Part I

Introduction

In the first part of this book we give an introduction to basic concepts from graph theory and systematics (Chapter 1). We briefly discuss the problem of aligning molecular sequences (Chapter 2) and give a more detailed introduction to the computation of phylogenetic trees from aligned sequences and distances (Chapter 3). Finally, we give a brief introduction to the computation of phylogenetic networks, which also serves as an overview for the material presented in the second and third parts of the book (Chapter 4).

Chapters 2 and 3 are provided for the sake of completeness and reference. They can be skipped by readers who have a basic knowledge of phylogenetic analysis.

1

Basics

In this chapter we introduce some basic concepts and results from mathematics and biological taxonomy.

1.1 Overview

The focus of this chapter is on the introduction of *graphs* (see Figure 1.1). Graphs come in two flavors, *undirected* and *directed*. One type of undirected graph that plays an important role in this book is *unrooted trees*. In the context of directed graphs, we discuss the concept of a *directed, acyclic graph (DAG)* and then introduce *rooted trees* as an important example. We introduce a number of different kinds of *traversals* of trees or DAGs that are often used in algorithms. This chapter concludes by introducing the concepts of *taxa*, *clusters*, *clades* and *splits*.

1.2 Undirected and directed graphs

A graph is a convenient and popular way of representing relationships between objects: in a graph, objects are represented by *nodes* and relationships between objects are represented by *edges*. There are two types of graphs, *undirected* and *directed*, and we make use of both types throughout the book. First, we define undirected graphs:

Definition 1.2.1 (Undirected graph) *An* undirected graph $G = (V, E)$ *consists of a finite set of* nodes V *and a finite set of* edges E*, where each* edge $e \in E$ *is of the form* $\{v, w\}$*, with* $v, w \in V$ *(see Figure 1.2a).*

For an edge $e = \{v, w\}$ we call v and w the *endpoints* of e and we say that e *connects* v and w. We also say that both v and w are *incident* to e. We say that two edges are *adjacent*, if they share an endpoint, and two nodes are adjacent if they are connected by an edge. The *degree* of a node v is the number of edges that are incident to v.

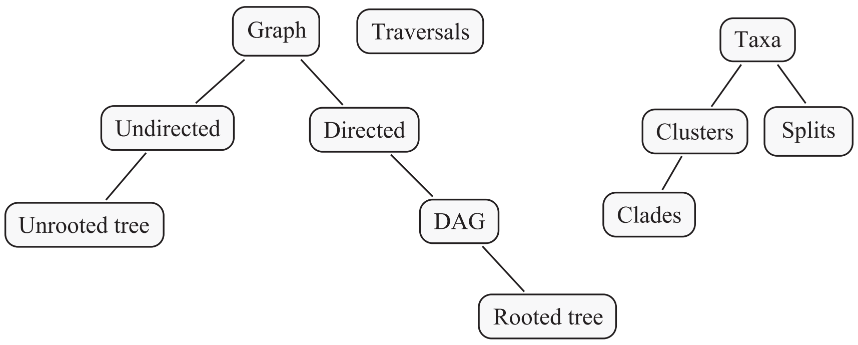

Figure 1.1 Overview of the main concepts introduced in this chapter.

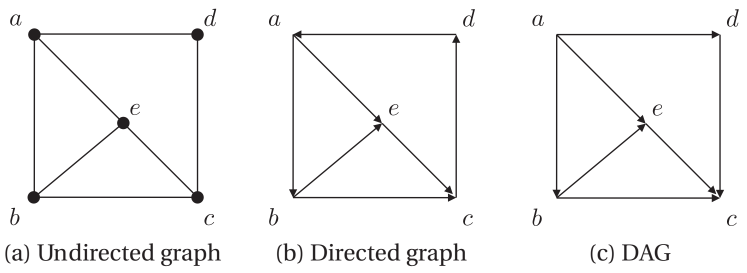

Figure 1.2 (a) An undirected graph with five nodes labeled $a, \ldots, e$. Here, each node is shown as a small disk •, however, we usually refrain from emphasizing nodes in this way. Edges are represented by lines connecting the incident nodes. (b) A directed graph with five nodes $a, \ldots, e$. An arrow from node v to node w represents the directed edge (v, w). (c) A DAG (directed acyclic graph) in which all edges are directed away from the node a.

Unless otherwise stated, we usually assume that each pair of nodes is connected by at most one edge (thus excluding so-called *multi-edges*) and that every edge is incident to two different nodes (thus excluding so-called *self-loops*). Graphs with these two additional properties are called *simple*.

So, in an undirected graph, an edge consists of an unordered pair of two different nodes. In a *directed graph*, an edge consists of an *ordered* pair of nodes and thus has a *direction*:

Definition 1.2.2 (Directed graph) *A* directed graph *(or* digraph *for short)* $G = (V, E)$ *consists of a finite set of* nodes V *and a finite set of* edges E, *where each edge* $e \in E$ *is of the form* (v, w), *with* $v, w \in V$ *(see Figure 1.2b).*

In a directed graph we say that the edge $e = (v, w)$ is *directed* from v to w, and we call v the *source* and w the *target* of e. We say that e is an *out-edge* of v and an *in-edge* of w. The *indegree* of a node u is the number of in-edges of u and the *outdegree* of u is the number of out-edges of u. The *degree* of u is the sum of its indegree and outdegree.

Let $G = (V, E)$ be a directed or undirected graph. A *subgraph* $G' = (V', E')$ of a graph G is a graph whose node and edge sets are subsets of those of G, that is, $V' \subseteq V$ and $E' \subseteq E$, such that the edges in E' only contain nodes in V'. Let U be a subset of nodes of G. The subgraph $G|_U = (U, E|_U)$ *induced* by U has node set U and *induced* edge set $E|_U$ consisting of all edges in G whose endpoints both lie in U.

A *path* (or *undirected path*) P of length $\text{len}(P) = k \geq 1$ is a sequence of nodes and edges

$$P = (v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k) \tag{1.1}$$

such that the nodes v_{i-1} and v_i are the endpoints of the edge e_i, for $1 \leq i \leq k$, and no edge occurs more than once in P. Such a path is said to *connect* its two *endpoints* v_0 and v_k.

In some settings, we may understand a path P to consist only of the edges that it contains, in other settings it may be taken to contain only its nodes.

A *cycle* (or *undirected cycle*) is a path in which the first node equals the last node $v_0 = v_k$, and for which this is the only node that occurs more than once. An undirected graph is called *acyclic* if it contains no cycles. In a directed graph we can additionally define directed paths and cycles: a *directed path* is a path P in which every edge e_i is directed from v_{i-1} to v_i, for $i = 1, \ldots, k$. Similarly, a *directed cycle* is a cycle with this property. Note that in a directed graph it may happen that two nodes v and w are connected by a path, but not by a directed path. For example, in Figure 1.2c, nodes b and d are connected by a path, but not by a directed path. A directed graph is called *acyclic* if it contains no directed cycles. Directed acyclic graphs play an important role for phylogenetic networks and thus are discussed later in more detail.

Let G be a directed graph. When necessary, we can also think of G as an undirected graph, simply by ignoring the direction of its edges. A special case arises when there are two nodes v and w such that both (v, w) and (w, v) are edges of G. In this case, the two directed edges are replaced by a single undirected one $\{v, w\}$. Similarly, if G is an undirected graph, then we can convert G into a directed graph by choosing a direction for each edge in G.

When modifying a graph $G = (V, E)$, to *delete* an edge e means simply to remove e from E. However, to *delete* a node v, one must remove v from V and also delete all edges that are incident to v. If v is a node of degree two in an undirected graph, then v is called *suppressible* and to *suppress* v means to first connect the two nodes adjacent to v by a new edge and then to delete v. Similarily, in a directed graph, a node v can be suppressed when it has both indegree one and outdegree

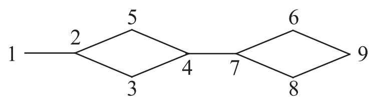

Figure 1.3 This graph G has three cut nodes, namely 2, 4 and 7. The graph has four biconnected components, containing the node sets $\{1, 2\}$, $\{2, 3, 4, 5\}$, $\{4, 7\}$ and $\{6, 7, 8, 9\}$, respectively. The graph is bipartite, as every edge connects an even-numbered node to an odd-numbered one.

one, by first connecting the source node of the in-edge to the target node of the out-edge by a new edge and then deleting v.

To *contract* an edge $e = (v, w)$ means to delete e and then to merge the two nodes v and w, in other words, to replace the two nodes v and w by a single node u and to change all edges incident to v and w so that they become incident to u, instead. A graph $G = (V, E)$ is called a *refinement* of another graph $G' = (V', E')$ if we can contract edges of G in such a way that the resulting graph equals G'.

A graph $G = (V, E)$ is called *bipartite* if and only if its set of nodes can be partitioned into two subsets V_1 and V_2, with $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$, such that for every edge $e \in E$ one of the endpoints lies in V_1 and the other endpoint lies in V_2. Equivalently, a graph is bipartite if one can color the set of nodes using two colors such that the two endpoints of any edge have different colors.

Any two nodes v and w of G are said to be *connected*, if there exists an undirected path that starts at v and ends at w. A graph $G = (V, E)$ is called *connected* if every pair of nodes $v, w \in V$ is connected by some undirected path. A *connected component* of a graph G is a maximal connected subgraph. For example, the graph displayed in Figure 1.4 has precisely three connected components. A *cut node* or *cut edge* is a node or edge, respectively, whose removal disconnects the graph.

In the literature a graph G is sometimes also called *weakly connected*, if any two nodes are joined by an undirected path, and *strongly connected*, if any two nodes are joined by a directed path. Throughout this book we only make use of the former definition and refer to such graphs as connected.

A *biconnected component* is a maximal subgraph that is induced by a set of edges and does not contain a cut node. In consequence, a graph G is called a biconnected component, if either it consists of only one node, or it consists of two nodes joined by a single edge, or it has more than two nodes and any two nodes v and w are connected by at least *two* different paths that are node-disjoint (except at v and w). For an example of this concept, and some of the others just defined, see Figure 1.3.

Exercise 1.2.3 (Number of cut nodes and components) *Determine the number of cut nodes and biconnected components in Figure 1.4. Is the graph bipartite?*

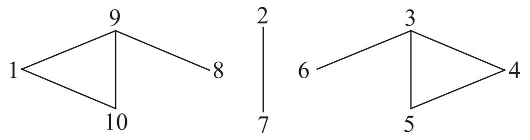

Figure 1.4 Graph with ten nodes labeled 1–10 and nine edges.

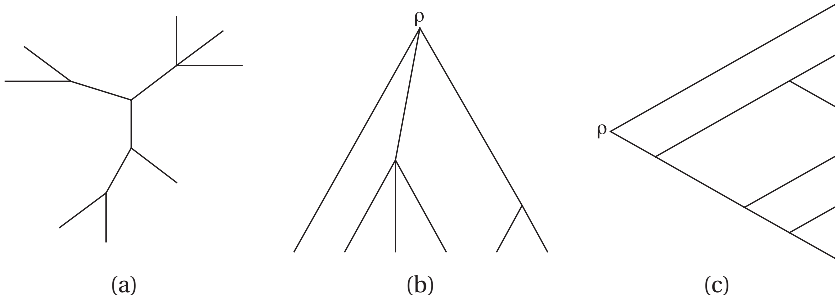

Figure 1.5 (a) An unrooted tree with 13 nodes and 12 edges. Exactly 8 nodes are leaves and 5 are internal nodes. (b) A rooted tree with 9 nodes and 8 edges, the root here being the top-most node ρ. (c) A rooted tree with 11 nodes and 10 edges, the root here being the left-most node ρ. This tree is bifurcating, as all internal nodes have outdegree 2.

Throughout this book, we always refer to the constituents of a graph as *nodes* and *edges*, but remark that in the literature nodes are also called *vertices* (usually when they are contained in an undirected graph) and edges are also called *arcs* (usually when they are directed) or branches (usually when they are contained in a *phylogenetic tree*).

1.3 Trees

A tree is a special kind of graph:

Definition 1.3.1 (Tree) *A* tree *is a connected graph with no (undirected) cycles.*

In this book, we distinguish between *unrooted* and *rooted* trees. By an *unrooted tree* we mean any graph satisfying Definition 1.3.1, see Figure 1.5a. In a *rooted tree T*, one node ρ is declared the *root*, see Figure 1.5b. In the case of such a *rooted tree* $T = (V, E, \rho)$, we usually think of the graph as being a directed graph in which all edges are directed away from the root.

In an unrooted tree, all nodes of degree 1 are called *leaves* and all other nodes are called *internal nodes*. In a rooted tree, all nodes of outdegree 0 are called *leaves*, all nodes of outdegree ≥ 1 are called *internal nodes* and there is only one node with indegree 0, namely the root ρ.

When drawing rooted trees, the directions of the edges are not indicated explicitly, e.g. by drawing arrows, but rather implicitly by special placement of the root, usually at the top, left or bottom of the drawing, with the tree extending to the bottom, right or top, respectively, see Figure 1.5b–c.

An unrooted tree is called *bifurcating* if every internal node has degree 3. A rooted tree is called *bifurcating* if every internal node v has outdegree 2, see Figure 1.5c. Nodes that violate this condition are called *multifurcating* nodes. A bifurcating tree is also called a *resolved tree*.

Definition 1.3.2 (Subtree rooted at v) *Let T be a rooted tree and v a node in T. We define the* subtree rooted at v *to be the subgraph T_v of T that is induced by the set of nodes consisting of v and all nodes that lie on directed paths starting at v. The root of T_v is v.*

Exercise 1.3.3 (Equivalent definitions of a tree) *Prove that the following statements are equivalent for any undirected graph $T = (V, E)$:*

(i) *T is a tree.*
(ii) *Any two nodes $v, w \in V$ of T are connected by exactly one path P.*
(iii) *T is connected and it is not possible to delete any edge without producing a graph that is not connected.*
(iv) *T is connected and the number of nodes exceeds the number of edges by one, that is $|V| = |E| + 1$.*
(v) *T is acyclic and the number of nodes exceeds the number of edges by one, that is $|V| = |E| + 1$.*
(vi) *T is acyclic and it is not possible to add an edge without creating a cycle.*

1.4 Rooted DAGs

Just as trees provide the mathematical basis for phylogenetic trees in Chapter 3, a generalization of rooted trees called rooted *DAGs* forms the basis for rooted phylogenetic networks presented in Chapter 6. A DAG is a special kind of directed graph:

Definition 1.4.1 (Rooted DAG) *A* directed acyclic graph (DAG) *is a directed graph that is free of directed cycles. A DAG is called* rooted *if it contains precisely one node of indegree* 0*, called the* root *(see Figure 1.2c).*

If G is a DAG, then for any two nodes v and w we call v an *ancestor* of w, and w a *descendant* of v, if there exists a directed path from v to w. We call v a *parent* of w, and w a *child* of v if there exists an edge $e = (v, w)$ from v to w. If w is a descendant of v, then we also say that w is *lower* than v. It follows from the definition of a DAG that a node v cannot be simultaneously both a descendant and also an ancestor of any other node w. Similarly, we say that an edge $e = (v, w)$ is *lower* than an edge $f = (v', w')$, if v is lower than w'. We say that two nodes v and w are *incomparable*, if neither node is lower than the other. Similarly, two edges e and f are called *incomparable* if neither is lower than the other.

Lemma 1.4.2 (All nodes are descendants of the root) *In a rooted DAG all nodes are descendants of the root ρ. In particular, a rooted DAG is always (weakly) connected.*

Proof Consider some node v. If v is not the root, then there exists a parent v_1 of v. Again, either v_1 is the root, or v_1 has a parent v_2. Repeating this argument we obtain a chain of nodes $v, v_1, v_2, \ldots, v_r$. Because G is acyclic, any two nodes in this chain are distinct. As the set of nodes of G is finite, the chain must end at some node with indegree 0, which is ρ. Following the chain of nodes and edges used from ρ to v provides a directed path. $\square$

A useful characterization of a rooted tree is the following:

Lemma 1.4.3 (DAG with indegrees one is rooted tree) *A DAG is a rooted tree if and only if it contains precisely one node of indegree* 0*, which is the root, and all other nodes have indegree one.*

Exercise 1.4.4 (Proof) *Prove Lemma 1.4.3.*

From this characterization and Lemma 1.4.2, it follows for any DAG G with root ρ that, if we delete all but one in-edge for every node v of indegree ≥ 2 in G, then the remaining graph is a tree with root ρ.

In analogy to the definition of the *subtree rooted at* a node v, we define:

Definition 1.4.5 (Sub-DAG rooted at v) *Let G be a DAG and v a node in G. We define the* sub-DAG *rooted at v as the subgraph G_v of G that is induced by the set of nodes consisting of v and all descendants of v. The root of G_v is v.*

1.5 Traversals of trees and DAGs

The term *tree traversal* refers to the process of systematically examining, that is visiting and processing, every node in a tree exactly once. Such traversals are classified by the order in which the nodes are examined. In the following definition we assume that the trees are rooted and bifurcating. However, all the algorithms can be easily adapted to the case of non-bifurcating trees.

Definition 1.5.1 (Tree traversals) *Let T be a bifurcating rooted tree. We distinguish the following types of* tree traversals*:*

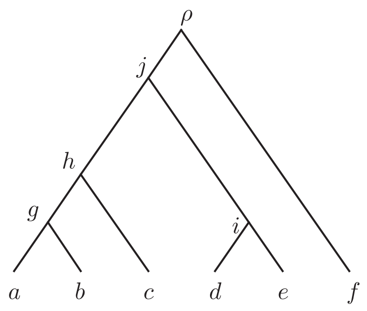

Figure 1.6 A bifurcating tree rooted at node ρ, with other nodes labeled $a, \dots, j$.

Traversal	Order in which nodes are examined
preorder	$\rho, j, h, g, a, b, c, i, d, e, f$
postorder	$a, b, g, c, h, d, e, i, j, f, \rho$
inorder	$a, g, b, h, c, j, d, i, e, \rho, f$
breadth-first	$\rho, j, f, h, i, g, c, d, e, a, b$

Figure 1.7 Different traversals of the tree depicted in Figure 1.6.

- Preorder *traversal: Examine the root of T. Then traverse its left subtree in preorder, then traverse its right subtree in preorder.*
- Postorder *traversal: First traverse the left subtree of the root in postorder, then traverse the right subtree in postorder, and finally examine the root.*
- Inorder *traversal: First traverse the left subtree of the root in inorder, then examine the root, and then traverse its right subtree in inorder.*
- Breadth-first *traversal: Put the root in a queue. Repeat the following until the queue is empty: Pop a node off the beginning of the queue and examine it. Then, if the node has any children, push them onto the end of the queue.*

The different traversals give rise to different orders in which the nodes are examined. For example, consider the tree shown in Figure 1.6. The order in which nodes are examined in each of the traversals is listed in Figure 1.7.

Preorder, postorder and breadth-first traversals apply more generally to rooted DAGs:

Definition 1.5.2 (DAG traversals) *Let G be a rooted DAG. We distinguish between the following types of* DAG traversals*:*

- Preorder *traversal: Examine the root ρ. While ρ has any child v that has not yet been examined, traverse the sub-DAG rooted at v in preorder.*
- Postorder *traversal: While the root ρ has any child v that has not yet been examined, traverse the sub-DAG rooted at v in postorder. Examine ρ.*

- Breadth-first *traversal: Put the root in a queue and mark it. Repeat the following until the queue is empty: Pop a node off the beginning of the queue and examine it. Then, if the node has any children that have not yet been marked, mark them and push them onto the end of the queue.*

Exercise 1.5.3 (DAG for given traversals) *Construct a rooted DAG with nodes labeled $a, b, \dots, i$ such that the three traversals visit the nodes in the following order:*

- *preorder:* $i, a, d, g, h, e, b, f, c$
- *postorder:* $g, h, d, e, a, f, b, c, i$
- *breadth-first:* $i, a, b, c, d, e, f, g, h$

Construct a second rooted DAG that gives rise to the same traversals.

A preorder traversal is also called a *top-down algorithm* and a postorder traversal is also called a *bottom-up algorithm*.

1.6 Taxa, clusters, clades and splits

Throughout this book, we use $\mathcal{X} = \{x_1, \dots, x_n\}$ to denote a set of *taxa*, in which each *taxon* x_i represents some species, group or individual organism whose evolutionary history is of interest to us. For example, $\mathcal{X} = \{x_1, x_2, \dots\}$ might denote a set of mammals, with x_1 representing gorillas, x_2 representing seals, etc.

We use expressions such as "C is a cluster *on* $\mathcal{X}$", "T is a phylogenetic tree *on* $\mathcal{X}$", "D is a distance matrix *on* $\mathcal{X}$" or "A is a multiple sequence alignment *on* $\mathcal{X}$", to indicate that the corresponding set of taxa is $\mathcal{X}$.

Let $\mathcal{X}$ be a set of taxa. A *cluster* is any subset of $\mathcal{X}$, excluding both the empty set $\emptyset$ and the full set $\mathcal{X}$.

A *phylogeny* describes the evolutionary history of a set of taxa and the ultimate goal of any phylogenetic analysis is to reconstruct some part of the *tree of life*, the evolutionary history of all life on Earth.

In this context two species are called *related*, if they share a recent common ancestor. We say that a species a is *more closely related* to a species b than to a species c, if the last common ancestor of a and b is more recent that the last common ancestor of a and c. For example, humans are more closely related to mice (last common ancestor about 100 million years ago) than they are to fruit flies (last common ancestor about 600 million years ago).

A group of organisms is called a *clade* or a *monophyletic group*, if it contains all (and only) the descendants of a common ancestor and the ancestor itself. In a *monophyletic cluster* C (of extant species) on $\mathcal{X}$, all species are more closely related to each other than to any taxon outside of C. In other words, all species in C are descended from a common ancestor, which is not an ancestor of any other taxon