Genome-Scale Algorithm Design

Biological Sequence Analysis in the Era of High-Throughput Sequencing

High-throughput sequencing has revolutionized the field of biological sequence analysis. Its application has enabled researchers to address important biological questions, often for the first time.

This book provides an integrated presentation of the fundamental algorithms and data structures that power modern sequence analysis workflows. The topics covered range from the foundations of biological sequence analysis (alignments and hidden Markov models), to classical index structures (k-mer indexes, suffix arrays, and suffix trees), Burrows–Wheeler indexes, graph algorithms, and a number of advanced omics applications. The chapters feature numerous examples, algorithm visualizations, exercises, and problems, each chosen to reflect the steps of large-scale sequencing projects, including read alignment, variant calling, haplotyping, fragment assembly, alignment-free genome comparison, transcript prediction, and analysis of metagenomic samples. Each biological problem is accompanied by precise formulations and complexity analyses, providing graduate students and researchers in bioinformatics and computer science with a powerful toolkit for the emerging applications of high-throughput sequencing.

The book is accompanied by a website (www.genome-scale.info) offering LaTeX source files for the exercises, along with relevant links.

Veli Mäkinen is a Professor of Computer Science at the University of Helsinki, Finland, where he heads a research group working on genome-scale algorithms as part of the Finnish Center of Excellence in Cancer Genetics Research. He has taught advanced courses on string processing, data compression, biological sequence analysis, along with introductory courses on bioinformatics.

Djamal Belazzougui is a postdoctoral researcher at the University of Helsinki. His research topics include hashing, succinct and compressed data structures, and string algorithms.

Fabio Cunial is a postdoctoral researcher at the University of Helsinki. His research focuses on string algorithms and genome analysis.

Alexandru I. Tomescu is a postdoctoral researcher at the University of Helsinki. His current research interests lie at the intersection of computational biology and computer science.

Genome-Scale Algorithm Design

Biological Sequence Analysis in the Era of High-Throughput Sequencing

VELI MÄKINEN
DJAMAL BELAZZOUGUI
FABIO CUNIAL
ALEXANDRU I. TOMESCU
University of Helsinki, Finland

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107078536

First published 2015

Printed in the United Kingdom by TJ International Ltd. Padstow Cornwall

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

Library of Congress Cataloging in Publication data
Mäkinen, Veli, author.
Genome-scale algorithm design : biological sequence analysis in the era of high-throughput sequencing / Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu.
p. ; cm.
Includes bibliographical references and index.
ISBN 978-1-107-07853-6 (hardback : alk. paper)
I. Belazzougui, Djamal, author. II. Cunial, Fabio, author. III. Tomescu, Alexandru I., author. IV. Title.
[DNLM: 1. Genomics. 2. Algorithms. 3. Sequence Analysis–methods. QU 460]
QH447
572.8′629–dc23 2014045039

ISBN 978-1-107-07853-6 Hardback

Contents

Notation *page* xii
Preface xvii

Part I Preliminaries 1

1 Molecular biology and high-throughput sequencing 3
1.1 DNA, RNA, proteins 3
1.2 Genetic variations 6
1.3 High-throughput sequencing 7
Exercises 9

2 Algorithm design 10
2.1 Complexity analysis 10
2.2 Data representations 12
2.3 Reductions 13
2.4 Literature 17
Exercises 17

3 Data structures 20
3.1 Dynamic range minimum queries 20
3.2 Bitvector rank and select operations 22
3.3 Wavelet tree 24
3.3.1 Balanced representation 24
3.3.2 Range queries 26
3.4 Literature 27
Exercises 27

4 Graphs 30
4.1 Directed acyclic graphs (DAGs) 30
4.1.1 Topological ordering 30
4.1.2 Shortest paths 31

4.2 Arbitrary directed graphs 33
4.2.1 Eulerian paths 33
4.2.2 Shortest paths and the Bellman–Ford method 34
4.3 Literature 38
Exercises 38

5 Network flows 41

5.1 Flows and their decompositions 41
5.2 Minimum-cost flows and circulations 45
5.2.1 The residual graph 47
5.2.2 A pseudo-polynomial algorithm 50
5.3 Bipartite matching problems 51
5.3.1 Perfect matching 52
5.3.2 Matching with capacity constraints 54
5.3.3 Matching with residual constraints 56
5.4 Covering problems 58
5.4.1 Disjoint cycle cover 58
5.4.2 Minimum path cover in a DAG 60
5.5 Literature 64
Exercises 65

Part II Fundamentals of Biological Sequence Analysis 69

6 Alignments 71

6.1 Edit distance 72
6.1.1 Edit distance computation 73
6.1.2 Shortest detour 76
*6.1.3 Myers' bitparallel algorithm 78
6.2 Longest common subsequence 83
6.2.1 Sparse dynamic programming 84
6.3 Approximate string matching 86
6.4 Biological sequence alignment 88
6.4.1 Global alignment 89
6.4.2 Local alignment 90
6.4.3 Overlap alignment 92
6.4.4 Affine gap scores 94
6.4.5 The invariant technique 97
6.5 Gene alignment 98
6.6 Multiple alignment 101
6.6.1 Scoring schemes 101
6.6.2 Dynamic programming 103
6.6.3 Hardness 103
6.6.4 Progressive multiple alignment 104

6.6.5 DAG alignment 105
6.6.6 Jumping alignment 107
6.7 Literature 108
Exercises 109

7 Hidden Markov models (HMMs) 113
7.1 Definition and basic problems 114
7.2 The Viterbi algorithm 118
7.3 The forward and backward algorithms 118
7.4 Estimating HMM parameters 120
7.5 Literature 122
Exercises 123

Part III Genome-Scale Index Structures 127

8 Classical indexes 129
8.1 *k*-mer index 129
8.2 Suffix array 132
8.2.1 Suffix and string sorting 133
8.3 Suffix tree 140
8.3.1 Properties of the suffix tree 142
8.3.2 Construction of the suffix tree 143
8.4 Applications of the suffix tree 145
8.4.1 Maximal repeats 145
8.4.2 Maximal unique matches 147
8.4.3 Document counting 149
8.4.4 Suffix–prefix overlaps 151
8.5 Literature 151
Exercises 153

9 Burrows–Wheeler indexes 157
9.1 Burrows–Wheeler transform (BWT) 158
9.2 BWT index 160
9.2.1 Succinct LF-mapping 160
9.2.2 Backward search 162
9.2.3 Succinct suffix array 163
9.2.4 Batched locate queries 165
*9.3 Space-efficient construction of the BWT 166
9.4 Bidirectional BWT index 171
*9.4.1 Visiting all nodes of the suffix tree with just one BWT 175
*9.5 BWT index for labeled trees 177
*9.5.1 Moving top-down 179
*9.5.2 Moving bottom-up 181

*9.5.3 Construction and space complexity 182
*9.6 BWT index for labeled DAGs 182
*9.6.1 Moving backward 185
*9.6.2 Moving forward 186
*9.6.3 Construction 187
9.7 BWT indexes for de Bruijn graphs 188
9.7.1 Frequency-oblivious representation 190
9.7.2 Frequency-aware representation 192
9.7.3 Space-efficient construction 193
9.8 Literature 194
Exercises 196

Part IV Genome-Scale Algorithms 199

10 Read alignment 201
10.1 Pattern partitioning 202
10.2 Dynamic programming along suffix tree paths 204
10.3 Backtracking on BWT indexes 204
10.3.1 Prefix pruning 206
10.3.2 Case analysis pruning with the bidirectional BWT index 208
10.4 Suffix filtering for approximate overlaps 209
10.5 Paired-end and mate pair reads 211
10.6 Split alignment of reads 212
10.7 Alignment of reads to a pan-genome 214
10.7.1 Indexing a set of individual genomes 214
*10.7.2 Indexing a reference genome and a set of variations 215
10.8 Literature 216
Exercises 217

11 Genome analysis and comparison 220
11.1 Space-efficient genome analysis 221
11.1.1 Maximal repeats 221
11.1.2 Maximal unique matches 223
11.1.3 Maximal exact matches 225
11.2 Comparing genomes without alignment 229
11.2.1 Substring and k-mer kernels 232
*11.2.2 Substring kernels with Markovian correction 238
11.2.3 Substring kernels and matching statistics 244
11.2.4 Mismatch kernels 251
11.2.5 Compression distance 253
11.3 Literature 255
Exercises 256

12 Genome compression 262

12.1 Lempel–Ziv parsing 263
12.1.1 Basic algorithm for Lempel–Ziv parsing 264
12.1.2 Space-efficient Lempel–Ziv parsing 265
*12.1.3 Space- and time-efficient Lempel–Ziv parsing 266
*12.2 Bit-optimal Lempel–Ziv compression 270
*12.2.1 Building distance-maximal arcs 275
*12.2.2 Building the compact trie 278
12.3 Literature 279
Exercises 280

13 Fragment assembly 282

13.1 Sequencing by hybridization 282
13.2 Contig assembly 284
13.2.1 Read error correction 285
13.2.2 Reverse complements 286
13.2.3 Irreducible overlap graphs 287
13.3 Scaffolding 291
13.4 Gap filling 297
13.5 Literature 299
Exercises 301

Part V Applications 305

14 Genomics 307

14.1 Variation calling 308
14.1.1 Calling small variants 308
14.1.2 Calling large variants 309
14.2 Variation calling over pan-genomes 313
14.2.1 Alignments on a set of individual genomes 313
14.2.2 Alignments on the labeled DAG of a population 314
14.2.3 Evaluation of variation calling results 315
14.3 Haplotype assembly and phasing 315
14.4 Literature 322
Exercises 323

15 Transcriptomics 325

15.1 Estimating the expression of annotated transcripts 325
15.2 Transcript assembly 329
15.2.1 Short reads 329
15.2.2 Long reads 330
15.2.3 Paired-end reads 335

15.3 Simultaneous assembly and expression estimation 337
15.4 Transcript alignment with co-linear chaining 342
15.5 Literature 345
Exercises 346

16 Metagenomics 350

16.1 Species estimation 351
16.1.1 Single-read methods 351
16.1.2 Multi-read and coverage-sensitive methods 353
16.2 Read clustering 357
16.2.1 Filtering reads from low-frequency species 357
16.2.2 Initializing clusters 359
16.2.3 Growing clusters 363
16.3 Comparing metagenomic samples 364
16.3.1 Sequence-based methods 365
16.3.2 Read-based methods 365
16.3.3 Multi-sample methods 366
16.4 Literature 366
Exercises 367

References 370
Index 386

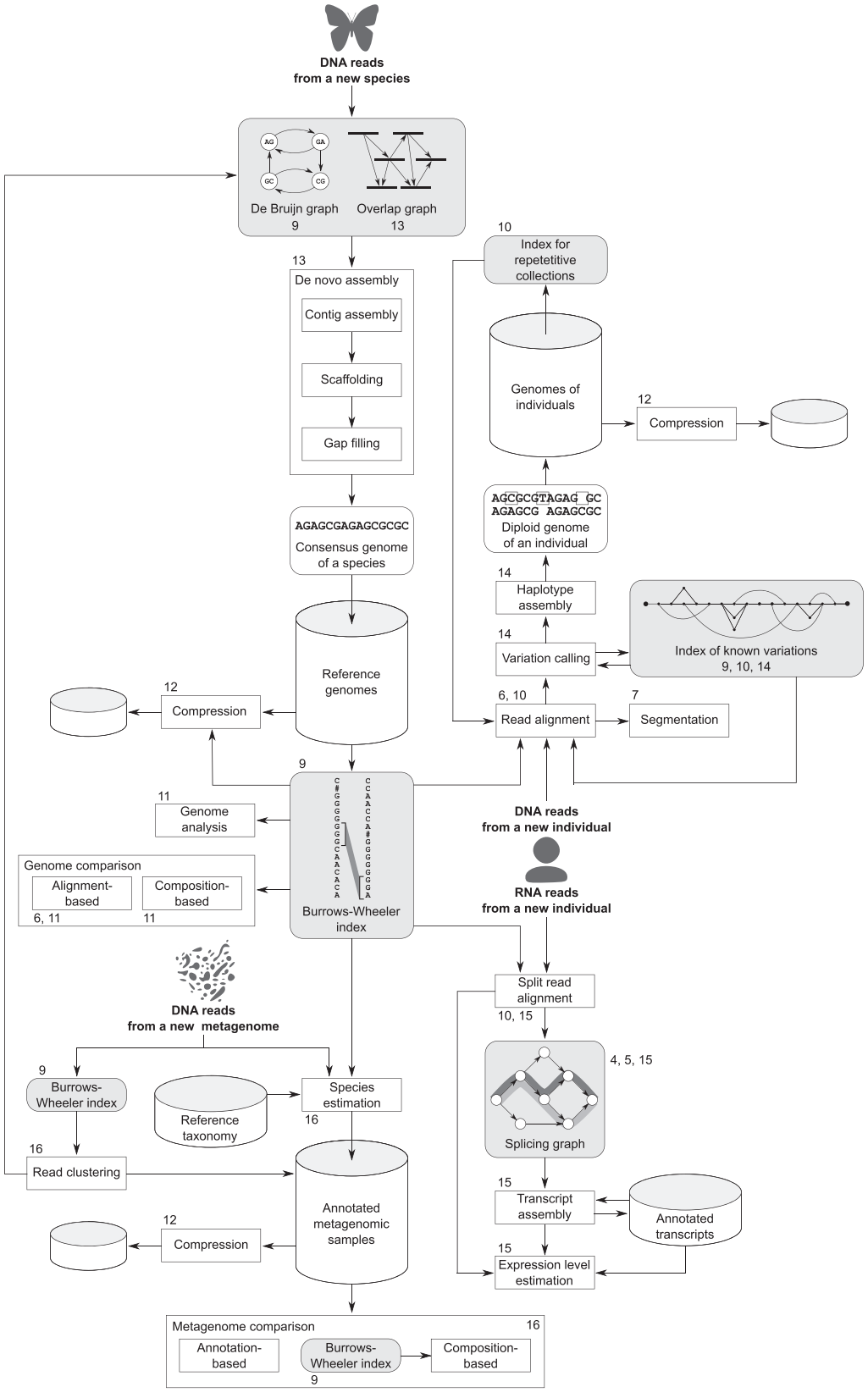

Figure 1 High-level summary of the main computational steps in high-throughput sequencing. Key data structures are highlighted in gray. Cylinders represent databases. Numbers indicate the chapters that cover each step.

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.
$\overleftarrow{x}, \overleftarrow{x}, \vec{x}, \|\vec{x}\|$	Assume that $\mathbb{I}(x, y) = [i..j]$, where both function $\mathbb{I}$ and object y are clear from the context. Then, $\overleftarrow{x} = [i..j]$, $\overleftarrow{x} = i$, $\vec{x} = j$, and $\|\vec{x}\| = \|[i..j]\|$.
$\Sigma = [1..\sigma]$	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 call them *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 operands of the concatenation/multiplication are simple.
$T = \texttt{ACGATAGCTA}$	A *string*, with characters given explicitly and represented as letters of the English alphabet.
$\underline{T}$	The *reverse* of a string T, i.e. string T read from right to left.
$\mathcal{T}$	The *reverse complement* of a DNA sequence 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 positions $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ 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

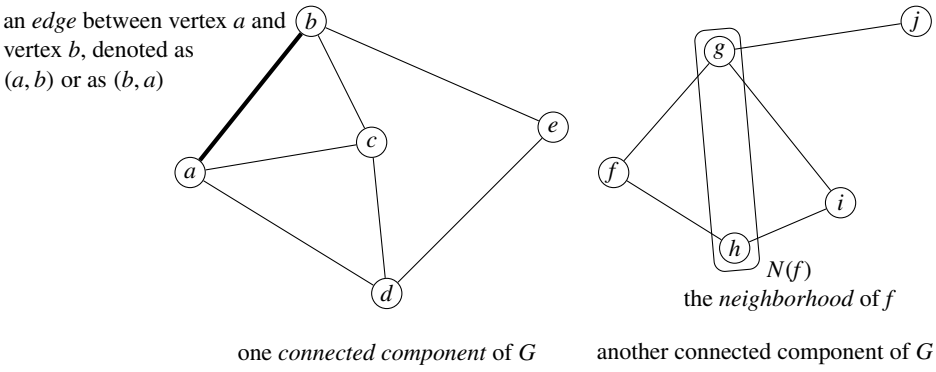

Figure 2 An *undirected graph* $G = (V, E)$, with *vertex* set V and *edge* set E.

$V(G)$	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

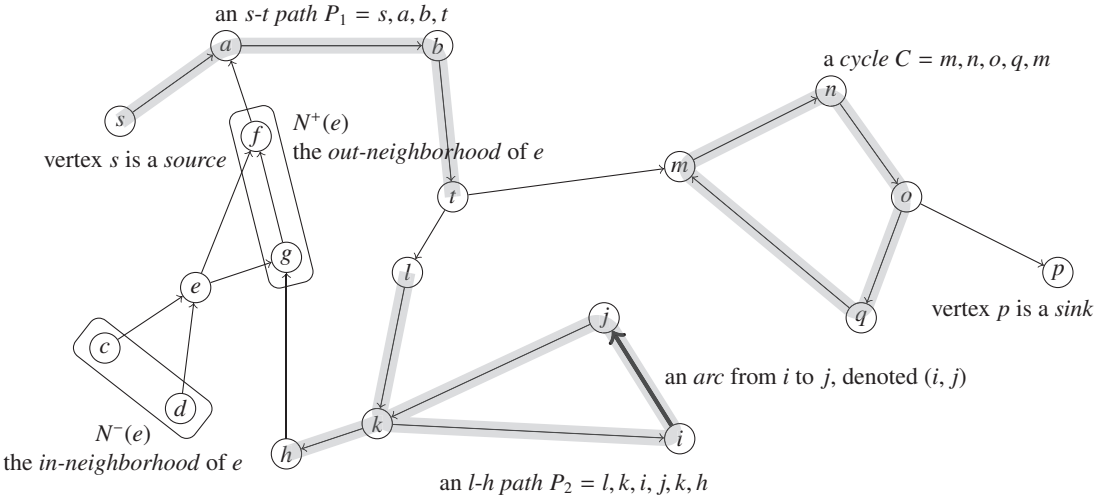

Figure 3 A *directed graph* $G = (V, E)$, with *vertex* set V and *arc* set E.

$(x, y) \in E(G)$	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 do not allow 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

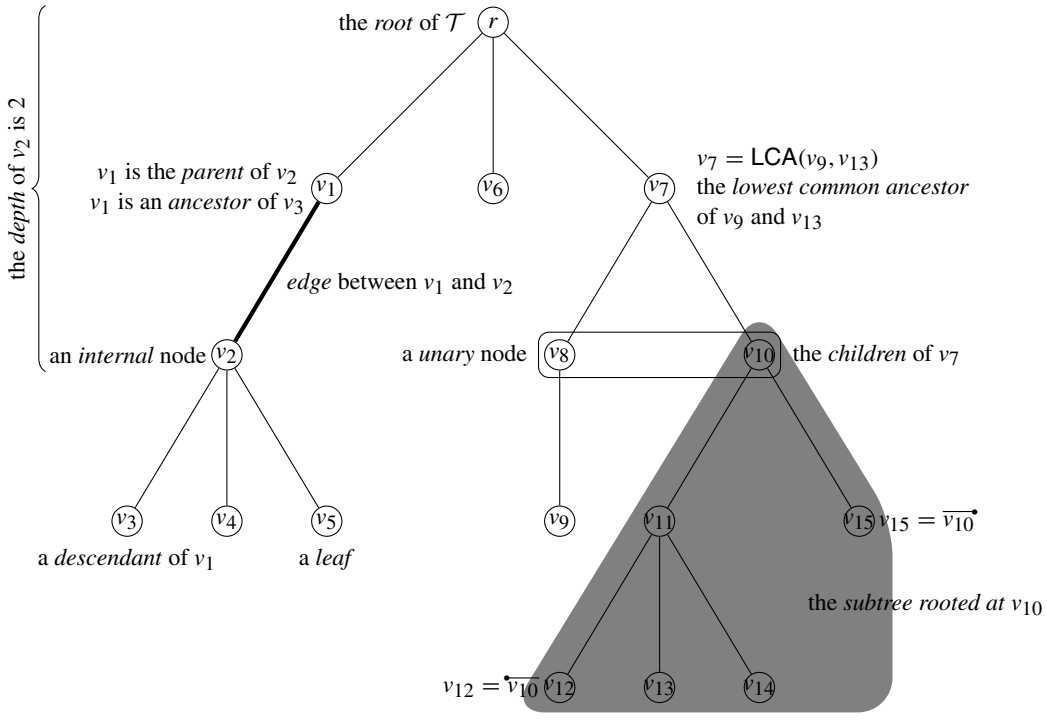

Figure 4 A tree $\mathcal{T} = (V, E)$, with *node* set V and *edge* set E. Unless stated otherwise, we assume all trees to be *ordered*, that is, we assume that there is a total order on the children of every node.

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 1.
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.

Preface

Background

High-throughput sequencing has recently 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. Most 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 ten 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, 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 modern 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 to 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 implementations of many index structures and algorithms described here. The website also maintains a list of typos and mistakes found, and we encourage the reader to send corrections as requested therein.

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' bitparallel 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 pseudocode 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 key 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 insights that describe algorithms whose worst-case complexity is not interesting.

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 20 years, and this field would deserve a textbook on its own. 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, entropy-compressed data structures, and data structures to support constant time range minimum queries. These choices resulted in some new insights on algorithm design. Namely, we observed that one can still obtain optimal solutions to a number of suffix tree problems by resorting to amortized analysis on batched queries solvable without the need for any advanced basic data structure. A noteworthy new result in the book exemplifying this sufficiency is that Lempel–Ziv factorization can be obtained space-efficiently in near-linear time with the chosen minimal data structure setup.

Owing to the wide adaptation of the algorithms appearing in the literature to our self-contained minimal data structure setup, we decided to gather all of the references into a literature section appearing at the end of each chapter. This enables an undisturbed text flow and lets us explain the difference between our description and the original work. Some exercises are also directly related to some published work, and in such cases we mention the reference in the literature section. With regard to new results developed for the book or work under preparation, we added some future references in the literature sections.

Finally, 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 *semantical 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 secondary-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.

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 latter, 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 in 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 recent 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.

Acknowledgements

First and foremost, we wish to warmly thank our families and friends for their constant support throughout the sometimes laborious process of writing a book. We apologize once again for the long nights spent at work, and for frequent periods of pressure and discouragement that they helped us endure. We are also profoundly grateful to the editors, Katrina Halliday and Megan Waddington, for their constant encouragements – particularly important to keep us motivated at the beginning of the writing process. Our deep gratitude goes also to Gonzalo Navarro, Nadia Pisanti, and Romeo Rizzi, for volunteering to read major parts of the book, and for providing invaluable comments on it. Selected chapters were also reviewed by Carlo Comin, Emanuele Giaquinta, Anna Kuosmanen, Paul Medvedev, Martin Milanič, Alberto Policriti, Nicola Prezza, Kristoffer Sahlin, and Leena Salmela. Some chapters were in a quite early phase during such reviews, and we probably introduced new errors during the last weeks of intensive writing, so the reviewers should not be held responsible for any error that slipped through. Veli Mäkinen wishes also to thank the Finnish Cultural Foundation for supporting the sabbatical year that enabled his full concentration on the book.

The selection of content and structure on edit distances, alignments, hidden Markov models, and text indexing was highly influenced by the *String Processing Algorithms*, *Data Compression Techniques*, and *Biological Sequence Analysis* courses taught repeatedly at the Department of Computer Science, University of Helsinki, starting in the nineties with Esko Ukkonen, Jorma Tarhio (now at Aalto University), and Juha Kärkkäinen, and occasionally inherited by other colleagues, including Veli Mäkinen. The seed of this book consisted in a lecture script of some 150 pages by Veli, which gradually emerged in multiple rounds of lectures for the *Biological Sequence Analysis* course at the University of Helsinki, for the *Genome-Scale Algorithmics* course at the University of Bielefeld, and for a tutorial on *Genome-Scale Sequence Analysis Using Compressed Index Structures* at ISMB 2009. The students of the 2015 edition of the Biological Sequence Analysis course gave valuable feedback on a few chapters of the preprint of this book.

However, this initial draft was then heavily rewritten, restructured, and extended, capitalizing on the heterogeneous educational backgrounds of the authors to broaden its original scope. In the end, 11 out of the 16 chapters of this book were written from scratch, while the other 5 were heavily reworked. Some of the new chapters are based on some distinctive expertise that each one of us brought to the book. For example, Djamal brought in competence on succinct data structures, Fabio on genome analysis methods, and Alexandru on graph theory and computational complexity. As unexpected as it might sound, the very experience of writing a book together contributed to establishing a genuine friendship. We also thank our academic advisors, as well

as all our teachers, research colleagues, and students, for their invaluable contribution in enriching our knowledge – as well as our lives. Finally, our understanding of high-throughput sequencing, and of the computational problems that underpin it, had been greatly increased by collaborating with colleagues inside the Finnish Center of Excellence in Cancer Genetics Research, led by Lauri Aaltonen, inside the *Melitaea cinxia* sequencing project, led by Ilkka Hanski, and inside the HIIT bio-focus area on metagenomics, led by Antti Honkela.

Sometimes chance events have long-term consequences. While visiting our research group at the University of Helsinki, Micheaël Vyverman, from Ghent University, gave a talk on maximal exact matches. Veli's lecture script already contained Algorithm 11.3, which uses the bidirectional BWT index to solve this problem, but without any analysis of its time and space complexity. Micheaël's talk inspired us to study this algorithm with more care: soon we realized that its running time is $O(n \log \sigma)$, but the use of a stack caused a space problem. Luckily, Juha Kärkkäinen happened to hear our discussions, and directed us to a classical stack-trick from the quicksort algorithm, which solved our space problem. After even more thinking, we realized that the algorithm to solve maximal exact matches could be adapted to solve a large number of seemingly unrelated problems in sequence analysis, essentially becoming Algorithm 9.3, one of the centerpieces of a book that was still in its infancy. Later we learnt that the related strategy of implicit enumeration of the internal nodes of a suffix tree with bidirectional Burrows–Wheeler index had already been developed by Enno Ohlebusch and his colleagues.

Another random event happened when we were working on a problem related to RNA transcript assembly. We had sent Romeo Rizzi a submitted manuscript containing an NP-hardness proof of a certain problem formulation. After some time, Romeo replied with a short message claiming he had a polynomial-time algorithm for our problem, based on minimum-cost flows. After a stunned period until we managed to get into contact, it of course turned out that we were referring to different problems. The NP-hard formulation is now presented in Exercise 15.11, and the problem Romeo was initially referring to is Problem 15.5. Nonetheless, this initial connection between assembly problems and minimum-cost flow solutions led to the many applications of minimum-cost flow throughout this book.

While we were working hard on this project, Travis Gagie and Simon Puglisi were working even harder to keep the research productivity of the group at a high level. Simon kept the group spirit high and Travis kindly kept Daniel Valenzuela busy with spaced suffix arrays while there was yet another book meeting.

Many students got the opportunity to implement our ideas. These implementations by Jarno Alanko, Jani Rahkola, Melissa Riesner, Ahmed Sobih, and many others can be found at the book webpage.