Genome-Scale Algorithm Design

Presenting the fundamental algorithms and data structures that power bioinformatics workflows, this book covers a range of topics from the foundations of sequence analysis (alignments and hidden Markov models) to classical index structures (k-mer indexes, suffix arrays, and suffix trees), Burrows–Wheeler indexes, graph algorithms, network flows, and a number of advanced omics applications. The chapters feature numerous examples, algorithm visualizations, and exercises, providing graduate students, researchers, and practitioners with a powerful algorithmic toolkit for the applications of high-throughput sequencing. An accompanying website (www.genome-scale.info) offers supporting teaching material.

The second edition strengthens the toolkit by covering minimizers and other advanced data structures and their use in emerging pangenomics approaches.

Veli Mäkinen is Professor of Computer Science at the University of Helsinki, where he leads a research team on genome-scale algorithmics. He has taught advanced courses on algorithm design and analysis, string processing, data compression, and algorithmic genome analysis, as well as introductory courses on bioinformatics.

Djamal Belazzougui is a permanent researcher at the Research Centre for Scientific and Technical Information in Algiers, Algeria. He worked in the Genome-Scale Algorithmics group at the University of Helsinki as a postdoctoral researcher from 2012 to 2015. His research topics include hashing, succinct and compressed data structures, string algorithms, and bioinformatics.

Fabio Cunial is a computational scientist at the Broad Institute of MIT and Harvard. He has served as a postdoctoral researcher at the Genome-Scale Algorithmics group at the University of Helsinki as well as at the Myers Lab in Dresden.

Alexandru I. Tomescu is Associate Professor of Algorithmic Bioinformatics at the University of Helsinki, where he leads a research group on graph algorithms and their application to high-throughput sequencing problems. He is the recipient of a Starting Grant from the European Research Council.

Genome-Scale Algorithm Design

Bioinformatics in the Era of High-Throughput Sequencing

Second Edition

VELI MÄKINEN
University of Helsinki

DJAMAL BELAZZOUGUI
Research Centre for Scientific and Technical Information (CERIST), Algiers

FABIO CUNIAL
Broad Institute, Cambridge, MA

ALEXANDRU I. TOMESCU
University of Helsinki

CAMBRIDGE
UNIVERSITY PRESS

Shaftesbury Road, Cambridge CB2 8EA, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

314–321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre, New Delhi – 110025, India

103 Penang Road, #05–06/07, Visioncrest Commercial, Singapore 238467

Cambridge University Press is part of Cambridge University Press & Assessment, a department of the University of Cambridge.

We share the University's mission to contribute to society through the pursuit of education, learning and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781009341233
DOI: 10.1017/9781009341257

First published 2015
Second edition 2023

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging-in-Publication Data
Names: Mäkinen, Veli, author. | Belazzougui, Djamal, author. | Cunial, Fabio, author. | Tomescu, Alexandru I., author.
Title: Genome-scale algorithm design : bioinformatics in the era of high-throughput sequencing / Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, Alexandru I. Tomescu.
Description: Second edition. | New York : Cambridge University Press, 2023. | Includes bibliographical references and index.
Identifiers: LCCN 2023017673 (print) | LCCN 2023017674 (ebook) | ISBN 9781009341233 (hardback) | ISBN 9781009341257 (epub)
Subjects: LCSH: Genomics. | Nucleotide sequence.
Classification: LCC QH447 .M3522 2023 (print) | LCC QH447 (ebook) | DDC 572.8/6–dc23/eng/20230605

ISBN 978-1-009-34123-3 Hardback

To Alberto Apostolico, who kindled our fascination with maps that are much smaller than the empire.

Contents

List of Insights

Preface

High-throughput sequencing has revolutionized the field of biological sequence analysis, both by stimulating the development of fundamentally new data structures and algorithms, and by changing the routine workflow of biomedical labs. Many key analytical steps now exploit index structures based on the Burrows–Wheeler transform, which have been under active development in theoretical computer science for over twenty years. The ability of these structures to scale to very large datasets quickly led to their widespread adoption by the bioinformatics community, and their flexibility continues to spur new applications in genomics, pangenomics, transcriptomics, and metagenomics. Despite their fast and still ongoing development, the key techniques behind these indexes are by now well understood, and they are ready to be taught in graduate-level computer science courses.

This book focuses on the rigorous description of the *fundamental algorithms and data structures* that power standard sequence analysis workflows, ranging from the foundations of biological sequence analysis (like alignments and hidden Markov models) and classical index structures (like k-mer indexes, suffix arrays and suffix trees), to Burrows–Wheeler indexes and a number of advanced *omics* applications built on such a basis. The topics and the computational problems are chosen to cover the actual steps of large-scale sequencing projects, including read alignment, variant calling, haplotyping, fragment assembly, alignment-free genome comparison, compression of genome collections and of read sets, transcript prediction, and analysis of metagenomic samples; see Figure 1 for a schematic summary of all the main steps and data structures covered in this book. Although strongly motivated by high-throughput sequencing, many of the algorithms and data structures described in this book are general and can be applied to a number of other fields that require the processing of massive sets of sequences. Most of the book builds on a coherent, self-contained set of algorithmic techniques and tools, which are gradually introduced, developed, and refined from the basics to more advanced variations.

The book is accompanied by a website

`www.genome-scale.info`

that provides references to additional teaching material covering selected chapters of the book. The website also maintains a list of typos and mistakes found, and we encourage the reader to send corrections as requested therein.

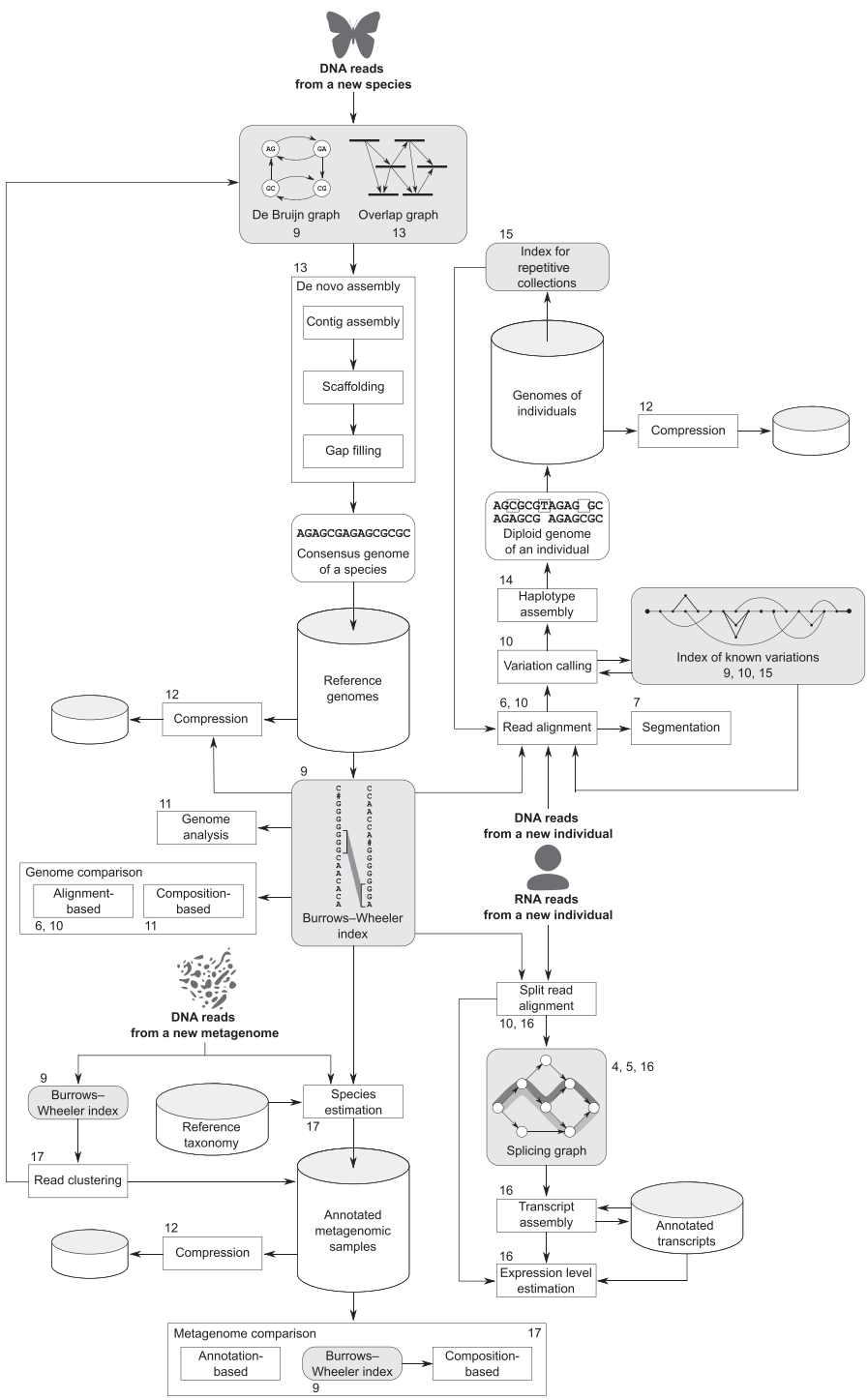

Figure 1 High-level summary of the main computational steps in high-throughput sequencing. Key data structures are highlighted in gray. Cylinders represent databases. Numbers indicate the chapters that cover each step.

This book introduces a number of significant novelties in presenting and organizing its content. First, it raises to a central role the *bidirectional Burrows–Wheeler index*: this powerful data structure is so flexible as to be essentially the only index needed by most sequence analysis primitives, like maximal repeats, maximal unique and exact matches, and alignment-free sequence comparison. In this book we use k-mer indexes, suffix arrays, and suffix trees mostly as *conceptual tools* to help the reader learn the bidirectional Burrows–Wheeler index and formulate problems with its language.

Another key concept that recurs in a large fraction of the book is *minimum-cost network flow*, a flexible model in combinatorial optimization that can be solved in polynomial time. We use this model as a "Swiss Army knife," both by providing a unified presentation of many well-known optimization problems in terms of minimum-cost flow (like maximum-weight bipartite matching and minimum-weight minimum path cover) and by showing that a number of key problems in fragment assembly, transcriptomics, and metagenomics can be elegantly solved by reductions to minimum-cost flow.

Finally, the book spices up the presentation of classical bioinformatics algorithms by including a number of advanced topics (like Myers' bit parallel algorithm) and by presenting inside a unifying framework the dynamic programming concepts that underlie most such algorithms. Specifically, many seemingly unrelated problems in classical bioinformatics can be cast as *shortest-path problems on a directed acyclic graph* (DAG). For example, the book describes the Viterbi algorithm for hidden Markov models as a special case of the Bellman–Ford algorithm, which can itself be interpreted as a solution to the shortest-path problem on a DAG created by layered copies of the input graph. Even the gap-filling problem in fragment assembly is solved through a similar reduction to a DAG problem. The book contains a number of other problems on DAGs, like aligning paths in two labeled DAGs, indexing labeled DAGs using an extension of the Burrows–Wheeler transform, and path covering problems on weighted DAGs arising from alternative splicing.

The book is designed so that key concepts keep reappearing throughout the chapters, stimulating the reader to establish connections between seemingly unrelated problems, algorithms, and data structures, and at the same time giving the reader a feeling of organic unity.

Structure and content

The book adopts the style of theoretical computer science publications: after describing a biological problem, we give precise problem formulations (often visually highlighted in a frame), algorithms, and pseudo-code when applicable. Finally, we summarize our results in one or more theorems, stating the time and space complexity of the described algorithms. When we cannot obtain a positive result in the form of a polynomial-time algorithm, we give an NP-hardness proof of the problem, thereby classifying each problem as either tractable or intractable.

Nonetheless, ad-hoc optimizations and heuristics are important in practice, and we do explain a number of such strategies for many problems. We choose to present these

methods inside a special frame, called *Insight*. An Insight can also contain additional background information, methods explained only by example, mathematical and statistical derivations, and practical bioinformatics issues. Visually separating Insights from the main text puts the main algorithmic concepts in the foreground, helping the reader to focus on them and to potentially skip marginal details.

Every chapter ends with a collection of exercises, of varying difficulty. Some exercises ask the reader just to practice the topics introduced in the chapter. Other exercises contain simple, self-contained parts of long proofs, stimulating the reader to take an active part in the derivations described in the main text. Yet other exercises introduce new problem formulations, or alternative strategies to solve the same problems, or they ask the reader to play with variants of the same data structure in order to appreciate its flexibility. This choice makes some exercises quite challenging, but it allows the main text to stay focused on the key concepts. By solving the majority of the exercises, the reader should also gain a broad overview of the field.

A small number of sections describe advanced, often technical concepts that are not central to the main flow of the book and that can be skipped safely: such sections are marked with an asterisk. The book is designed to be self-contained: apart from basic data structures such as lists and stacks, and apart from basic notions in algorithm complexity, such as the big-oh notation, every chapter of the book builds only on data structures and algorithms that have been described in previous chapters. Therefore, a reader could potentially implement every algorithm we describe, by just reading this book: using the computer science jargon, we could say that the book is entirely "executable." For pedagogical reasons we choose sometimes not to present the most time- or space-efficient algorithm for a problem. In all such cases, we briefly sketch the more efficient variants in the main text, leave them as exercises for the reader, or cite them in the literature section of the chapter.

The book focuses on *algorithm design*. This means that we mainly focus on combinatorial strategies that can be used to solve a variety of different problems, and that we try to find analogies between the solution of every problem and the solution of other problems described in previous chapters. Our focus on design also implies that we do not include in the book any algorithm that requires advanced mathematical tools for analyzing its performance or for proving its correctness: a basic understanding of amortized analysis and of combinatorics is enough to follow the derivations of all worst-case bounds. No average- or expected-case analysis is included in the book, except for a small number of hashing-related derivations and Insights.

A significant part of the book focuses on applications of space-efficient data structures. Research on such structures has been very active over the last 30 years. Our goal was not to delve into the technical data structure fundamentals but to select the minimal setup sufficient to keep the book self-contained. With this aim, we avoided fully dynamic data structures and entropy-compressed data structures. Such technically more advanced structures are now covered in the textbook by Gonzalo Navarro (Navarro, 2016).

Owing to our extensive adaptation of the algorithms in the literature to our self-contained, minimal setup of data structures, we decided to gather all references into a Literature section appearing at the end of each chapter. This enables an undisturbed

reading experience and lets us explain the difference between our description and the original work. To avoid being too influenced by the literature, we often aimed at reproducing a result given just the final time and space complexity and a few hints, like our memories from studying the method or hearing a presentation some years back. Some exercises are directly related to published work as well, and in such cases we mention the reference in the literature section.

Almost all algorithms presented in the book are sequential and are designed to work in random access main memory. By *genome-scale* we mean both a first logical layer of space-efficient and near-linear-time algorithms and data structures that process and filter the *raw data* coming from high-throughput-sequencing machines, and a second layer of polynomial-time algorithms that work on the inherently smaller output of the first layer and that solve *semantic problems* closer to biology (for example in transcriptomics and haplotype assembly). The concepts and methodologies detailed in this book are, however, a vantage point for designing external memory algorithms, parallel shared-memory algorithms, GPU algorithms, and distributed algorithms, whose importance is bound to increase in high-throughput sequencing, and in particular in metagenomics. Some exercises explicitly ask the reader to explore such directions, and the last chapter of the book suggestively ends by referring to an existing distributed algorithm, whose sequential version is described in the book. For readers interested in more general concepts at the intersection of big data, external memory, and distributed computing, we recommend the book by Paolo Ferragina (Ferragina, 2013).

Target audience

Since the book has a clear focus on algorithmic sequence analysis for high-throughput sequencing, the main audience consists in graduate students in bioinformatics, graduate students in computer science with a strong interest in molecular biology, and bioinformatics practitioners willing to master the algorithmic foundations of the field. For the last of these, the insights scattered throughout the book provide a number of techniques that can be of immediate use in practice. The structure of the book is strongly focused on applications; thus the book could be used as an introduction to biological sequence analysis and to high-throughput sequencing for the novice. Selected parts of the book could even be used in an introductory course on bioinformatics; however, such basic topics were not chosen to cover the whole of bioinformatics but just to give the minimal foundations required to understand the more advanced concepts that appear in later chapters. Our fresh presentation of theoretical topics in succinct data structures, and in biological applications of minimum-cost flow and dynamic programming, might also appeal to the specialist in algorithm design.

Changes for the second edition

Over the years readers have spotted 57 errors, which have now been fixed. One error required significant changes: Our design goal of managing with a minimal setup of

data structures backfired, as we avoided constant-time range minimum queries (RMQ) believing that a simpler technique would suffice for our purposes. Our workaround contained an error, which could not be fixed. So, the second edition now includes a detailed exposition of the RMQ data structure.

While Burrows–Wheeler techniques are still central to many applications, they have often been replaced by simpler methods like *hashing* in practical tools. This is mostly due to developments around *minimizers*. The second edition includes an introduction to these concepts, including also *Bloom filters* and the *Karp–Rabin* rolling hash. Alignment algorithms have been extended to include $O(kn)$ time approximate string matching, where k is the edit distance threshold and n is the length of the input. It is now possible to explain this algorithm in a self-contained manner, as we included the RMQ data structure. Also, an extension of alignments to cyclic graphs is now covered.

The last two parts of the book (Part IV, Genome-Scale Algorithms, and Part V, Applications) have gone through a significant logical rearrangement. Variant calling has been moved from Applications to form the introductory material of chapters Alignment-based genome analysis and Alignment-free genome analysis and comparison. The latter includes new sections on *de novo variant calling* and on *approximating Jaccard similarity with min-hash*. The chapter on Genome compression has been renamed Compression of genome collections to highlight that large sets of similar genomes are the dataset that is most amenable to such algorithms. It also includes a new section on *prefix-free parsing* for constructing the *run-length encoded Burrows–Wheeler transform*. Haplotype analysis now forms the first chapter of Part V; it includes a linear-time algorithm for *haplotype matching* and introduces the *positional Burrows–Wheeler transform*. Since variant calling has been moved to earlier chapters, the Genomics chapter is now called Pangenomics, and it includes an overview of pangenome representations such as *colored de Bruijn graphs* and *founder graphs*. A significant part of the Pangenomics chapter is devoted to the *r-index* – a data structure invented since the first edition that makes indexing repetitive genome collections practical. We also summarize some recent developments, especially in practical pangenome data structures. Such summaries take the form of seven new Insights that work as mini reviews, largely accounting for the 25% increase in the number of references covered.

Acknowledgments

See the first edition or the book webpage for the acknowledgments about the initial birth process. The selection of new content for this second edition was influenced by an online poll. We wish to thank every participant for the great suggestions. Some new topics were suggested by our anonymous reviewers, whom we thank also for spurring the idea of rearranging the content. Our description of prefix-free parsing is based on an excellent whiteboard explanation by Jarno Alanko. The cover illustration was created with the help of the TRAViz software (Jänicke et al., 2014).

Notation

Strings, arrays, sets

i, j, k, n	Integer numbers.
$[i..j]$, $\|[i..j]\|$	The set of integers $\{i, i+1, \ldots, j-1, j\}$ and its cardinality $j - i + 1$.
$\mathbb{I}(x, y)$, $\mathbb{I}(x)$	A function that returns the set of integers $[i..j]$ associated with the pair of objects (x, y). We use $\mathbb{I}(x)$ when y is clear from the context.
$\overrightarrow{x}, \overleftarrow{x}, \overline{x}, \|\overrightarrow{x}\|$	Assume that $\mathbb{I}(x, y) = [i..j]$, where both function $\mathbb{I}$ and object y are clear from the context. Then, $\overrightarrow{x} = [i..j]$, $\overleftarrow{x} = i$, $\overline{x} = j$, and $\|\overrightarrow{x}\| = \|[i..j]\|$.
$\Sigma = [1..\sigma]$	Alphabet of size σ. All integers in Σ are assumed to be used.
$\Sigma \subseteq [1..u]$	An ordered alphabet, in which not all integers in $[1..u]$ are necessarily used. We denote its size $\|\Sigma\|$ with σ.
a, b, c, d	*Characters*, that is, integers in some alphabet Σ. We also denote them with *symbols*.
$T = t_1 t_2 \cdots t_n$	A *string*, that is, a concatenation of characters in some alphabet Σ, with character t_i at position i. We use the term *sequence* in a biological context.
$T \cdot S$, $t \cdot s$	Concatenation of strings T and S or multiplication of integers t and s, with the operation type being clear from the context. We sometimes omit $\cdot$ if the operants of the concatenation/multiplication are simple.
$T =$ `ACGATAGCTA`	A *string*, with characters given explicitly and represented as letters of the English alphabet.
$\overleftarrow{T}$	The *reverse* of string T, i.e. string T read from right to left.
$\widetilde{T}$	The *reverse complement* of string T, that is, string T read from right to left, replacing `A` with `T` and `C` with `G`, and vice versa.
$T_{i..j}$	The *substring* $t_i t_{i+1} \cdots t_{j-1} t_j$ of string T induced by the indexes in $[i..j]$.
$T[i..j]$	Equivalent to $T_{i..j}$, used for clarity when i or j are formulas rather than variables.

subsequence	A string $t_{i_1}t_{i_2}\cdots t_{i_k}$ obtained by selecting a set of position $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ and by reading the characters of a string $T = t_1 t_2 \cdots t_n$ at those positions. In a biological context, subsequence is used as a synonym of substring.
$\mathcal{S} = \{T^1, T^2, \ldots, T^n\}$	Set of strings, with T^i denoting the ith string.
$\Sigma^*, \Sigma^+, \Sigma^n$	The set of all strings over alphabet Σ, the set of all non-empty strings over alphabet Σ, and the set of strings of length n over alphabet Σ, respectively. We use shorthand $A = a^n$ for $A \in \{a\}^n$, that is, for a string consisting of n occurrences of a.
$\delta(i..j, c)$	A function that maps an interval $[i..j]$ and a character $c \in \Sigma$ onto exactly one interval $[i'..j']$.
$\ldots, \#_2, \#_1, \#_0$	Shorthands for non-positive integers, with $\#_0 = 0$ and $\#_i = -i$.
$\#$	Shorthand for $\#_0$.
$\$_1, \$_2, \ldots$	Shorthands for positive integers greater than σ, with $\$_i = \sigma + i$.
$\$$	Shorthand for $\$_1$.
$A[1..n]$	Array A of integers, indexed from 1 to n.
$A[i..j]$, $A[\vec{x}]$	The subarray of array A induced by the indexes in $[i..j]$ and in $\vec{x}$, respectively.
$X = (p, s, v)$	Triplet of integers, with primary key $X.p$, secondary key $X.s$, and value $X.v$.
$D[1..m, 1..n]$	An array / matrix with m rows and n columns.
$D_{i_1..j_1, i_2..j_2}$	Subarray of D.
$D[i_1..j_1, i_2..j_2]$	Same as above.
$d_{i,j} = D[i, j]$	An element of the array D.

Undirected graphs

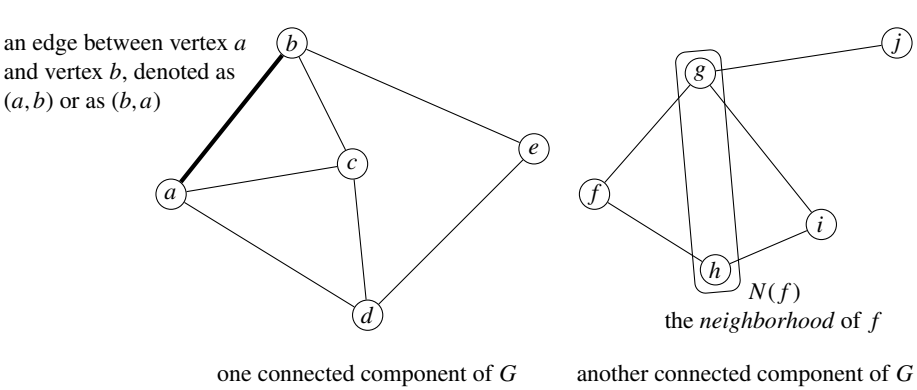

Figure 2 An *undirected graph* $G = (V, E)$, with *vertex* set V and *edge* set E.

$V(G)$	set V of vertices of a graph $G = (V, E)$
$E(G)$	set E of edges of an undirected graph $G = (V, E)$
$(x, y) \in E(G)$	an edge of an undirected graph G; the same as (y, x)
$N(x)$	the *neighborhood* of x in G, namely the set $\{y \mid (x, y) \in E(G)\}$

Directed graphs

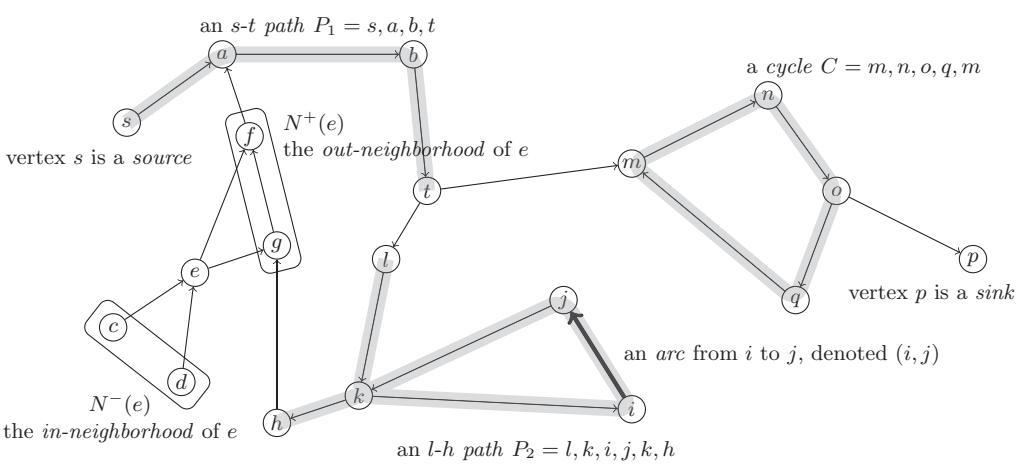

Figure 3 A *directed graph* $G = (V, E)$, with *vertex* set V and *arc* set E.

$(x, y) \in E(G)$	an arc of a directed graph G; the arc (x, y) is different from (y, x)
$N^-(x)$	the in-neighborhood of x in G, namely the set $\{y \mid (y, x) \in E(G)\}$
source	a vertex v is a source if $N^-(v) = \emptyset$
$N^+(x)$	the out-neighborhood of x in G, namely the set $\{y \mid (x, y) \in E(G)\}$
sink	a vertex v is a sink if $N^+(v) = \emptyset$
$P = v_1, \ldots, v_k$	a path in G, namely a sequence of vertices of G connected by arcs with the same orientation, from v_1 to v_k; depending on the context, we allow or not P to have repeated vertices
s-t path	path from vertex s to vertex t
$C = v_1, \ldots, v_k, v_1$	a cycle in G, namely a path in G in which the first and last vertex coincide; depending on the context, we allow or not C to have other repeated vertices than its first and last elements
$(x, y) \in S$	arc $(x, y) \in E(G)$ appears on S, where S is a path or cycle of G

Trees

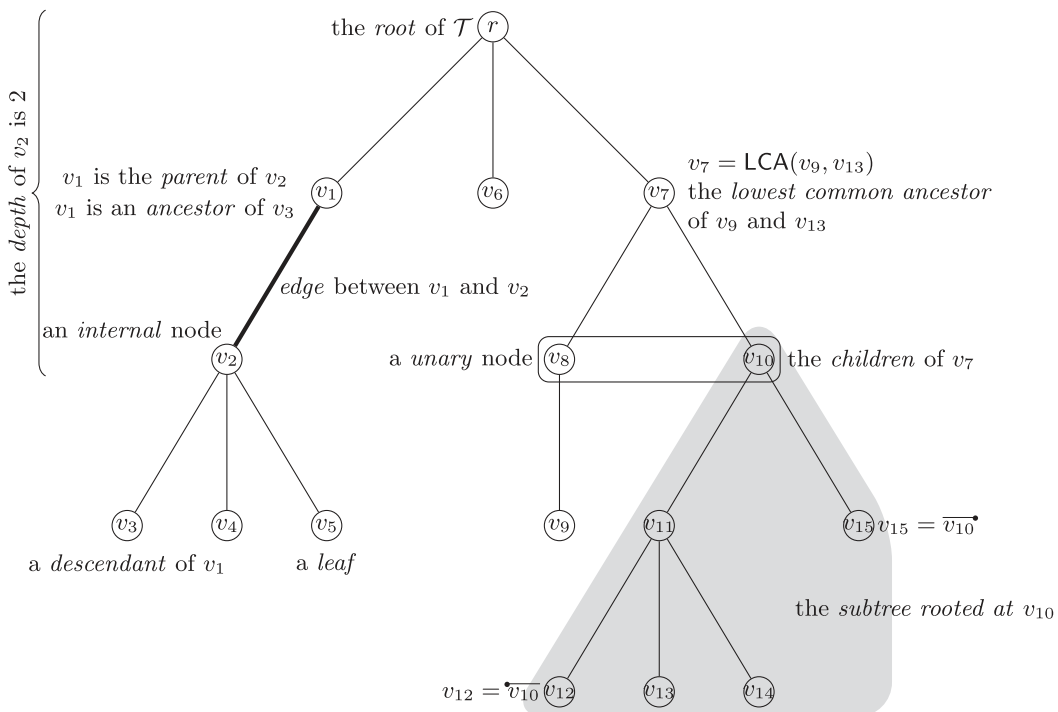

Figure 4 A tree $\mathcal{T} = (V, E)$, with *node* set V and *edge* set E. Unless otherwise stated, we assume all trees to be *ordered*, that is, we assume that there is a total order on the children of every node.

v_2 is a child of v_1	if there is an edge between v_1 and v_2 and v_1 appears on the path from the root to v_2
v_1 is the parent of v_2	if v_2 is the child of v_1
degree of a node v	the number of children of v
leaf	a node with degree 0
internal node	a node with degree at least one
unary node	a node with degree 1
depth of v	the number of edges of the path from the root to v
subtree rooted at v	the subtree of $\mathcal{T}$ having root v and consisting of all nodes reachable through a path starting at v made up only of nodes of depth at least the depth of v
v_2 is descendant v_1	if $v_2 \neq v_1$ belongs to the subtree rooted at v_1
v_1 is ancestor v_2	if $v_2 \neq v_1$ belongs to the subtree rooted at v_1
$\mathsf{LCA}(v_1, v_2)$	the *lowest common ancestor* of v_1 and v_2, that is, the deepest node which is an ancestor of both v_1 and v_2
$\overline{v}$	the left-most leaf of the subtree rooted at a node v
$\overline{v}$	the right-most leaf of the subtree rooted at a node v