2

GRAPH THEORY

FALK SCHREIBER

2.1 INTRODUCTION

The term network is an informal description for a set of elements with connections or interactions between them. A typical example from biology is a protein interaction network. It consists of a set of proteins (elements) and a set of interactions between them (connections). The previous chapter introduced many different biological networks. Given such networks, we could be interested in questions such as Which protein has the highest number of interactions with other proteins in a protein interaction network? Are there clusters of proteins where every protein interacts with every other? Or, in a metabolic network, we might like to study the shortest path of reactions that transform one metabolite into another. Such questions can be answered if we analyze the structure of the network, that is, the way the elements are connected.

To deal with networks in a formal way they are modeled as graphs. A graph is a mathematical object consisting of vertices and edges representing elements and connections, respectively. This usage of the term “graph” should not be confused with another meaning often used in biology: the graphical representation of a function in the form of a curve or surface. The theory of graphs reaches back to Leonard Euler and his “Königsberg bridge problem” in 1736. The problem is as follows: In Königsberg (today Kaliningrad), the river Pregel runs through the town as shown in Fig. 2.1. Seven bridges were built over the river. The question is whether it is possible to walk around the town in a way that would involve crossing each bridge exactly once. By
analyzing the structure of the graph representing the problem, as shown in Fig. Euler proved that this is not possible.

This chapter gives an introduction to most of the mathematical and computer science terminologies used later in the book. It is aimed at readers not familar with these topics, and formal concepts are restricted to a minimum. Readers with prior knowledge may wish to skip this chapter. More detailed presentations can be found in many good textbooks about graph theory, network analysis, and algorithms. Here, we discuss basic terminology and notation for graphs. Section 2.2, special graphs used in modeling biological networks in Section 2.4, and some fundamental algorithms for the analysis of graphs in Section 2.5.

### 2.2 BASIC NOTATION

#### 2.2.1 Sets

A set $A = \{a_1, a_2, \ldots, a_n\}$ is a collection of distinct objects $a_1, a_2, \ldots, a_n$ considered as a whole, and can be defined by listing its elements between braces. An example is the set $A = \{6, 3, 4, 2, 1\}$ of numbers. The objects $a_i$ of a set $A$ are called its members. In case an object is a member of a set this is symbolized by $\in$. For example, if set defined above 2 is a member of $A$, written $2 \in A$. Two sets $A_1$ and $A_2$ are equal (written $A_1 = A_2$) if every member of $A_1$ is a member of $A_2$, and every member of $A_2$ is a member of $A_1$. If every member of set $A_1$ is a member of set $A$ but not necessarily every member of $A_2$ is a member of $A_1$ then the set $A_1$ is a subset of $A_2$, written $A_1 \subseteq A_2$. Two sets $A_1$ and $A_2$ can be combined into a new set, called the union of the sets $A_1$ and $A_2$, the set of all objects that are members of either $A_1$ or $A_2$ and is denoted by $A_1 \cup A_2$. The intersection of the sets $A_1$ and $A_2$ is the set of objects that are members of both $A_1$ and $A_2$ (denoted by $A_1 \cap A_2$). An empty set is denoted by $\emptyset$. Special sets used in this book are the set of natural numbers including zero ($\mathbb{N}$), the set of integers ($\mathbb{Z}$), and the set of real numbers ($\mathbb{R}$).

#### 2.2.2 Graphs

A graph $G = (V, E)$ consists of a set of vertices (also called nodes or points) $V$ and a set of edges (arcs, links) $E$, where each edge is assigned to two (not necessi}
disjunct) vertices. An edge $e$ connecting the vertices $u$, $v$ is denoted by $[u, v]$, we say $u$ and $v$ are incident with $e$ and adjacent (or neighbors) to each other. The vertices incident to an edge are called its end-vertices. The degree of a vertex $v$ is the number of edges that have $v$ as end-vertex. An edge where the two end-vertices are the same vertex is called a loop. A loop-free graph does not contain loops.

This definition describes undirected graphs, that is, graphs where connections between vertices are without a direction. Undirected graphs are used, for example, to model protein interaction networks (see Chapter 9), phylogenetic networks (see Chapter 11), and correlation networks (see Chapter 13). In the following, we describe general graph concepts based on undirected graphs. Section 2.3 deals with other types of graphs, especially directed graphs that are used, for example, to model gene regulation networks (see Chapter 8) and ecological networks (see Chapter 12).

The usual way to visualize a graph is by drawing a point for each vertex and a line for each edge that connects the corresponding points of its end-vertices, see Fig. 2.2. It is not important how a graph is drawn, as long as it is clearly visible which pairs of vertices are connected by edges and which not. The positions of the vertices and the drawing of the lines are called the layout of the graph.

A subgraph $G' = (V', E')$ of the graph $G = (V, E)$ is a graph where $V'$ is a subset of $V$ and $E'$ is a subset of $E$. This implies that $E'$ contains only edges with end-vertices in $V'$. If graph $G'$ is a subgraph of graph $G$ and the edge set $E'$ contains all edges of $E$ that connect vertices of $V'$, the subgraph is called an induced subgraph of $G$. See Fig. 2.3 for subgraph and induced subgraph.

One graph can have many different graphical representations, see Fig. 2.2. But two graphs can also be the same, see Fig. 2.4. Both graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ have different vertex and edge sets. Graph $G_1$ consists of

![Graphs](image)

**FIGURE 2.2** Two graphical representations of the graph $G = (V, E)$ with vertex set $V = \{1, 2, 3, 4, 5, 6, 7\}$ and edge set $E = \{[1, 2], [2, 3], [1, 3], [3, 6], [4, 5], [5, 7]\}$.

![Graphs](image)

**FIGURE 2.3** A graph $G$, a subgraph of $G$, and an induced subgraph of $G$ (from left to right).
Two isomorphic graphs:

\[ V_1 = \{1, 2, 3, 4\} \text{ and } E_1 = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{2, 4\}\}, \text{ graph } G_1 \]

\[ V_2 = \{a, b, c, d\} \text{ and } E_2 = \{\{a, b\}, \{b, c\}, \{b, d\}, \{c, d\}\}. \text{ graph } G_2 \]

However, even though both graphs appear to be different, they contain the same number of connected in the same way and are therefore considered as the same or isomorphic graphs. Formally, two graphs \(G_1\) and \(G_2\) are isomorphic, if there exists a bijection \(f\) mapping between the vertices in \(V_1\) and \(V_2\) with the property that any two \(u, v \in V_1\) are adjacent if and only if the two corresponding vertices in the other are adjacent. Such a bijection is called an isomorphism.

A sequence \((v_0, e_1, v_1, e_2, v_2, \ldots, v_k)\) of vertices and edges such that every edge \(e_i\) has the end-vertices \(v_{i-1}\) and \(v_i\) is called a walk. Usually the first edge is omitted and the walk is denoted by a sequence \((v_1, e_2, \ldots, v_k)\). We say the walk connects \(v_0\) with \(v_k\) and call \(v_0\) and \(v_k\) the start- and end-vertex of the walk, respectively. If all edges of a walk are distinct the walk is called a path. Additionally all vertices are distinct the walk is called a simple path. The length of a path is given by its number of edges. A path with the same vertex at its start and end-vertex is a cycle. A graph without cycles is called an acyclic graph. For example in the graph in Fig. 2.2, the sequence \((1, 2, 3, 6, 3, 1)\) is a path whereas \((1, 2, 3, 3, 1)\) is a cycle.

Two vertices of a graph are called connected if there exists a walk between them. If any pair of different vertices of the graph is connected, the graph is called connected. A connected component of a graph \(G\) is a maximal connected subgraph of \(G\). For example, the graph in Fig. 2.5 consists of four connected components.

A shortest path between two vertices is a path with minimal length. The length between two vertices is the length of a shortest path between them or \(\infty\) if such a path exists. For example, in Fig. 2.2, the path \((1, 3, 3, 6)\) is a shortest path between vertex 1 and vertex 6 and the distance between these two vertices is 2. Note that there may be several different shortest paths between two vertices in a graph.

An unconnected graph consisting of four connected components.
2.2.3 Graph Attributes

Often attributes such as text, numerical values, types, colors, and coordinates are associated with the vertices and edges of a graph. Typical examples are the stoichiometry of reactions in metabolic networks represented as numerical values along the edges, protein classes for proteins represented as vertex types or textual vertex labels in protein interaction networks, and the coordinates of the vertices represented as numerical value pairs. Figure 2.6 shows a typical example.

Attributes can be represented as functions from the vertices or edges to the attribute type. For example, the mentioned stoichiometry of metabolic reactions can be represented as edge weights, that is, numerical values connected to edges. The function $\omega : E \to \mathbb{R}$ assigns each edge $e \in E$ a weight $\omega(e)$.

2.3 SPECIAL GRAPHS

There are many different biological networks with different properties. Often the graph model has to be tailored to the specific network under consideration. Typical graph models for the different networks are considered in the following section.

2.3.1 Undirected, Directed, Mixed, and Multigraphs

Graphs can be undirected, directed, or mixed, see Fig. 2.7. In an undirected graph, an edge between the vertices $u$ and $v$ is represented by the unordered vertex pair $\{u, v\}$. The graphs defined in the previous section are undirected. Typical examples from...
biology are protein interaction networks, phylogenetic networks, and correla
networks.

In a directed graph, an edge between the vertices \( u \) and \( v \) is represented by
ordered vertex pair \( (u, v) \). In visualizations of graphs, the direction of an edge
usually represented by an arrowhead. The two edges \( (u, v) \) and \( (v, u) \) between
vertices \( u \) and \( v \) can be represented either by two lines as shown in Fig. 2.7 or by
line with arrowheads at both ends. Typical examples of biological networks in-
elled by directed graphs are metabolic networks, gene regulation networks, and f
webs.

In a mixed graph undirected and directed edges occur. This type of graph is
relevant in biology, an example is special protein interaction networks where s
interactions are undirected (e.g., obtained by two-hybrid experiments) and ot
are directed representing activation, phosphorylation, and other directed inte-
tions.

An undirected edge has end-vertices, a directed edge \( (u, v) \) has a source verti-
t (also called origin or head) and a target vertex \( v \) (destination, tail). In a directed gr
a vertex has an out-degree that is defined as the number of edges going out of it an
in-degree defined as the number of edges coming into it. The degree of the vertex is
sum of its in- and out-degrees. In directed graphs the definitions for walks, paths, c
cycles are similar to undirected graphs, but take the edge direction into account.
e.g., a walk in a directed graph is a sequence \( (v_0, e_1, v_1, e_2, v_2, \ldots, v_k, e_k) \)
of vertices and edges such that every edge \( e_j \) has the source vertex \( v_{j-1} \) and the ta
vertex \( v_j \). We say that in a directed graph the walk \( v_0 \) with \( v_k \)
the edge direction is taken into account, otherwise (i.e., if each edge is consid
undirected) the walk simply \( v_0 \) with \( v_k \). Two vertices of a graph are ca
strongly connected if there exists such a walk between them. If any pair of diffe
vertices of the graph is strongly connected, the graph is strongly connected. A stron
c connected component of a graph \( G \) is a maximally strongly connected subgr
of \( G \).

Multigraphs are graphs containing multiple edges, that is, two or more edges
are incident to the same two vertices and in case of directed graphs have the s
direction. Such edges are also called parallel edges or a multiedge, see Fig. 2.4
an example of a multigraph. Multiple edges are, for example, useful for the mode
of metabolic pathways where the same substances can be transformed by dif
cent reactions. Undirected, loop-free graphs without multiple edges are called sin
graphs.

2.3.2 Hypergraphs and Bipartite Graphs

There are biological networks where more than two elements are connected by
interaction. An example are metabolic networks where often several substances n
with each other to build other substances, see Fig. 2.8. To model such netwo
hypergraphs are used. A hypergraph \( G = (V, E) \) consists of a set of vertices \( V \) a
set of hyperedges \( E \); each hyperedge is a nonempty subsets of \( V \). Hypergraphs
be directed or undirected.
Hypergraphs are not commonly used in graph theory, and many algorithms developed for graphs cannot be directly applied to hypergraphs. Therefore, such graphs are seldom used to model biological networks. Instead these networks are modeled by bipartite graphs, a structure generally used to represent hypergraphs.

A graph $G = (V, E)$ is called bipartite if there is a partition of its vertex set $V = S \cup T$ such that each edge in $E$ has exactly one end-vertex in $S$ and one end-vertex in $T$ (see Fig. 2.9).

To model a hypergraph $G = (V, E)$ by a bipartite graph $G' = (V', E')$ with $V' = S' \cup T'$, the bipartite graph is build in the following way. Each vertex $v \in V$ is represented by a vertex in $S'$ and each hyperedge $e \in E$ by a vertex in $T'$. For each vertex $v \in V$ and each hyperedge $e \in E$ incident with $v$, an edge is inserted into $E'$, which connects a vertex $s \in S'$ representing the vertex $v$ of the hypergraph and a vertex $t \in T'$ representing the hyperedge $e$. Figure 2.10 shows a hypergraph and its representation as a bipartite graph, and Fig. 2.11 shows a typical modeling of metabolic networks by bipartite graphs.

### 2.3.3 Trees

The last type of special graphs we will consider in this introduction are trees. Trees play an important role in biology where they represent, for example, the evolutionary relationships between species as a phylogenetic tree (see Chapter 11).
A tree is an undirected, connected, acyclic graph. The vertices of a tree with degree 1 are its leaves, all other vertices are inner vertices. A rooted tree consists of a graph $G = (V, E)$ and a distinguished vertex $r \in V$ called the root. The depth of a vertex $v$ is the length of the path between the root and this vertex, the height of a tree is maximum depth of a vertex. A binary tree is a tree where each vertex has at most two children. See Fig. 2.12 for a tree and a binary tree.

A rooted tree is often regarded as a directed graph such that all edges are directed away from the root. For a directed tree $G = (V, E)$ and an edge $(u, v) \in E$, the vertex $u$ is the parent of $v$ and $v$ is the child of $u$.

For a connected, undirected graph $G$, a special tree can be computed, the spanning tree $T$ of $G$. The spanning tree $T$ is composed of all the vertices of $G$ and a minimal set of edges (some or perhaps all of the edges of $G$) that connect all vertices.

A tree and a binary tree. Again, there are many different graphical representations of a tree.
FIGURE 2.13  An adjacency matrix representation for the graph shown in Fig. 2.2.

A tree contains a subset of the edges of $G$ that form a tree spanning every vertex of $G$, hence the name spanning tree.

2.4 GRAPH REPRESENTATION

To use graphs in a computer program, we have to represent them in the computer. There are two common representations: adjacency matrix and adjacency list. The choice of one or the other depends on the operations needed to deal with the graph and whether the graph is dense or sparse. We will discuss this aspect in Section 2.5.

2.4.1 Adjacency Matrix

A graph $G$ with $n$ vertices can be represented by a $(n \times n)$ adjacency matrix $A$, see Fig. 2.13 for an example. The rows and columns correspond to the vertices and a matrix-element $A_{ij} = 1$ if and only if there is an edge between the vertices $v_i$ and $v_j$ and $A_{ij} = 0$ otherwise. The adjacency matrix of an undirected graph is symmetric, that is, $A_{ij} = A_{ji}$.

The simplest way to implement an adjacency matrix is as an array $[1 \ldots n, 1 \ldots n]$ of numbers or boolean values. Adjacency matrices are often used to represent biological networks as their structure is very simple and matrix operations can be directly applied. However, adjacency matrices need a lot of memory, $n^2$ places for a network with $n$ elements. Furthermore, several algorithms have a longer running time if they are based on this network representation. In particular for graphs with a low number of edges in relation to the number of vertices, another representation, the adjacency list, is usually more efficient.

2.4.2 Adjacency List

A graph $G$ with $n$ vertices can be represented by $n$ lists, see Fig. 2.14 for an example. For each vertex $v \in V$, a list $L_v$ contains all edges incident to this vertex (and therefore all vertices adjacent to it).

A common way to implement an adjacency list is an array $[1 \ldots n]$ of lists.
2.5 GRAPH ALGORITHMS

Many problems concerning biological networks can be answered using standard graph algorithms. Let us consider some of the questions raised in the introduction of this chapter. The protein with the highest number of interactions can be found by visiting all vertices of the graph and counting for each the number of its neighbors. A shortest path between two elements in a metabolic network can be computed with the Dijkstra algorithm (e.g., see [6]). This section gives an introduction to graph algorithms and discusses how to make a good choice between different algorithms computing the same result.

2.5.1 Running Times of Algorithms

Running time and memory requirement are key properties of an algorithm. They are usually specified in the $O$ notation. This is a theoretical measure of the algorithm's running time or space needed for a given size $n$ of input data. For networks, the problem size $n$ is often the number of elements.

We will focus on the running time of algorithms. The $O$ notation is used to compare the running times of algorithms for a large enough problem size and to decide whether an algorithm and a related data structure are adequate or will always be too slow for a large problem size. Formally, for two functions $f, g$ we say that $f$ is in $O(g)$ if there are positive constants $n_0 \in \mathbb{N}$ and $c \in \mathbb{R}$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

Let us consider an example. We compare two typical sorting algorithms, QuickSort and BubbleSort (e.g., see [7]). Both use different strategies to sort a set of $n$ unsorted items. We will not discuss these strategies; however, QuickSort's running time is $O(n \log n)$ on average, whereas BubbleSort needs $O(n^2)$. For small sizes of input data, the running times of both algorithms do not differ much. If we want to sort one million elements, QuickSort may still give us the result a reasonable time, whereas BubbleSort may take an excessively long time even on a supercomputer.

We want to sort all vertices of a graph depending on the number of neighbors vertex has. Let $n$ be the number of vertices and $m$ be the number of edges of the graph. The running times above are only the times for the sorting of a set of unsorted elements. Now let us consider how the graph representation may influence the time to sort all vertices depending on the number of neighbors a vertex has. For this number of neighbors of each vertex has to be computed first. The time needed for
counting depends on the chosen representation. First, let us consider the adjacency matrix: For each vertex we have to test all elements of the row representing it and add 1 to the number of neighbors if the matrix element is 1. To count the number of neighbors for one vertex, we need $O(n)$, and to compute the number of neighbors for all vertices, we need $O(n^2)$. Now consider the adjacency list: Given a vertex we have to test each element in the corresponding list and add 1 to the number of neighbors for each. The length of the list may be different for each vertex; however, there are a total of $m$ edges and therefore a total of $m$ list items. The running time to count the numbers of neighbors for all vertices using the adjacency list is $O(n + m)$.

The adjacency list representation is therefore much better for counting the neighbors of vertices in graphs containing a low number of edges but high number of vertices. Furthermore, combining a poorer graph representation for a specific problem such as the adjacency matrix representation for counting neighbors with the QuickSort algorithm means that the overall running time is no longer in $O(n \log n)$ but in $O(n^2)$.

### 2.5.2 Traversal

Graph traversal algorithms are used to visit all vertices and subsequently perform an action with each vertex. The vertices may be visited in an arbitrary order, or a specific order may be requested. For example, we could be interested in visiting genes in a gene regulation network in an order that follows the regulatory steps. There are many different possibilities to traverse the vertices of a graph, the most important ones are depth first search (DFS) and breadth first search (BFS).

The DFS algorithm shown below works as follows. In the beginning all vertices are marked as unvisited. The algorithm starts with a given vertex, visits this vertex, and then recursively visits all neighbors of this vertex, see Fig. 2.15.

**depth first search component algorithm** (vertex $v$)

1. visit $v$;
2. mark $v$ as visited;
3. for each edge $\{v, w\}$
   - if $w$ is unvisited
     - depth first search component algorithm($w$)

This algorithm visits all vertices within one connected component. To visit all components in an unconnected graph, an enclosing loop is needed:

**depth first search algorithm** (vertex $v$)

1. for each vertex $w$
   - if $w$ is unvisited
     - depth first search component algorithm($w$)

If the graph is represented by an adjacency list the running time is $O(n + m)$. For a representation by an adjacency matrix the running time is $O(n^2)$. Modifications of
2.6 SUMMARY

Biological networks are commonly represented by graphs, and different graph models are used for specific networks. This chapter gives a brief yet concise introduction to most of the graph-related terminology used in the book and presents some simple algorithms for graphs.

The first part of this book discusses graph-based analysis methods in more detail. The focus is on the following topics.

Global network properties: The structure of biological networks is significantly different from random networks, and new models have been introduced [1, 10]. Chapter 3 deals with the analysis of global network properties and relevant models for biological networks.

Network centralities: In biological networks some vertices are often more important or central than others. For example, highly connected vertices in protein interaction networks can be functionally important and the removal of such vertices is related to lethality [5]. Centrality indices are used to rank vertices, and Chapter 4 presents some of the more important centrality indices and their application to biological networks.

Network motifs: A way to understand complex biological networks is to break them down into units of commonly used network architecture. Such patterns of local interconnection are called network motifs [8]. They have been found in many different networks, but are particularly important for the understanding of signal transduction and gene regulation networks. Chapter 5 discusses network motif analysis and presents insights gained with this method.

Network clustering: Biological networks are hierarchically structured from network motifs and pathways at the lowest level, to functional modules, to the large-scale organization of the networks [9]. Chapter 6 studies the clustering of network elements into modules and their application to biological networks.

2.7 EXERCISES

1. Different biological networks are modeled by different graphs. Which types of graphs are typically used to model the following networks: gene regulation networks, protein interaction networks, metabolic networks?

2. Consider the undirected graph $G = (V, E)$ shown in Fig. 2.2 (right). For each vertex $v \in V$ do the following:
   (a) Compute the degree of $v$.
   (b) List all neighbors of $v$.
   (c) Find paths to all other vertices that are in the same connected component as the vertex $v$.

3. For the graph shown in Fig. 2.2 (right), find a different graphical representation of this graph and show how the graph is represented using an adjacency matrix and an adjacency list representation.
FIGURE 2.15  Depth first search of a graph. A vertex already visited is marked by its numb
(in the order of the visits), the currently visited vertex is represented by a circle, and the current
chosen edge to the next neighbor by a thick gray line.

the depth first search algorithm can be used to solve a number of problems such as 1
the network connected or are there separate parts? Or, is the network a tree or does
contain cycles?

Another classic algorithm for the traversal of graphs is BFS. Whereas DFS follow
a path into the graph as long as possible, BFS visits all neighbors of a vertex before
it visits other vertices. The algorithm for BFS is as follows:

**breadth first search component algorithm (vertex v)**

visit(v):
mark v as visited;
queue Q = {v};
while Q is nonempty |
remove vertex w from the front of queue Q;
visit(w);
for each neighbor x of w
if (x is unvisited) |
mark x as visited;
add x to the end of queue Q
}

Again, this algorithm visits all vertices within one connected component. To vis-
all components in a unconnected graph, an enclosing loop similar to DFS is needed.
4. Take a graph with nine vertices, four of them of degree 2 and four of degree 3; is this graph connected?

5. An undirected loop-free graph with \( n \) vertices has at most \( n(n-1)/2 \) edges. Is this statement correct? Can you prove it?

6. Draw an undirected, connected graph \( G \) with 10 vertices and 20 edges. Construct two different spanning trees of \( G \).

7. Metabolite networks can be constructed from metabolic networks modeled by bipartite graphs. Bipartite graphs consist of two sets of vertices where each vertex in one set is connected to all vertices in the other set. Construct a metabolite network from the metabolic network given in Fig. 2.11 (right).

8. Apply the algorithms DFS and BFS to traverse the graph in Fig. 2.2 (right). Start with vertex 1, then apply the algorithms again starting with vertex 5.

9. A Eulerian path in a graph is a path \( (v_0, e_1, v_1, e_2, v_2, \ldots, v_k-1, e_k, v_k) \) in an undirected graph that contains each edge of the graph exactly once. Write an algorithm to check whether an undirected graph \( G = (V, E) \) has an Eulerian path.

REFERENCES


