to answer is, *Can we identify groups of persons interested in the same movies as well as groups of movies watched by the same persons?* Most of the methods explored are standard and have been known for many years in the context of contingency tables analysis, that is, simple correspondence analysis, a latent class model, and a bi-clustering approach. However, others are more recent, such as the reputation model introduced in Section 9.4.

Chapter 10. Finally, we introduce **graph embedding** in Chapter 10, where the aim is to associate a position or vector in a Euclidean space with each node of the graph. Thus, this mapping corresponds to the configuration of the nodes in a Euclidean space that preserves the structure of the graph as much as possible. The techniques described here try to answer the following question: *Is it possible to represent the network in a two-dimensional plane in an accurate way, that is, while conserving the structure of the network?* In this chapter, we only present some of the most popular methods, including spectral methods (which define the embedding according to certain eigenvectors of graph-related matrices), a latent space method, and some basic force-directed techniques, which produce the layout based on a physical analogy (spring networks or attractive forces). After a graph embedding has been computed, it can be used for *graph drawing* (when the embedding space has dimension two or three), but more generally, it associates a data matrix with the graph, where each row of the matrix corresponds to a node. This data matrix can then be used in multivariate statistical techniques such as clustering and classification.

1.3 Basic Definitions and Notation

This section is intended to provide an informal description of the notation and vocabulary used throughout this book. Note that a **list of symbols and notation** is provided in the preamble of the book.

In mathematics and computer science, graph theory involves the study of graphs, where a graph is a collection of nodes and the edges that connect pairs of nodes. More precisely, graph theory provides a set of definitions, tools, and techniques for describing graphs and their properties (e.g., see [9, 78, 102, 106, 223, 331, 777, 810] for some standard textbooks on mathematical graph theory, as well as [85, 170, 233, 261, 316, 331, 432, 450, 706, 754] for textbooks related to algorithms on graphs, and [804] in the

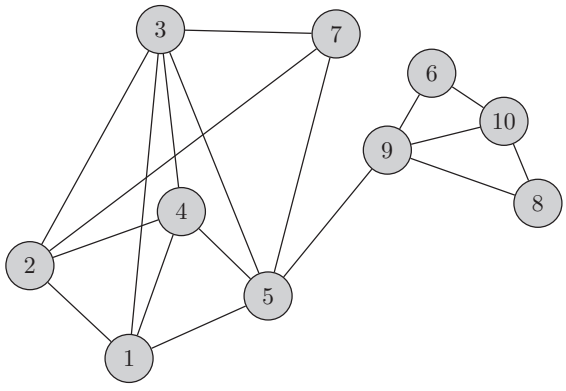

Figure 1.4. A simple unweighted, undirected graph G.

context of social sciences). Largely inspired by [102, 332, 469, 608, 706], this section reviews the basic terminology and concepts of graphs, while introducing important connections between graphs and matrix algebra and providing a brief presentation of some basic algorithms that address various questions related to graphs.

In this work, we are more interested in tools and techniques for analyzing and extracting information from **network data**, that is, graphs that model some real system or "sample" graphs observed in real life.

We must stress that this section is very compact and only outlines a very small part of the useful theory (please see the references given earlier for more rigorous and in-depth treatments).

An illustrative example. A small example is used throughout this section to illustrate various graph concepts. The graph G of this small and simple example is shown in Figure 1.4. Note that the positions of the nodes have no particular meaning.

1.3.1 Basic Graph Concepts

A **graph** or **network** G is a mathematical structure that can be formally defined by providing

- a finite nonempty set $\mathcal{V}(G) = \mathcal{V}$, the elements of which are called **nodes** (or **vertices**)
- a set $\mathcal{E}(G) = \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, the elements of which are (ordered or not) pairs of nodes called **edges** (or **arcs**, **links**)

Thus, a graph is a collection of nodes linked by edges, $(\mathcal{V}, \mathcal{E})$. In general, the nodes represent some **objects** or **entities** (e.g., people in a social network) and the edges represent the existence of a **relation** between two objects (e.g., "is a friend of" or "went together to a concert" in a social network). The theory of relations is a well-known field studied in discrete mathematics and relational databases (e.g., see [1, 672] for more information). The nodes are usually identified by a number, called the index of the node.

In other words, graphs can be viewed as a convenient way of representing pairwise relations between objects. In this book, we are mainly interested in graphs that represent

only a single relation. Many studies have addressed multirelational graphs [199, 234, 314] and the extension of simple relations, but this topic is not investigated in the present study. In general, there is at most one single edge between two nodes. However, in some situations, multiple edges are permitted (parallel edges), where we refer to this structure as a **multigraph**.

Note that we use the terms *graph* and *network* interchangeably. The number of nodes $n = |\mathcal{V}|$ and the number of edges $e = |\mathcal{E}|$ are sometimes called the **order** and the **size** of the graph G, respectively. A graph where the pairs of nodes that determine the edges are ordered (i.e., (i, j) is distinct from (j, i)) is called a **directed** graph. By convention, for a directed graph, the first node of the pair determines the **starting node** of the edge and the second node of the pair determines the **ending node** of the edge. If there is no order (i.e., (i, j) and (j, i) correspond to one and only one edge), the graph is said to be **undirected**. A directed edge connecting node i and node j is often denoted quite naturally by $i \to j$, or (i, j), whereas an undirected edge is denoted by $i \leftrightarrow j$, or simply by (i, j) (the order is not important in this case).

We say that two nodes are **adjacent** when an edge exists that connects these two nodes; they are therefore **neighbors**. A node and an edge are **incident** when the edge is connected to the node. When applying an algorithm designed for directed graphs to an undirected graph, each edge $i \leftrightarrow j$ of the undirected graph is considered as the superposition of two directed edges, $i \to j$ and $j \to i$. An obvious example of a directed graph is the World Wide Web, where the nodes are web pages and the edges are the (directed) hyperlinks between pages. By contrast, a graph of coauthorship is undirected.

Moreover, in the case of an undirected graph, the set of neighbors of i, that is, nodes adjacent to node i, will be denoted as $\mathcal{N}(i)$, or $\mathcal{N}(i) = \{j \mid (i, j) \in \mathcal{E}\}$. In the directed case, when there exists an edge $i \to j$, node j is said to be a **successor** of node i, and conversely, node i is said to be a **predecessor** of j. The set of successors of i will be denoted as $\mathcal{S}ucc(i)$, and the set of predecessors of i will be denoted as $\mathcal{P}red(i)$. Thus $\mathcal{S}ucc(i) = \{j \mid i \to j \in \mathcal{E}\}$ and $\mathcal{P}red(i) = \{j \mid j \to i \in \mathcal{E}\}$.

In the case of a **weighted undirected graph**, a nonnegative symmetric **weight** w_{ij} (with $w_{ij} = w_{ji}$), which quantifies the *degree of "affinity,"* the *degree of "similarity,"* or the *"closeness"* between the two nodes i and j, or alternatively a nonnegative symmetric **cost** c_{ij}, which quantifies the *cost of following the link* $i \to j$, is associated with each edge. In a coauthorship network, for example, weights can be the number of papers cosigned by two authors. As mentioned earlier, an undirected graph is often considered as a directed graph where, for each edge, both $i \to j$ and $j \to i$ are present with the same weight (the edge is bidirectional). A graph without weights assigned to the edges is called **unweighted**.

If the graph is **weighted** and **directed**, the (directed) weight w_{ij} can usually be interpreted as a *degree of endorsement*, *credit*, *reward*, or *dependency* of object i relative to object j, which defines a binary (weighted) relation between pairs of nodes, where the starting node delivers some kind of "credit" to the ending node. For example, this occurs in a citation network where papers cite other papers or in social organizations where employees depend on their direct managers.

By contrast, for some directed networks, the weights on the edges may instead reflect a relation that involves the *dominance* or *influence* of i on j. For instance, these weights could be obtained from a tournament where the corresponding relation could

be "has defeated" or "is stronger than." Thus, the weights are set to the (positive) score differential between the two opponents. In this situation, reverting the relation and thus the link to $j \to i$ brings us back to the first interpretation, that is, endorsement or dependency. In the sequel, if not explicitly stated otherwise, the first interpretation is assumed for directed graphs. Of course, it is very important to interpret and understand the correct relation that has been captured in a network before its analysis because each algorithm makes implicit assumptions about the semantics of the edges. Sometimes, as with undirected graphs, a nonnegative directed cost c_{ij} is specified in addition to the weights or to replace the weights.

A graph can also contain **self-loops**, which is an edge that may be weighted, starting at one node and ending at the same node. In the sequel, unless stated otherwise explicitly, it is assumed that a graph does not contain self-loops. A graph without self-loops and without multiple edges between two nodes is often called a **simple graph**.

In an unweighted undirected graph, the **degree** of a node is the number of edges incident with it, or equivalently, the degree of a node is the number of nodes adjacent to it. The degree ranges from a minimum of 0 if no node is adjacent to a given node, to a maximum of $n - 1$ if the given node is adjacent to all other nodes in the graph. In the case of a weighted graph, the **generalized degree** (sometimes called the **strength**), or simply the **degree**, of a node is the sum of the weights (the total weight) of the edges incident with it. A node with a degree equal to 0 is called an **isolated node**.

For directed graphs, **indegrees** and **outdegrees** must be introduced, where the indegree of a node is the number of incoming edges (or the total weight for a weighted graph) ending at the considered node and its outdegree is the number of outgoing edges (or the total weight for a weighted graph) starting from the considered node.

A **subgraph** H is a subset of a graph's nodes that, together with the subset of edges of G connecting the nodes in H, also constitutes a graph. Many computational tasks involve identifying subgraphs of various types.

A **path** $\wp$ (sometimes called a **walk**) in a graph is a sequence of edges where each successive node (after the first) is adjacent, through an existing edge, to its predecessor in the path. A path between i and j is denoted by $i \rightsquigarrow j$ or $\wp_{ij}$. A **cycle** or **loop** is a path for which the starting node is equal to the ending node. The set of all possible paths of G, differing in terms of **length** (i.e., number of hops or steps when following the path), starting from node i and ending at node j (including cycles), is denoted by $\mathcal{P}_{ij}$, and the set of all t-steps paths of G starting from node i and ending at node j (including cycles) is denoted by $\mathcal{P}_{ij}(t)$.

In an unweighted graph, the **geodesic** or **shortest-path** distance between two nodes is defined as the length of a minimum length path between them. For weighted graphs, the shortest path refers to the path for which the total accumulated *cost* along the path is a minimum. The **shortest-path** (or **lowest-cost**) **distance** is then defined as this minimum accumulated cost. Now, if there is no path between two nodes, then the distance between them is considered as infinite (or sometimes undefined) as they are not reachable. Note that, in an undirected graph, a shortest path between nodes i and j is also a shortest path between nodes j and i. In some cases, we are interested in **simple paths**, that is, paths that do not include repeating nodes, so the nodes are all distinct. In other words, each node does not appear more than once in a simple path; therefore, simple paths have length at most $n - 1$ in a graph with n nodes.

Let us now define the concept of a **bipartite graph** (or bigraph). Bipartite graphs are encountered frequently, such as in collaborative recommendation problems where there are two different types of nodes (e.g., customer nodes and item nodes). For example, there is a link between a customer i and an item j if the customer bought this item. A bipartite graph is a graph where the nodes can be divided into two disjoint sets $\mathcal{X}$ and $\mathcal{Y}$ such that each edge links a node in $\mathcal{X}$ to a node in $\mathcal{Y}$ or vice versa. In other words, for all of the edges $i \to j \in \mathcal{E}$, either $i \in \mathcal{X}$ and $j \in \mathcal{Y}$ or $i \in \mathcal{Y}$ and $j \in \mathcal{X}$, with $\mathcal{X} \cap \mathcal{Y} = \varnothing$ and $\mathcal{X} \cup \mathcal{Y} = \mathcal{E}$. Consequently, no edge connects two nodes in $\mathcal{X}$ or connects two nodes in $\mathcal{Y}$. It can be shown that a bipartite graph is a graph that does not contain any odd-length cycle [810].

A graph is **connected** (or **strongly connected** if directed) if at least one path exists from each node to each other node in the graph. Stated otherwise, each node is reachable from each other node. A graph that is not connected is **disconnected**, and it comprises a set of **connected components**, which are maximal connected subgraphs. The term *maximal connected subgraph* H means that

- the subgraph H is connected
- there is no path from a subgraph node in H to any other node in the graph that is not part of the subgraph H, and thus
- this subgraph H is maximal, that is, it contains the largest number of nodes and edges that have this property

Many algorithms are available for identifying the maximal connected subgraphs (e.g., see [706] and Section 1.3.3).

The **diameter** of a connected graph G is the distance of the largest shortest path between any pair of nodes in G, providing, therefore, information about the distance between the two farthest nodes in the graph. In an unweighted graph, the diameter can range from a minimum of 0 (a single isolated node) to a maximum of $n - 1$.

In some cases, associated **features** on the nodes provide information about the object the node represents. For instance, if we consider a graph of a social network like Facebook, each node represents a person, and the associated features on the node are simply information that the person has published, for example, the person's gender and age. All of these features are gathered in a **feature vector x**, which usually contains missing values when the person has not published the corresponding information. Then, each node i has a feature vector $\mathbf{x}_i$, which contains its features.

An illustrative example. The illustrative graph shown in Figure 1.4 is unweighted and undirected, and it is defined by

- $\mathcal{V} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- $\mathcal{E} = \{(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 7), (3, 4), (3, 5), (3, 7), (4, 5), (5, 7), (5, 9), (6, 9), (6, 10), (8, 9), (8, 10), (9, 10)\}$,

thereby leading to $n = 10$ and $e = 18$.

Moreover,

- the graph is connected (but it would not be connected if there were no edge between nodes 5 and 9, or if node 5 or node 9 is removed)